Parsing of Math Formulas and Chemical Diagrams using Graph-Based Representation and Attention Models

by

Ayush Kumar Shah

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
June 2025

Parsing of Math Formulas and Chemical Diagrams using Graph-Based Representation and Attention Models

by

Ayush Kumar Shah

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Dr. Richard Zanibbi — Date
Dissertation Advisor

Dr. Qi Yu — Date
Dissertation Committee Member

Dr. Weijie Zhao — Date
Dissertation Committee Member

Dr. Masaki Nakagawa — Date
Dissertation Committee Member

Dr. Dan Phillips — Date
Dissertation Defense Chairperson

Certified by:

Dr. Pengcheng Shi — Date
Ph.D. Program Director, Computing and Information Sciences

Parsing of Math Formulas and Chemical Diagrams using Graph-Based Representation and Attention Models

by

Ayush Kumar Shah

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Mathematical formulas and chemical diagrams appear frequently in scientific documents but are often embedded as visual content, either rasterized or vector-based images, limiting their accessibility and automated analysis. This thesis aims to bridge this gap by presenting a graph-based visual parsing framework that recognizes and parses these notations from both vector and raster image inputs in digital documents.

For mathematical formulas in born-digital PDFs, we construct Symbol Layout Trees (SLTs) using a graph defined over vector-based primitives, capturing spatial relationships, avoiding relying on OCR. For born-digital chemical diagrams, we introduce a Minimum Spanning Tree (MST)-based technique that extracts molecular structure graphs by interpreting vector graphics using domain-specific spatial and symbolic constraints.

To parse rasterized images, we develop a multi-task, segmentation-aware neural network that operates on over-segmented visual primitives extracted via line segment detection and watershed-based segmentation. We create annotated training data by aligning vector-based ground truth with detected visual primitives in raster images. The model jointly performs symbol classification, segmentation, and relationship classification in a multi-task learning framework, utilizing discrete attention mechanisms to dynamically modify input features over iterative passes. We enhance robustness using synthetic structural and visual noise applied at the primitive level to simulate degradations in real document images and mitigate class imbalance through stratified sampling and loss reweighting strategies, including weighted cross-entropy, class-balanced and focal losses. We introduce a two-stage graph attention model to support cross-task learning, where class distributions from the first stage are used to inform refinement in the second.

Evaluation metrics compare nodes and edges in the predicted graphs to ground truth using adjacency matrices and Hamming distances to quantify structural and labeling errors. The results and analysis across mathematical and chemical datasets show that (1) input line-of-sight (LOS) graph representation improves expression coverage (the upper bound on the number of expressions that can be correctly parsed) and reduce number of edge hypotheses for math, while 6 nearest-neighbor (6NN) graphs are better suited for chemistry due to their local structure, (2) attention mechanisms and cross task interaction enhance structural prediction, (3) primitive-level noise augmentation and loss rebalancing and aggregation improve generalization across input conditions. Together, these findings support the development of a unified and extensible framework for visual parsing of structured scientific notations across domains.

Acknowledgments

I am deeply grateful to my advisor, Dr. Richard Zanibbi, for his invaluable guidance, encouragement, and support throughout my research journey. His expertise and insightful feedback have greatly shaped my work and inspired me to achieve new heights. I am also profoundly thankful to Dr. Scott Denmark and his student, Blake Ocampo for their collaboration, insightful discussions, and contributions that helped enrich my work on chemical diagram parsing.

I would like to extend my heartfelt thanks to the members of the Document and Pattern Recognition Lab (DPRL) at Rochester Institute of Technology (RIT) for their invaluable feedback, collaborative spirit, and stimulating discussions. Special thanks to Matt Langsenkamp, Abhisek Dey, Bryan Amador, Ming Creekmore, Brandon Kirincich, and Patrick Philippy for their support, and contributions during this research journey.

I am also deeply grateful to Matthew Berry, Yashdeep Thorat, Sara Lambert, and Kate Arneson, along with other members of the National Center for Supercomputing Applications (NCSA), for their assistance, technical support, user feedbacks, and insightful contributions to the ChemScraper system.

My sincere gratitude goes to my dissertation committee members: Dr. Masaki Nakagawa, Dr. Qi Yu, Dr. Weijie Zhao, for their invaluable feedback, encouragement, and constructive suggestions, which have been instrumental in shaping this dissertation proposal. I would also like to thank the defense chair, Dr. Dan Phillips for reviewing the work.

This work was made possible through the support of the Alfred P. Sloan Foundation under Grant No. G-2017-9827 and the National Science Foundation (USA) under Grant Nos. IIS-1717997 (MathSeer project) and 2019897 (Molecule Maker Lab Institute project). I am deeply appreciative of this support, which has facilitated my research and collaborations.

Finally, I would like to express my gratitude to all colleagues, mentors, and collaborators who have, directly or indirectly, supported my research endeavors.

Co-Authorship

Chapter 3 includes work on the original Symbol Scraper, which was initially coded and written by Alexander Keller, Ritvik Joshi, Jessica Diehl, and Dr. Richard Zanibbi. Additionally, the first version of the born-digital parser for chemical diagrams, also discussed in Chapter 3, was coded and written by Ming Creekmore and Abhisek Dey. The sections on visual primitive extraction and evaluation metrics in Chapter 2 were coded and written in collaboration with Bryan Amador.

The math formula parsers presented in Chapter 3 and Chapter 4 previously appeared as papers at the International Conference on Document Analysis and Recognition (ICDAR) in September 2021 and August 2023, respectively, coauthored by Abhisek Dey and Dr. Richard Zanibbi. The chemical diagram parsers discussed in Chapter 3 and Chapter 4 collectively appeared in the International Journal of Document Analysis and Recognition (Volume 27, September 2024), coauthored by Bryan Amador, Abhisek Dey, Ming Creekmore, Blake Ocampo, Dr. Scott Denmark and Dr. Richard Zanibbi.

For my parents, Ram Adhar Sah and Pratibha Shah.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parsing is the process of analyzing and extracting meaningful information from data. It plays an important role across a wide range of domains including natural language processing, computer vision, speech recognition, and data mining. It enables computers to interpret structured and unstructured data, transforming complex inputs into meaningful information. Document parsing, a subset of this broader field, focuses on extracting and interpreting content from structured documents such as books, scientific articles, web pages, and technical manuals. Document parsing has become a vital tool for making information accessible and searchable, enhancing digital libraries, automating data extraction workflows, and enabling intelligent retrieval of knowledge across various domains.

Among the most challenging tasks in document parsing is the recognition and interpretation of mathematical formulas and chemical diagrams. These notations are prevalent in scientific literature, technical reports, and patents, serving as a rich source of domain-specific knowledge. However, despite their ubiquity and significance, mathematical and chemical notations are typically embedded in documents as images, vector graphics, or other non-machine-readable formats. Hence, most retrieval systems only accept text queries, and cannot handle structures like mathematical formulas or chemical diagrams. This lack of accessibility creates a barrier to automatic indexing, retrieval, and analysis of scientific and technical knowledge. Consequently, there is a pressing need for effective recognition and interpretation methods for these diagrams to enable their integration into computer-readable forms. This would facilitate easy access, searchability, and interactivity across various platforms, including digital libraries, educational technologies, chemical informatics, and more. To aid in the development of search engines that can process queries containing mathematical notation [5, 34, 74, 90, 153, 158] or chemical diagrams, parsing them is needed for indexing formulas and

diagrams in documents, and to support handwritten input of formulas in queries [73, 81, 109, 140].

1.1 Motivation and Applications

The motivation for this research stems from the role that mathematical and chemical diagrams play in scientific communication and the challenges associated with their effective recognition and interpretation. Mathematical formulas encode complex expressions, equations, and relationships that are central to fields such as mathematics, physics, engineering, and computer science. Similarly, chemical diagrams capture detailed molecular structures, reaction mechanisms, and chemical bonds, serving as the cornerstone of disciplines such as chemistry, pharmacology, and materials science. These graphical elements hold valuable information essential for research, education, and industry applications.

The scientific literature holds vast amounts of useful chemical and mathematical information, but much of it remains locked within non-machine-readable formats. This makes it challenging to retrieve, analyze, and utilize such information effectively. Current manual processes for extracting and analyzing chemical data or mathematical expressions are labor-intensive and error-prone. Automating these tasks through advanced parsing techniques can save significant time and labor costs while helping in research and innovation.

There is a growing interest in developing techniques to extract information from graphics within documents because not all content can be effectively retrieved using text alone. Effective recognition and parsing of mathematical and chemical diagrams have broad implications across multiple domains including:

- **Enhancement of Chemical Informatics:** Automating the extraction and recognition of molecular structures, reaction data, and related content from chemical diagrams accelerates chemical research and synthesis processes. This has implications for drug discovery, material design, and the broader field of chemical synthesis.
- **Development of Graphics-Aware Search Engines:** By enabling search engines to process queries involving graphical elements, such as mathematical symbols or molecular diagrams, new opportunities for information retrieval and knowledge discovery are unlocked. Users can retrieve related content based on visual patterns, expanding the possibilities of scientific and technical searches.

- **Advancements in Educational Technology and Digital Libraries:** Parsing and integrating graphical content into digital platforms improves accessibility, for example, by enabling screen readers to interpret formulas and diagrams, and enhances searchability by allowing content to be indexed and retrieved based on visual and structural features. This supports more effective learning environments for students, educators, and researchers.
- **Support for Academic Research:** Providing tools that automatically parse complex graphical content frees researchers from manual data extraction tasks, allowing them to focus on data analysis, hypothesis generation, and scientific advancement.

Consider a scenario in chemical informatics where a student encounters a chemical diagram of aspirin (acetylsalicylic acid) and seeks to learn about catalysts that can enhance its yield or identify reactions that produce this compound. Answering such queries requires extracting, parsing, and recognizing molecular diagrams and reaction pathways, a task that would be greatly facilitated by automated tools. Similarly, imagine a researcher wanting more information about a complex mathematical formula they found in a publication. Accurate recognition, segmentation, and parsing of mathematical symbols and relationships would enable efficient searches and comparisons across scientific literature, as well as reuse in their own documents and tools (e.g., computer algebra systems).

The development of robust parsers that can handle both mathematical and chemical diagrams is important for transforming how information is accessed and used in scientific workflows. This research aims to address this need by leveraging (1) input graph representations, (2) graph attention-driven methods including edge-aware attention via GAT [129], and (3) multi-task learning and interaction approaches, and (4) strategies to handle class imbalance and noise, which are tailored to the unique characteristics of these domains. Graph attention-driven methods, such as GAT [129], allow nodes to attend selectively to their neighbors based on learned attention weights, enabling more expressive and context-aware representations of local structure. Multi-task learning and interaction approaches allow the model to jointly optimize related tasks such as symbol classification, segmentation, and relationship identification, while enabling shared information flow across tasks to improve generalization. In parallel, we explore techniques to mitigate class imbalance and simulate realistic noise to improve robustness and real-world applicability.

1.2 Parsing Mathematical Formulas and Chemical Diagrams

We propose developing efficient and interpretable parsers for mathematical and chemical formulas using graph-based methods and attention-driven techniques. Two key design goals in this work include: (1) *efficiency*, measured in terms of both computational runtime and the amount of training data required; and (2) *interpretability*, by maintaining input–output correspondence at the level of primitives, enabling error analysis directly over nodes and edges in the predicted graph, compared to the ground truth graph. These priorities shape the architectural choices, learning strategies, and evaluation protocols throughout the dissertation.

The parsing problem in our work is approached through two modalities: born-digital parsing and visual parsing. **Born-digital parsing** uses content in vector-based formats such as PDF files, where the underlying representation retains visual and structural information about the formulas and diagrams. This format enables more precise extraction of graphical primitives, including lines, symbols, curves, and spatial arrangements. Born-digital content often appears in modern technical documents, research papers, and digital libraries, representing a wealth of data that can be leveraged to improve recognition accuracy and computational efficiency.

In contrast, **visual parsing** targets raster images or stroke-based inputs which are typically derived from scanned documents, handwritten notes, or digital sketches. This modality poses unique challenges due to variations in image quality, resolution, and potential noise or distortions. The visual parsing approach focuses on converting these pixel-based or stroke-based representations into machine-readable formats, produced by born-digital parsers.

Obtaining accuracy and interpretability in visual parsing requires effective feature extraction, robust segmentation, and graph-based modeling techniques capable of handling diverse and ambiguous input data. Segmentation plays a critical role in enabling *interpretability*, as it establishes clear correspondence between visual input entities (e.g., strokes, line segments, CCs) and their corresponding entities in the output structure. This alignment is essential for downstream tasks such as symbol classification and relationship inference, and it supports transparency and verification in system behavior. In contrast, encoder-decoder models that treat input images holistically often lack this fine-grained correspondence, making it difficult to trace how visual elements contribute to the final output and limiting their usefulness for interpretable models and error analysis.

By addressing both born-digital and visual parsing, this thesis aims to create a unified framework capable of processing mathematical and chemical diagrams from a variety of input sources.

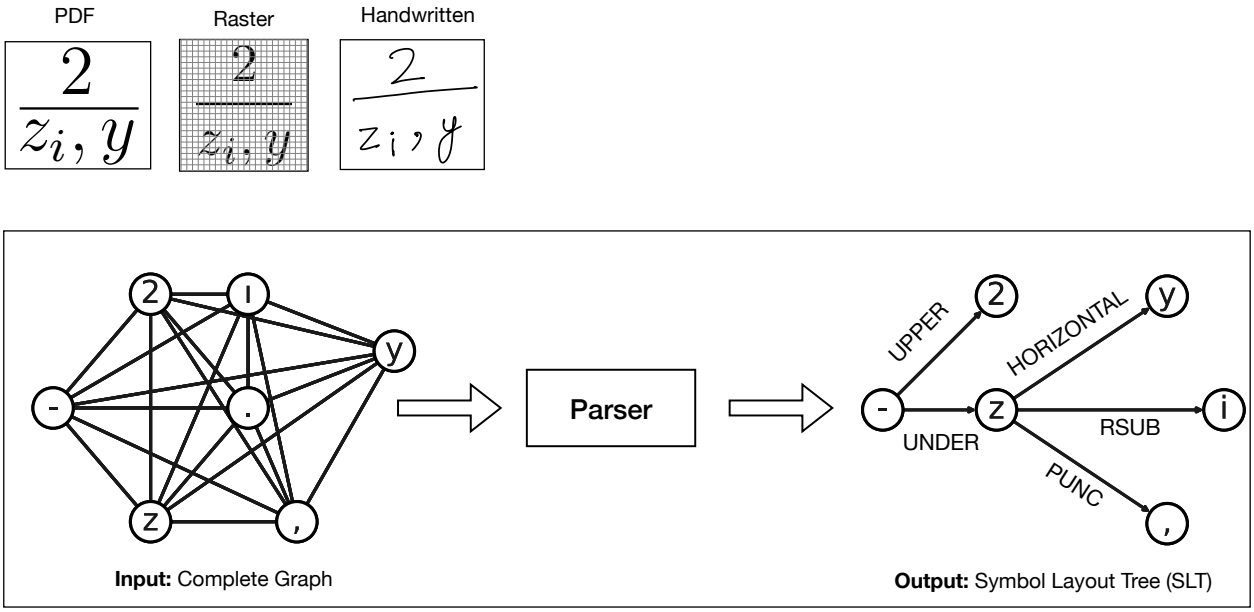


Figure 1.1: Parsing the math formula $\frac{2}{z_i,y}$ from three input sources: (left) a vector-based PDF rendering, (middle) a rasterized image version of the same formula (shown with overlaid grids to emphasize its pixel-based structure), and (right) a handwritten version represented using pen stroke data. The complete graph shows the raster image input, where each node represents a connected component (CC) of the formula.

Mathematical Formula Parsing. Mathematical formulas consist of symbols and structural relationships arranged in complex spatial configurations. In our segmentation-aware approach, parsing these formulas involves segmenting the image into symbols, classifying these symbols, and identifying relationships among them to reconstruct the underlying structure accurately as shown in Figure 1.1. This process becomes more challenging with visual inputs due to variations in font, size, layout, and noise from scanning artifacts. In contrast, born-digital parsing of mathematical formulas benefits from access to vector graphics, which often provide clean representations but still require careful handling of spatial relationships and symbol connectivity.

Chemical Diagram Parsing. Chemical diagrams depict molecular structures, including atoms, bonds, and stereochemical features. Parsing these diagrams requires accurately identifying atoms and bonds, classifying bond types, and capturing both spatial and chemical relationships, as illustrated in Figure 1.2. Spatial relationships refer to the geometric layout of diagram elements, such as the relative position, orientation, and distance between atoms and bonds, which are critical for interpreting stereochemistry and layout-dependent annotations. Chemical relationships, in contrast,

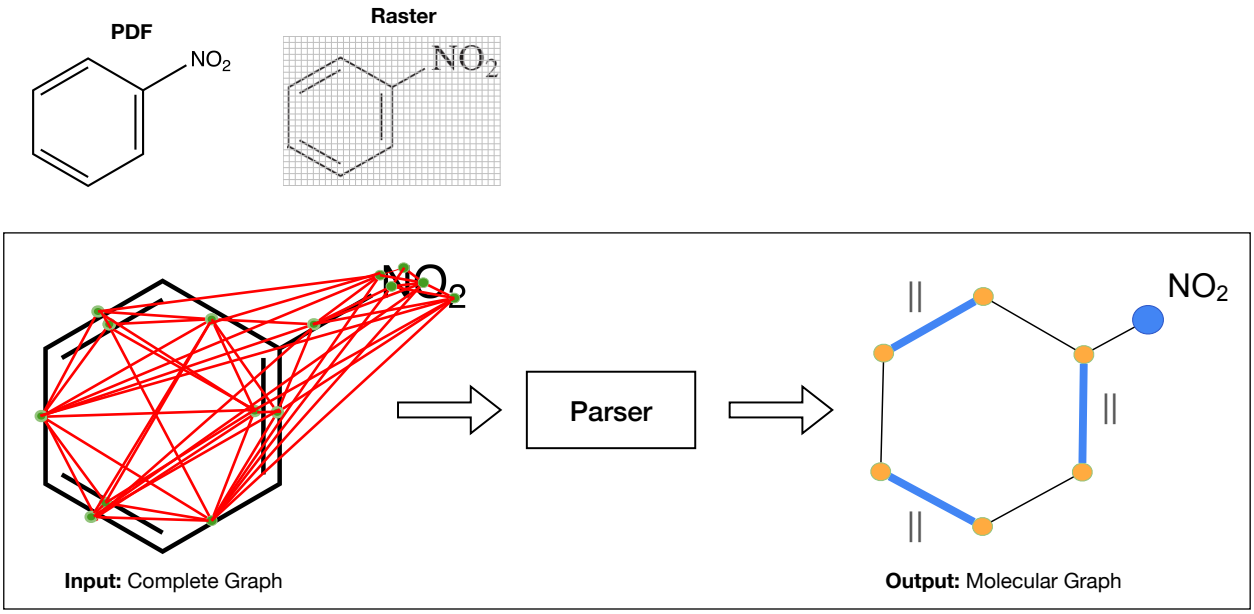


Figure 1.2: Parsing chemical diagram (Nitrobenzene) from PDF image and raster image sources. Manual grids are added to the raster image to highlight its pixel-based nature, visually distinguishing it from the vectorized PDF representation. The || symbol in output represents merged lines indicating double bonds.

describe the underlying molecular connectivity, i.e., which atoms are bonded and how they form a connected graph, independent of their specific positions on the page.

Similar to mathematical parsing, visual parsing of chemical diagrams from raster images poses challenges due to noise, varying drawing styles, and ambiguities in structural representations. On the other hand, born-digital parsing leverages vector graphics, providing precise geometric data but requires robust algorithms to interpret atom connectivity and chemical semantics.

Commonalities and Differences in Parsing Tasks. In our segmentation aware approach, both mathematical and chemical parsing tasks involve structures that require accurate symbol segmentation, classification, and relationship identification. However, there are notable differences in how these notations are represented and interpreted. Mathematical formulas emphasize hierarchical relationships among symbols, often requiring detailed spatial reasoning and layout analysis as shown by the different relationships in Figure 1.1. In contrast, chemical diagrams focus on connectivity and topological arrangements, where bonds and atom types must be identified and correctly interpreted in the context of chemical rules. We describe these representations in detail in Chapter 2.

The born-digital and visual parsing approaches further introduce distinct challenges. Born-digital content provides precise vector data that simplifies segmentation but still requires graph-based spatial reasoning to interpret relationships. Visual parsing, on the other hand, faces challenges with noise, variations, and image artifacts but is essential for document analysis in more diverse and non-standard scenarios, such as scanned documents, handwritten notes, or publications that embed raster images without accompanying vector metadata. While born-digital documents also support document analysis when vector information is available, they assume structurally consistent input. Visual parsing methods are therefore critical when such assumptions do not hold, requiring image processing and neural network models to recover accurate and interpretable representations from visual data.

1.3 Research Questions and Thesis Outline

The goal of this thesis is to improve the accuracy, and interpretability of visual parsers for mathematical formulas and chemical diagrams. To this end, we aim to answer the following key research questions, which guide the development, analysis, and evaluation of our proposed methods:

RQ1 How can primitive features be constructed and updated efficiently to support iterative refinement of segmentation and improve structural parsing?

RQ2 What types of input graph representations are most effective for parsing mathematical formulas and chemical diagrams, and do these representations differ between the two domains?

RQ3 How can graph context and class distributions be leveraged to improve contextual representation and task interaction in multi-task parsing?

RQ4 Can visual and structural noise at the primitive level, together with class balancing strategies, improve the robustness of the visual parser?

This dissertation aims to addresses the task of parsing mathematical formulas and chemical diagrams from both vector-based and raster image data. Each chapter explores specific technical methods, algorithms, and evaluation frameworks developed throughout this research.

Chapter 2 provides the necessary background on representations, input sources, graph structures, and output notations used in mathematical and chemical parsing. It details various evaluation metrics, including graph-based methods like adjacency matrices and LgEval, and string-based measures.

This chapter also reviews existing parsing models and relevant parsing techniques, setting the stage for the subsequent chapters.

In Chapter 3, we present methods for born-digital parsing from vector-based PDF symbols. This includes extracting symbols from PDFs, constructing Symbol Layout Trees (SLTs) for mathematical formulas using Edmonds' algorithm on Line-of-Sight graphs [113], and parsing molecular diagrams through Minimum Spanning Tree (MST) transformations. The chapter also discusses generating training data for visual parsers.

Chapter 4 focuses on visual parsing from raw images, extending our graph-based parsing methods to raster image data. The chapter introduces the Line-of-Sight with Graph Attention Parser (LGAP) [114] and the Line-of-Sight Chemical Graph Parser (LCGP) models [112], explaining their input structures, feature extraction techniques, and graph transformation processes. LCGP extends LGAP to chemical diagrams, introducing a contour-based primitive extraction method that directly leverages extracted contours from images to improve visual features, as well as introduces a segmentation-aware refinement process that iteratively merges primitives and updates visual features, addressing the goals of **RQ1**.

Chapter 5 addresses **RQ2** and **RQ3** by investigating the choice of input graph representations and the role of contextual interaction in multi-task parsing. We systematically compare Line-of-Sight (LOS), k-Nearest Neighbor (KNN), and complete graphs, evaluating their coverage, sparsity, and impact on parsing accuracy in both math and chemical domains. The chapter also introduces improvements in primitive extraction and unifies visual feature representations across domains using contour-based feature construction and a shared ResNeSt-50 encoder. We then propose EGATv2, a multi-hop, edge-aware graph attention model that enables joint node-edge updates and task interaction through a two-stage architecture with aggregation of class distribution and visual features together.

Chapter 6 addresses **RQ1** by improving the robustness and efficiency of visual primitive extraction from raster images, including the separation of over-segmented primitives. It also contributes to **RQ4** by focusing on robustness and generalization of the visual parser under noise and data imbalance. We simulate visual and structural degradation using synthetic noise transformations and apply these during training to increase resilience to real-world image variations. Additionally, we analyze class imbalance across tasks and propose sampling, loss weighting and loss aggregation strategies to improve performance on rare classes. All experiments in this chapter build on the EGATv2 model introduced previously, using its two-stage, two-hop configuration as the baseline.

Chapter 7 summarizes the key findings and contributions of this research, highlighting the advancements made in parsing mathematical formulas and chemical diagrams. The chapter discusses the implications of these findings for future research and applications, including potential extensions to other domains and the integration of these parsing techniques into broader document analysis systems.

Chapter 8 lists the publications that have resulted from this research, providing context for the contributions of our work.

1.4 Thesis Statement

Robust and interpretable graph-based visual parsing of mathematical and chemical diagrams can be achieved through: (1) domain-specific graph representations; (2) iterative merging of input nodes representing connected components and strokes (for math) or line-based visual primitives (for chemistry); and (3) multi-task learning incorporating randomized primitive splitting, visual noise augmentation on primitive query and context images, contextualized graph attention, and class-balanced loss functions.

1.5 Contributions

The key contributions of this research are as follows:

1. **Training Data and Evaluation Metrics for Chemical Diagram Recognition.** Developed a geometry-based ground truth generation method for parsing raster chemical diagrams by aligning visual primitives (line segments) nodes with vector primitive nodes in generated PDF visual graph using spatial and geometric heuristics (maximum overlap). This enables precise, interpretable annotations directly tied to input visual primitives. For evaluation, we adapted existing LgEval[1] metrics [84, 85] to chemical diagram recognition by generating label graph files compatible with its adjacency-based evaluation, supporting fine-grained, node- and edge-level performance assessment. This approach overcomes limitations of traditional string-based metrics (e.g., SMILES), which lack spatial alignment and error localization, especially for stereochemical structures. While designed for chemistry, the ground truth alignment

[1] `https://gitlab.com/dprl/lgeval`

method can be extended to other diagrammatic domains such as mathematical expressions, flowcharts, or circuit diagrams, wherever visual primitives can be mapped to structured representations.

2. **Parsing Mathematical Formulas and Molecular Diagrams Using Vector Information.** Developed graph-based parsers for extracting structures of mathematical formulas and molecular diagrams directly from vector-based PDF content. For mathematical formulas, Symbol Layout Trees (SLTs) were constructed by applying Edmonds' algorithm [38] on an input Line-of-Sight (LOS) graph with symbols extracted from PDF, capturing the spatial relationships among symbols and primitives without the need for OCR. Likewise, for molecular diagrams, a Minimum Spanning Tree (MST) approach based on distance was used to construct an input graph, which was subsequently transformed into a molecular structure graph through a series of graph transformations guided by chemical and spatial constraints. This approach bypasses rasterization and complex vector conversions, enabling efficient and interpretable graph construction that preserves geometric and contextual information directly from vector-based data [112, 113].

3. **Recurrent Segmentation-Aware Graph Parsing (RQ1, RQ2).** Developed segmentation-aware graph parsers for both mathematical formulas and chemical diagrams that iteratively refine symbol groupings and update visual features. This includes:

   - Enhanced visual feature extraction through the incorporation of line-of-sight (LOS) neighbors context features and spatial pyramidal pooling [114]. Also, introduced an input refinement mechanism that iteratively merges primitives into symbols and updates corresponding visual features [112].
   - Evaluated and selected domain-specific input graphs—LOS graphs for math, enabling long-range structural capture with reduced noise, and 6NN graphs for chemistry, offering a balance of local connectivity and precision. Also improved representation of punctuation relationships in math formulas to increase edge coverage in LOS graphs.

4. **Robustness: Line-Based Primitives, Primitive Noise, and Loss Functions for Class Imbalance (RQ1, RQ4).** Proposed a unified visual feature design based on primitive-centered query and context windows derived from drawn contours, enabling consistent local context modeling across domains without relying on full-image features. We also introduce an efficient and robust method for extracting visual line primitives from raster chemical diagrams, which effectively separates over-segmented components and this method can be extended to mathematical formulas, flowcharts, and other diagrammatic domains. To enhance robustness

under real-world degradations, we introduced a localized noise augmentation strategy inspired by Baird's degradation model [11], applying primitive-level visual perturbations (e.g., blur, downscaling, salt-and-pepper noise) and structural noise via random binary splits. Additionally, we addressed class imbalance in segmentation, symbol, and relationship tasks using a combination of reweighted loss functions including weighted cross-entropy, focal loss, and class-balanced loss and stratified sampling at the formula level to promote rare-class learning and task balance. To further support task-level balance, we aggregate losses across tasks using a complemented harmonic mean.

5. **Edge-Aware Graph Attention with Cross-Task Interaction (RQ3):** Introduced EGATv2, a two-stage, edge-aware graph attention network that aggregates visual features and class distributions for both nodes and edges, enabling bidirectional message passing. The architecture supports cross-task interaction, improving coordination among symbol, segmentation, and relationship predictions.

1.6 Limitations

Despite the contributions introduced in this work, several limitations remain:

1. **Primitive extraction sensitivity.** The current primitive extraction pipeline relies on hand-tuned heuristics and assumes clean segmentation. In degraded or stylized images, primitive detection and grouping can fail or require manual tuning. Furthermore, visual primitive extraction was applied only to chemical diagrams, limiting generalizability.

2. **Fixed graph structures.** The use of static input graphs (e.g., LOS or KNN) introduces tradeoffs between structural fidelity and computational efficiency. Heuristic edge pruning may not generalize well to all diagram types or layouts, especially in noisy or unconventional inputs.

3. **Symbol disambiguation challenges.** EGATv2 struggles to distinguish visually similar but semantically different symbols, such as minus versus equals signs in math, or single bonds versus strokes in characters, especially in dense or ambiguous contexts.

4. **Limited augmentation diversity.** The visual and structural noise models used for robustness rely on simple binary splits and a small set of perturbation types. These may not sufficiently model real-world degradations such as scanning noise, background clutter, or variable font rendering.

5. **Loss weighting and task balance.** Although focal loss and class-balanced loss were employed to address class imbalance, only a narrow range of parameters (e.g., $\gamma = 0.5$) was explored. The complemented harmonic mean used for multi-task loss aggregation improves balance across tasks, but further tuning or generalization may be needed to ensure stability across class distributions.

Chapter 2

Background

Parsing of mathematical formulas and chemical diagrams involves transforming raw raster or vector data into structured representations, such as Symbol Layout Trees (SLTs) for math and molecular graphs for chemistry. This chapter provides an overview of the foundational concepts, methodologies, and evaluation metrics essential for these tasks, setting the groundwork for the proposed research.

The chapter begins with *Data Representations and Structures* (2.1), which explores the various input sources, associated visual primitives, and their transformations into graph representations. This section covers input types such as born-digital PDFs, typeset images, and handwritten strokes, discussing how these are converted into graph structures that capture spatial and structural relationships. It also describes the output representations, including SLTs and molecular graphs, which serve as the final structured outputs of parsing systems.

The second major section, *Parsing Models* (2.2), reviews prior work in mathematical and chemical diagram parsing. This discussion traces the progression from early rule-based approaches to neural network-based methods, highlighting their application and limitations in both domains. The evolution of these models underscores the need for further innovations to address challenges in accuracy, speed, and interpretability.

The chapter then delves into *Techniques Relevant to Parsing Models* (2.3), discussing advanced methods that enhance graph-based parsing. Topics include multi-task learning and interaction [21], which leverage shared representations across related tasks to improve model performance. Aggregating and distributing contextual information is also examined, alongside local constraints like

nearest-neighbor pruning for focusing on relevant graph connections. The importance of incorporating edge features to capture structural relationships and advancements in graph attention methods, such as Relational Attention [33] and Graphormer-inspired encodings [148], are also explored in depth.

The final section, *Evaluation Metrics* (2.4), describes the methods used to assess the performance of parsing systems. This includes graph-based metrics like structure matching accuracy and frameworks like LgEval library[1] [84, 85], as well as string-based metrics, such as SMILES matching for chemical diagrams and LaTeX accuracy for mathematical formulas. These metrics provide a robust framework for evaluating the effectiveness of parsing models.

This chapter provides a detailed understanding of the key concepts and methodologies underlying visual parsing. It lays the foundation for the proposed research directions aimed at improving the robustness, accuracy, and interpretability of visual parsers for mathematical and chemical diagrams.

2.1 Data Representations and Structures

The recognition and parsing of mathematical formulas and chemical diagrams depend heavily on how input data is represented and structured [154]. The inputs for these tasks come from various sources such as born-digital PDFs, typeset images, and handwritten strokes, each providing unique challenges for extracting meaningful visual primitives. These primitives are then organized into graph-based representations, which capture the relationships between the individual components, such as symbols in math or atoms and bonds in chemistry. This section explores the different input types, the graph structures used to represent their relationships, and the domain-specific output formats required for the final interpretation of both mathematical and chemical structures.

2.1.1 Input Sources and Primitives

PDF (Born-digital): PDF primitives

Born-digital PDFs provide an essential source of data for parsing both mathematical formulas and chemical diagrams. Baker et al. [12] introduced the use of symbol information from PDFs directly rather than applying OCR to rendered images, which was later adopted by many subsequent math

[1] `https://gitlab.com/dprl/lgeval`

recognition systems [3, 67, 165]. Unlike raster images, PDFs store vectorized graphical and text data, which allow for precise extraction of visual and symbolic information. This precision removes the need for Optical Character Recognition (OCR), which is often error-prone, and instead allows for direct access to encoded symbols and graphical components. It is also faster since it avoids rendering and analyzing document images. We use the following PDF primitives in our work.

For mathematical formulas, we utilize the PDF primitives as characters such as alphabets, numbers, and mathematical symbols (e.g., equals signs, fraction lines, and dots) as seen in Figure 2.1 (a). These primitives are represented in the PDF data as vector graphics or embedded glyphs, with detailed properties such as position, size, font style, and typeface. The precise encoding of symbols within the PDF allows for a reliable representation of formula primitives.

For chemical diagrams, we use graphical elements such as lines, polygons, curves, and text characters that represent atoms and bonds as our PDF primitives as seen in Figure 2.1 (d). These graphical primitives provide detailed geometric encodings of molecular structures. For example, lines encode bonds between atoms (e.g., single, double, or triple bonds), while text characters encode atom labels (e.g., 'C' for carbon or 'O' for oxygen). Nodes correspond to atoms or groups and edges represent bonds. These lines and characters form the PDF primitives for the parsing task.

Overall, the advantage of using born-digital PDFs in both mathematical and chemical parsing lies in the accuracy and structure of the data. The vectorized encodings provide a high level of precision, enabling robust parsing that avoids the complexities associated with OCR or raster image processing.

Typeset Formula Images (Raster): Visual primitives

Typeset formula images, or raster images, provide a pixel-based representation of both mathematical formulas and chemical diagrams. Unlike vector-based formats like PDFs, raster images consist of pixel data, and the extraction of visual primitives from such data involves challenges related to resolution and noise. Usually encoder-decoder models [136, 160, 162] use raster images directly as inputs. However, in our work, for both domains, we extract visual primitives from the raster images to form the basis of our structured graph representation. This approach offers a higher-level abstraction by capturing distinct visual elements such as connected components or lines, rather than processing raw pixel data from the entire image

In mathematical formula parsing, we use connected components (CCs) as the primary visual primitives as shown in Figure 2.1 (c). CCs are extracted based on connected component analysis,

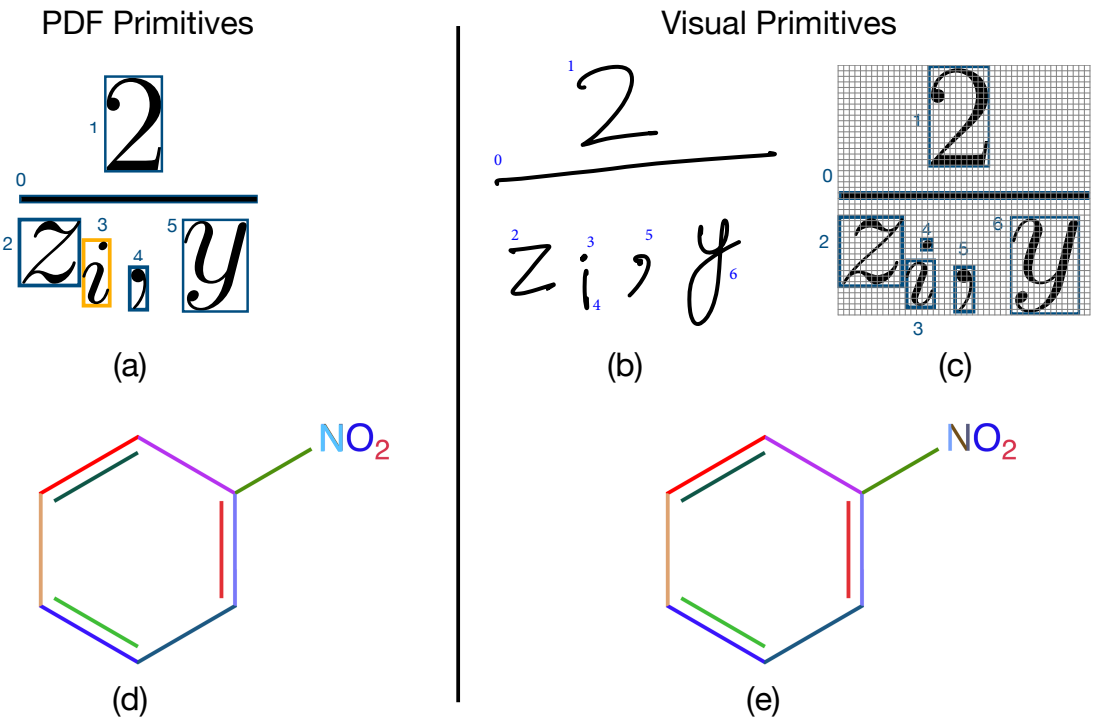


Figure 2.1: Different types of primitives in math and chemical diagrams. (a) PDF primitives for math formula $\frac{2}{z_i,y}$. Note that the symbol 'i' appears as a single PDF primitive. (b) and (c) are visual primitives for the same formula from handwritten strokes and raster image (CCs) respectively. Note that the symbol 'i' is segmented into two visual primitives (CCs/strokes) (d) and (e) correspond to the PDF and visual primitive for the chemical diagram of Nitrobenzene. Note that the character 'N' is split into 3 lines in (e) but appears as a single PDF primitive.

where touching pixels are grouped together. These components represent individual symbols or parts of symbols, such as variables, operators, or punctuation marks. In this process, CCs are rarely under-segmented but may be over-segmented, meaning that portions of a symbol might be split into multiple components (e.g. 'i' into line and dot). However, over-segmentation can be handled by our visual parser model, discussed in Chapter 4.

For chemical diagrams, CCs cannot be directly used, as the bond lines in molecular structures are often connected, resulting in large CCs that contain multiple bond lines and atom labels. This leads to under-segmentation, making it difficult for the model to distinguish between individual primitives such as bonds and atoms. To address this, we split these large CCs into smaller line primitives as seen in Figure 2.1 (e) using geometry-based segmentation approach described below.

Visual Primitives Extraction. The process begins by extracting CC contours from the raster image and converting them into polygons as shown in Figure 2.2 (b) using a simplification algorithm provided by `Shapely library`[2]. These polygons are then further split into skeletal lines by identifying pairs of adjacent parallel lines on the contour boundary. Each pair of parallel lines is replaced by its medial axis (i.e., the line between the midpoints of the parallel lines' endpoints), which represents the bond. Pixels within the CCs are then segmented by assigning them to the nearest skeletal line using a distance transform. This segmentation helps ensure that overlapping or connected bond lines are separated into individual primitives.

The stages of this visual primitive extraction process, from simplified polygons to skeletal lines and final line primitives, are illustrated in Figure 2.2. Some CC shapes such as curved lines and closed curves are unaltered by the process. The '2' and 'O' are unsegmented in Figure 2.2 (d) because after identifying all skeletal lines for CCs in a molecule, to avoid segmenting small CCs, we test whether the average skeletal line length in a CC is less than the average for all skeletal lines. If this average length is smaller than the global average, we do not segment the CC. We also remove skeletal lines within CCs that are smaller than the global average skeletal line length, which avoids over-segmenting lines at dense intersections (e.g., at the connection point between two single bonds and a double bond). We split a long line in a triple or double bond by projecting the floating line onto it, and then testing if the overlap ratio r for the longer line is in the interval of one third to one half, with a margin of 10% ($\frac{1}{3} - \frac{1}{10} \leq r \leq \frac{1}{2} + \frac{1}{10}$).

Overall, while raster images present additional challenges due to under-segmentation in chemical diagrams, the extraction of visual primitives allows for an accurate representation of molecular structures that can be processed and evaluated further.

Handwritten Strokes

In formula recognition, handwritten strokes present unique challenges compared to born-digital or typeset formats. Handwritten input introduces variability in symbol shapes, sizes, and spatial relationships due to individual handwriting styles. Handwritten strokes are treated as the fundamental visual primitives in this domain. Each stroke corresponds to a continuous pen movement from a starting point to an endpoint, captured as a sequence of 2D coordinates. These strokes are extracted directly from the input, and the model must determine which strokes belong together to form meaningful symbols, such as numbers, variables, operators, and punctuation marks (See Figure 2.1 (c)).

[2] `https://shapely.readthedocs.io/en/stable/`

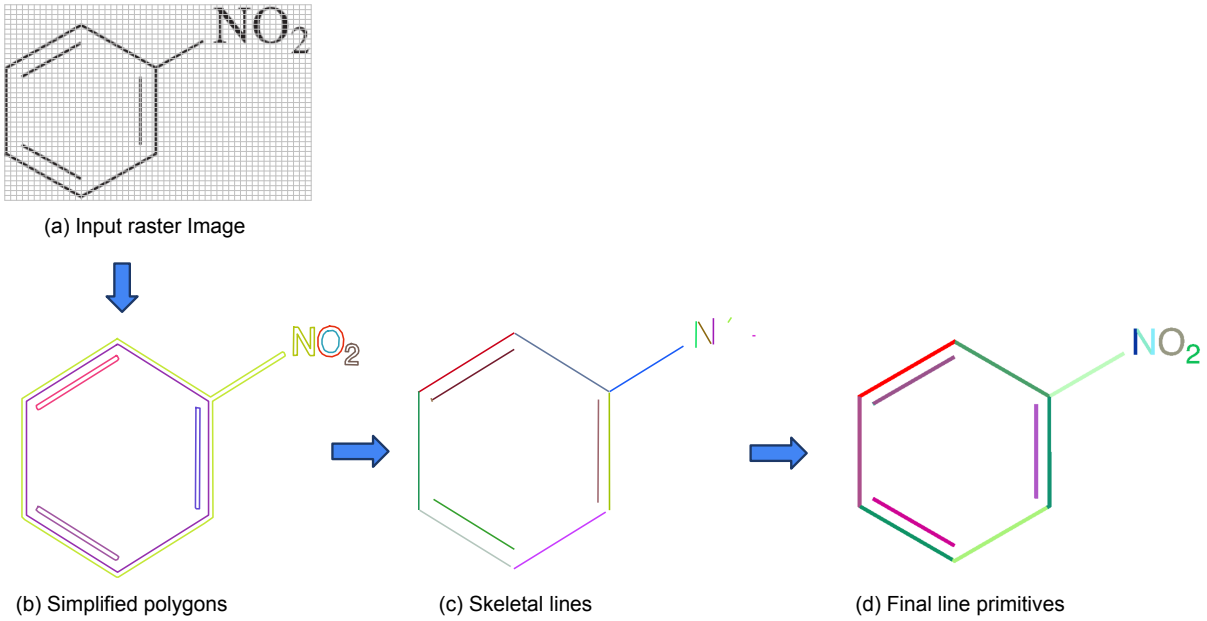


Figure 2.2: Stages of visual primitives extraction from a raster image of a chemical diagram. (a) Input raster image, (b) simplified polygons derived from CC contours, (c) skeletal lines extracted as medial axes of parallel line pairs, and (d) final line primitives after segmentation and refinement. Note that '2' and 'O' are unsegmented in (d) because their corresponding skeletal lines are smaller than global average (c).

The recognition of handwritten mathematical formulas has been extensively explored, with early CROHME competitions providing benchmarks and advancements to manage input variability and complexity [80, 82, 84, 85]. Similarly, handwritten chemical diagram recognition has been studied in various works [102, 124, 166]. The ICDAR 2024 Competition on Handwritten Chemical Structure Recognition [23] introduced a novel task focused on recognizing handwritten chemical structures, a challenging problem due to the lack of standardized data and the high variability in handwritten styles. The competition utilized the EDU-CHEMC dataset, containing over 60,000 handwritten images collected in educational settings, annotated with SSML strings. Unlike SMILES, SSML provides a representation closely tied to image structure, incorporating features such as bond angles and visual layout.

The major challenge in handwritten formula parsing is managing the variability in how symbols are written. For example, the same symbol can be written with different numbers of strokes (e.g., the digit '8' can be drawn in one continuous stroke or two separate loops). Similarly, symbols like 'x' or '+' can have variations in their structure due to different handwriting styles. As a result, stroke

segmentation and symbol recognition require robust methods that can adapt to these variations.

In practice, over-segmentation may occur, where a symbol is split into multiple strokes (e.g., a handwritten 'x' drawn with two separate strokes). The model must learn to merge these over-segmented strokes into a single symbol during the parsing process. Conversely, under-segmentation is rarely an issue, as individual strokes are generally isolated and separate.

Our work focuses on handwritten math formulas, while handwritten chemical formulas is beyond the scope of this work.

2.1.2 Graph Representations

In the context of formula recognition, graph representations are essential for capturing the relationships between symbols or primitives in mathematical and chemical diagrams. These representations offer a structured way to model the spatial and logical relationships between elements, whether they are PDF symbols in born-digital formulas, strokes in handwritten formulas, connected components in math formula images, or lines and characters in chemical diagrams. Eto et al. [39] were the first to take this approach for representing math formulas, creating a graph with symbol nodes containing alternative labels, and candidate spatial relationships on edges with associated costs. Line-of-sight (LOS) graphs select edges where strokes in a handwritten formula or connected components in an image are mutually visible [52, 71]. Systems such as Hu et al. [50], LPGA [71], and QD-GGA [72] use this LOS input graph constraint, and select the final interpretation as a directed Maximum Spanning Tree (MST).

MolGrapher [79], ChemGrapher [94], Tang et al. [124], MolScribe [100], image to graph transformer [149], use graph representations for parsing chemical diagrams. In our work, we treat the extracted born-digital (PDF symbols) and visual primitives (strokes, connected components, or geometric shapes) as nodes, while the spatial or structural relationships between them as edges. This allows for a flexible and interpretable way to represent the formula or diagram's overall structure. The advantage of graph-based representations is their ability to encode both the local relationships between elements (e.g., how individual primitives combine to form symbols) and the global structure (e.g., the spatial arrangement of symbols or primitives within a formula or molecule).

We divide the graph representations into two main categories: `Visual Syntax Graph`, which capture the immediate syntactic relationships among visual primitives and symbols, and `Representation Graph`, which represent the formal syntactic interpretation of the parsed formula or molecular dia-

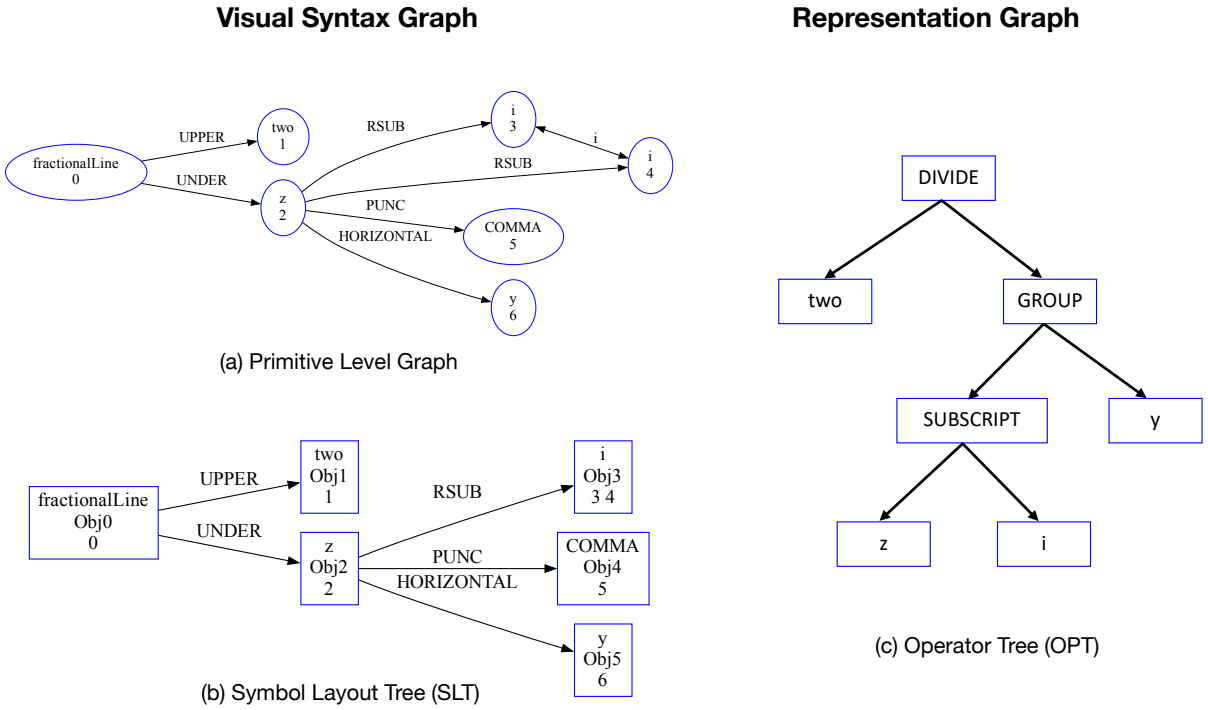


Figure 2.3: Illustration of different types of graph representations for the mathematical formula $\frac{2}{z_i, y}$. (a) The Primitive Level Graph represents the low-level primitives (e.g., connected components or strokes) as nodes, labeled with their unique identifiers (shown in blue), and spatial relationships between these primitives are depicted as directed edges. (b) The Symbol Layout Tree (SLT) abstracts symbols as nodes formed by grouping primitives (e.g., the 'i' comprises strokes/CCs {3,4}), with spatial relationships between symbols shown as directed edges. The numeric identifiers of strokes/CCs (a) and their corresponding grouped symbols (b) are consistently shown in blue. {3,4} is shown in one node in (b), but two nodes in (a). (c) The Operator Tree (OPT) represents the semantic structure, with operators (e.g., 'DIVIDE' for the fraction) and operands (e.g., 'two', 'z', 'i', and 'y') as nodes, and directed edges indicating hierarchical relationships. The 'GROUP' node in the OPT is treated as a grouping operator, connecting its operands 'z_i' and 'y'.

gram.

In the following sections, we detail these graph types for both math and chemistry. This includes explanations of the primitive-level and symbol-level graphs within the Visual Syntax Graph category and descriptions of the OPT for math and the molecular graph for chemistry within the Representation Graph category. Additionally, we discuss label graph (Lg) files [84,85], used to store these representations in a structured format, facilitating evaluation and downstream processing.

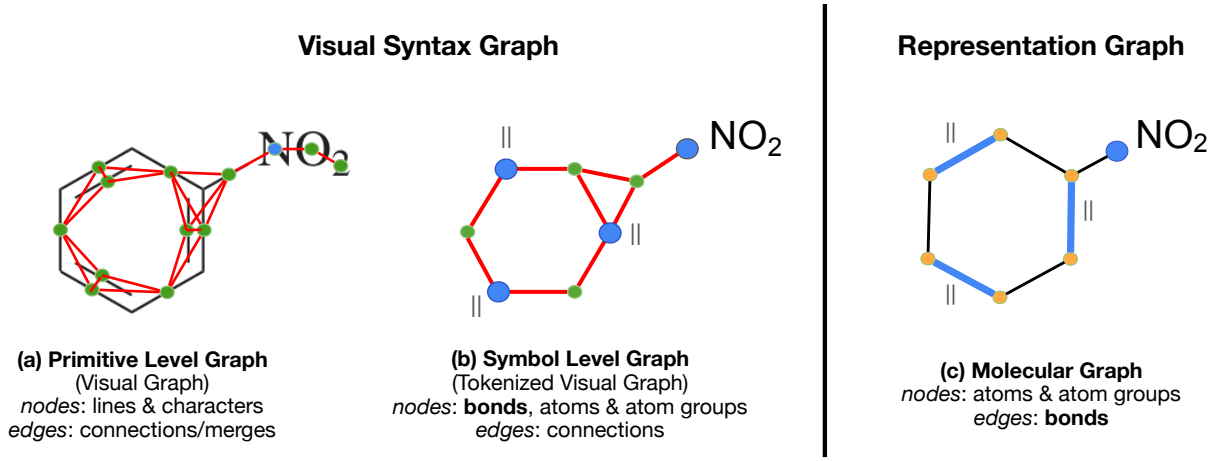


Figure 2.4: Different types of graph representations (visual syntax graph and representation graph) for the chemical diagram Nitrobenzene. (a) *Visual Graph* showing lines and characters as nodes (in green), and connections/merges as edges (in red). (b) *Tokenized Visual Graph* with merged nodes (bonds and named groups). (c) Molecular Graph. Blue nodes show the primitives of N merged into a character (a) and double bonds and atom/group names in (b,c). In (c) orange nodes are 'hidden' carbon atoms, and single/double bonds are converted from nodes to edges

Visual Syntax Graph

They capture the immediate syntactic relationships among visual primitives and symbols. Visual syntax graphs encompass the primitive-level graph and the symbol-level graph. The primitive-level graph operates at the level of individual strokes, connected components, or lines, representing basic visual primitives. In contrast, the symbol-level graph organizes these primitives into meaningful symbols, capturing their structural relationships within a mathematical formula or chemical diagram. For mathematical formulas, this results in a Symbol Layout Tree (SLT), while in chemical diagrams, it produces an intermediate tokenized visual graph.

1. Primitive Level Graph (CCs/Strokes/Lines) At the primitive level, we represent graphs as the lowest level of structure for both mathematical and chemical formula parsing. We treat the basic visual primitives—whether they are connected components (CCs) in typeset images, strokes in handwritten formulas, or characters and lines in chemical diagrams—as nodes in a graph. We use edges to capture the relationships between these primitives, such as spatial adjacency or merging into larger symbols.

In our mathematical formula parser, we build the primitive-level graph using connected com-

ponents (CCs) from typeset images [71, 72] or strokes from handwritten input [85, 141]. We encode the relationships between these elements as edges in the graph. Each node represents an individual CC or stroke, which can represent one of N symbol classes, such as digits, operators, or punctuation marks. We capture two types of relationships with edges: merge edges (binary), which are used to combine multiple strokes or CCs into a single symbol, and spatial relationship edges (E classes), which represent the spatial relationships between nodes. For example, if the letter 'x' is written with two separate strokes, we connect these strokes with a merge edge to form the final symbol. We capture spatial relationships such as subscript, superscript, horizontal alignment, inside, above, and below.

The number and arrangement of strokes or CCs may differ for a single formula. For the example in Figure 2.3, the primitive label graphs for the image and handwritten version are identical so that CCs and strokes have an exact correspondence. This is not always the case, for example when an 'x' drawn with two handwritten strokes appears as a single CC in a typeset image.

In chemical diagram parsing, we treat visual primitives as characters, parts of characters, and lines. We extract these primitives from the image and construct a primitive-level graph, where each node represents an extracted character, a part of a character, or a line. Each node can represent one of several categories, denoted as N classes, including atoms (e.g., 'N', 'H', and 'C'), charges ('+', '-'), and bond types ('Single', 'Solid Wedge', 'Brackets'). We use edges to represent binary relationships between nodes. There are two types of edges: connection edges indicate whether two nodes, such as an atom and a bond, are directly connected in the diagram, while merge edges are used to combine parts of symbols that may have been over-segmented during the extraction process. For example, we create a connection edge between an atom node (e.g., 'C') and a bond node (e.g., a single bond line) if they are spatially adjacent. Similarly, if a bond line or character has been split into two or more segments, we add merge edges to combine these segments into a single node (See Figure 2.4).

2. Symbol Level Graph At the symbol level, graph representations capture the relationships between higher-level, semantically meaningful components. These components, referred to as tokens or symbols, represent groupings of characters or visual elements that collectively carry meaning. Tokenization is the process of identifying these groupings—whether they are mathematical operators [150], chemical elements [79], or even language keywords—by segmenting and classifying sequences of connected characters or primitives. For instance, in math, the string 'cos' forms a token representing the trigonometric function cosine, while in chemistry, 'CH_3' represents a supteratom (group of atoms treated as a single entity, such as functional groups) called methyl group. Similarly, in other domains

like programming, tokens such as 'def' in Python signal a function definition. Thus, tokenization aims to transform sequences of connected components or primitives into symbols with specific meaning for further processing. These tokens form the nodes of the graph, and the edges capture the spatial or structural relationships between these elements. These symbol-level graphs allow for a more abstract representation of the formula or diagram, moving beyond the raw primitives toward meaningful semantic groupings.

In mathematical formula parsing, we represent the symbol-level graph as a Symbol Layout Tree (SLT). In this tree, we treat nodes as recognized symbols or tokens, such as numbers, variables, operators, and punctuation marks. For example, 'x', '2', '+', or '=' each form individual nodes. Additionally, we handle tokenized symbols like 'cos' as a single node to capture the semantic meaning of the trigonometric function. We classify each node into one of N symbol classes, and we use edges in the SLT to represent spatial relationships between symbols, including subscript, superscript, horizontal adjacency, above, below, or inside. These spatial relationships are used for capturing the two-dimensional structure of mathematical expressions. For instance, in the expression x^2, we encode the superscript relationship between 'x' and '2' as an edge in the SLT, labeled as 'Superscript'. Similarly, we capture subscript relationships (e.g., 'a_i'), horizontal adjacency (e.g., 'a + b'), punctuation relationships (e.g., commas or parentheses), above and below (e.g., numerators and denominators in fractions), and inside (e.g., numbers inside square roots) within the SLT. By forming a graph that represents the hierarchical structure of the entire formula, SLTs provide a consistent description for formulas in both handwritten strokes and typeset images.

One important characteristic of math graphs is that SLTs form directed trees, meaning each node maintains directed relationships with others, and cycles are not allowed. This tree structure ensures an unambiguous representation of nested expressions, where each sub-expression fits into the hierarchy without loops.

In chemical diagram parsing, we construct the symbol-level graph as an intermediate tokenized visual graph after extracting characters and bond lines from the image. In this graph, we represent nodes as atoms, atom groups, or bond types. Unlike typical chemical graphs [79, 100], where atoms are represented as nodes and bonds are represented as edges, we treat both the visual elements, atoms and bonds as nodes, with edges denoting the connectivity between them. For example, nodes may represent chemical elements such as 'C', 'O', or 'N', atom groups like 'NO_2' or 'CH_3', and bond types such as 'Single', 'Double', or 'Solid Wedge'. We use edges to represent binary, bidirectional connections between nodes. For instance, an edge is created to connect a carbon atom node ('C') to a 'Single' bond node if these elements are linked in the molecular structure.

Unlike in math, where spatial relationships such as superscript or subscript are crucial, chemical structure (molecular graph) can be obtained with simple graph rewrites using these direct binary connections between atoms and bonds. Also, the tokenized visual graphs for chemical diagrams can contain cycles (i.e., they are not trees). This reflects the cyclical nature of certain molecular structures, such as aromatic rings, where multiple atoms and bonds form loops. Additionally, the edges between nodes in chemical graphs are bidirectional, as the connections between atoms and bonds are not inherently directional.

In summary, while both symbol-level graphs for math and chemistry represent higher-level structures and relationships, their properties differ. SLTs for math are directed trees with hierarchical relationships, ensuring acyclic structures. In contrast, chemical visual graphs are bidirectional, with possible cycles, reflecting the nature of molecular structures.

Representation Graph

They represent the formal syntactic interpretation of the parsed formula or molecular diagram, with nodes and edges that represent elements of the expression's logical structure. For math, the representation graph takes the form of an Operator Tree (OPT) [152, 163], capturing the hierarchical organization of operators and operands within a formula. For chemical diagrams, the representation graph is a molecular graph [77, 79, 100], which encodes the relationships between atoms and bonds, representing the final structure of a molecule. These representation graphs are essential for applications that require a formal, machine-readable representation of the formula or chemical structure.

Operator Tree (OPT). In mathematical formula parsing, the syntactic graph takes the form of an Operator Tree (OPT), which represents the computational structure of the formula. Nodes in the OPT correspond to operators, variables, or numbers, and edges represent the hierarchical relationships between these components. For example, as seen in Figure 2.3 (c), for the expression $\frac{2}{z_i, y}$ in the given OPT, the division operator `DIVIDE` is positioned at the root, indicating it as the highest-priority operation. It connects to the operands `two` and `COMMA` through directed edges. The `COMMA` node is further expanded into its substructure, where `SUBSCRIPT` is represented as a child node, connecting to the operands `z` and `i`. This hierarchical structure ensures that the syntactic relationships and precedence of operations are correctly encoded, reflecting the mathematical structure of the expression. Similar to chemical molecular graphs, OPTs provide a higher-level syntactic structure compared to Symbol Layout Trees (SLTs). While SLTs capture the spatial relationships between symbols (e.g., subscript, superscript), OPTs represent the order of operations and their

precedence, focusing on the syntactic meaning of the formula.

Molecular Graph. The molecular graph serves as the final output for chemical parsing, representing the chemical structure at a high level. It is constructed by mapping the visual elements, such as atoms and bonds, into a graph format. At the molecular graph level, nodes correspond to atoms or atom groups. Each atom node is labeled according to the element it represents (e.g., 'C' for carbon, 'O' for oxygen, 'N' for nitrogen), while superatom nodes carry the labels of the respective functional groups, such as 'NO_2' or 'CH_3'.

The edges in the molecular graph represent the bond types between atoms or atom groups. Each edge is classified into one of several categories, including Single', Double', Triple', Wavy', Solid Wedge', and Hashed Wedge' bonds. These bond types not only describe the chemical connections but also convey geometric information, such as stereochemical arrangements (e.g., solid and hashed wedges indicating bonds going into or out of the plane of the page). For instance, a single bond between a carbon (C) and an oxygen (O) atom is represented by an edge labeled 'Single' connecting the corresponding nodes. This molecular graph can be used to generate chemical representations such as CDXML [88] or SMILES [135] strings, which encode the molecule's structure in a compact, searchable format.

Analogy Between Math and Chemistry. There is a clear analogy between syntactic graphs in math and chemistry. In math, SLTs capture the visual structure of formulas and can be converted into OPTs to represent their syntactic meaning. Similarly, tokenized visual graphs in chemical diagrams represent the visual layout of atoms and bonds and are transformed into molecular graphs to capture molecular syntax. This transformation from visual to syntactic representation ensures that abstract entities and hidden elements are identified, such as hidden carbons in chemical diagrams or implicit multiplications in mathematical expressions. Both transformations facilitate downstream tasks by providing structured, meaningful representations of the original input.

The construction of molecular graphs involves additional semantic analysis steps, such as identifying hidden elements in the molecular structure. For example, hidden carbon atoms are identified at line intersections, and named functional groups are replaced by their corresponding subgraphs. In this process, dictionaries are used to map names to molecular subgraphs. As shown in Figure 2.4, a functional group such as 'NO_2' is replaced by a subgraph comprising one nitrogen and two oxygen atoms, connected to a hidden carbon at the intersection.

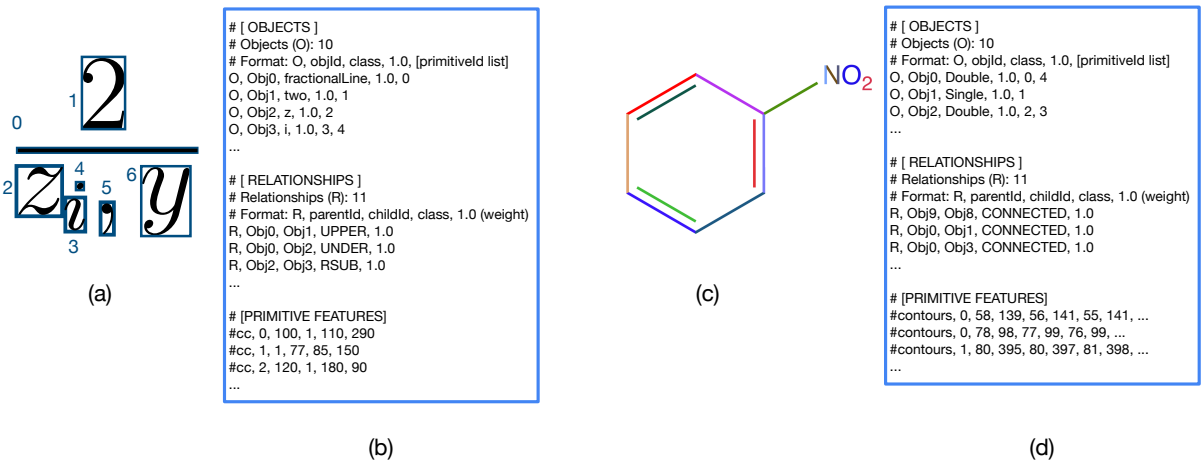


Figure 2.5: Label Graph File examples for math and chemical diagrams. (b) A Lg file for (a) input formula image ($\frac{2}{z_i, y}$). (d) A Lg file for (c) input chemical diagram image (Nitrobenzene).

Label Graph File (Lg file)

Label graph (Lg) files [84,85] are used to store input or output graph representations used during the parsing process, capturing the structure of both mathematical and chemical diagrams. These files define relationships between visual primitives—such as connected components (CCs), strokes, or line segments—and higher-level entities like symbols, atoms, or bonds. The lg file format is general-purpose, applicable to various types of graphs, regardless of whether they represent mathematical formulas, chemical structures, or other visual structures.

In an Lg file, the visual elements, referred to as primitives, are assigned unique numeric identifiers. These primitives may include strokes, CCs, characters, or line segments, depending on whether the task is mathematical formula parsing or chemical diagram parsing. Higher-level entities, known as objects (O), are constructed by grouping these primitives together. For example, in a mathematical expression, a symbol like 'x' may consist of two strokes, while in a chemical diagram, an atom such as 'N' may be composed of multiple line segments.

Each Lg file contains entries for both objects (O) and relationships (R) as shown in Figure 2.5, along with the corresponding primitive coordinate data (e.g., bounding boxes for CCs, contour points for visual primitives). The objects in the file represent groups of primitives, such as atom groups, bond segments, or mathematical symbols (e.g., the equals sign formed by two minus signs). These entries specify which primitives make up each object, whether they are individual lines of a bond in chemistry, characters forming an atom label, or characters forming a mathematical symbol.

The relationship entries in the Lg file capture the connections between these objects. In the case of math formulas, these relationships include spatial relationships such as horizontal, superscript, subscript, inside. For chemical diagrams, the connections can represent binary relationships in a tokenized visual graph (e.g., atom-bond connections) or bond types in a molecular graph (e.g., single, double, triple, solid wedge bonds).

The relationship entries in the Lg file define the connections between objects. For mathematical formulas, these relationships include spatial arrangements such as horizontal, superscript, subscript, and inside. In the case of chemical diagrams, the connections can represent binary relationships in the tokenized visual graph (e.g., atom-to-bond connections) or bond types in the molecular graph (e.g., single, double, triple, and solid wedge bonds).

Additionally, relationships like `MERGE` are implicitly defined between all primitive pairs within a single object. For example, in Figure 3.6, `MERGE` edges are present between primitives 10, 11, and 12, which together form the character 'N' (represented as `Obj10`). The connection between this character and the adjacent single bond (represented as `Obj9`) is captured by `CONNECTED` edges between primitives 9 and 10, 9 and 11, and 9 and 12. Similarly, all primitives within an object share the same label. For instance, primitives 10, 11, and 12 are all labeled as 'N' for `Obj10`.

2.1.3 Output Representations

The final step in parsing mathematical and chemical diagrams involves converting the recognized structures into domain-specific output representations that are suitable for downstream applications. These output formats are designed to convey the semantic meaning of the parsed diagrams and are widely used in various scientific and technical contexts. Depending on the domain—whether mathematics or chemistry—different output representations are employed.

In mathematical formula recognition, SLTs have popular encodings such as LaTeX and Presentation MathML (See Figure 2.6) that include additional formatting information such as fonts, font styles (e.g., italic), and spacing.

LaTeX is a widely-used typesetting system, especially for academic and technical documents. It allows precise representation of mathematical formulas in a textual format that can be rendered as high-quality typeset equations. The formulas are encoded as strings of LaTeX commands, which specify the arrangement of symbols and their spatial relationships (e.g., superscript, subscript, fraction). Parsing results are often converted into LaTeX for integration into academic papers or

Figure 2.6: Output representations for math formula and chemical diagrams. (a) Presentation MathML and (b) LaTeX representations translated from SLT for $\frac{2}{z_i,y}$. (c) CDXML output excerpt and (d) SMILES representation for Nitrobenzene.

web-based platforms.

Mathematical Markup Language (MathML) is an XML-based standard designed to represent both the presentation and content of mathematical notation. Its goal is to seamlessly integrate mathematical formulas into web pages and documents, making them accessible for both visual rendering and semantic interpretation.

MathML consists of two components: Presentation MathML and Content MathML. Presentation MathML encodes how an equation should visually appear, focusing on layout and formatting, such as superscripts, fractions, and spacing. Content MathML, on the other hand, encodes the meaning of the equation, allowing mathematical software to interpret its semantics independently of visual rendering [75].

In chemical diagram parsing, the most common output formats are SMILES and CDXML (See Figure 2.6), which are described below.

CDXML (ChemDraw XML). The chemical graph is represented in a ChemDraw[3] CDXML file [88], capturing both visual and chemical structure in molecular diagrams. CDXML is a file format representing molecules and reactions along with related text on a canvas or series of pages.

[3] `https://revvitysignals.com/products/research/chemdraw`

For molecular data, both chemical structure and the appearance of molecules on a 2D canvas are encoded in CDXML files. The format was created for the ChemDraw chemical diagram editor. In CDXML tags define molecules, nodes (e.g., atoms, named groups), and bond connections in the diagram, along with annotations for node positions and appearance. We encode the locations of nodes on their associated page, so that the appearance and location of recognized molecules match the original document. Positions are also helpful with accurate conversion to other chemical formats (e.g., SMILES), and to capture spatial information in the chemical structure (e.g., for wedge bonds).

SMILES (Simplified Molecular-Input Line-Entry System) [135] is a compact, text-based notation widely used in cheminformatics for storing, searching, and analyzing molecular data. It represents molecular structures as sequences of atoms (denoted by letters) and bonds (denoted by symbols such as '-', '=', and '#'). SMILES strings provide a concise 1-D representation of compounds that is both human-readable for domain experts and compatible with commonly used cheminformatics tools.

Each of these domain-specific representations serves a distinct purpose, ensuring that the parsed outputs can be accurately interpreted, visualized, or processed by relevant tools in mathematics and chemistry. While LaTeX and MathML focus on the presentation and content of mathematical formulas, SMILES and CDXML are designed to encode the structural and visual properties of molecular diagrams.

2.2 Parsing Models

2.2.1 Math formula parsing

Mathematical formula recognition has evolved significantly over the past few decades, transitioning from syntactic, rule-based methods to deep neural network-based techniques. Chan et al. [22], Zanibbi et al. [154], Zhelezniakov et al. [169], and Truong et al. [126] have surveyed a wide range of math recognition systems over the past few decades. Early efforts focused on leveraging grammars and syntactic rules to capture the complex spatial and structural relationships inherent in mathematical notation. However, modern approaches have shifted toward using data-driven models, including graph neural network models, encoder-decoder architectures, and transformer-based models, each offering unique advantages and addressing different aspects of the formula recognition task. In this section, we discuss the contributions and limitations of these models, and the relevance to our work.

Grammar-Based (Syntactic) Models

The recognition of mathematical expressions dates back to 1967, when Anderson introduced a syntax-directed approach based on replacement rules for recognizing two-dimensional handwritten mathematical expressions, including arithmetic formulas and matrix descriptions [6]. This approach utilized a 2D top-down parsing algorithm and an attribute grammar. Following this early work, many systems adopted syntactic or grammar-based approaches to recognize the complex structural relationships inherent in mathematical notation. Baker et al. [12] used a syntactic pattern recognition approach to recognize formulas from PDF documents using an expression grammar. The grammar requirement and need for manually segmenting mathematical expressions from text make it less robust than INFTY. However, the system was faster by avoiding rendering and analyzing document images, and improved accuracy using PDF character information. In later work, Sorge et al. [116] were able to reconstruct fonts not embedded in a PDF document, by mapping unicode values to standard character codes where possible. They then use CC analysis to identify characters with identical style and spacing from a grouping provided by `pdf2htmlEX`, allowing exact bounding boxes to be obtained.

Following on Baker et al.'s approach to PDF character extraction [12], Zhang et al. [165] use a dual extraction method based on a PDF parser and an OCR engine to supplement PDF symbol extraction, recursively dividing and reconstructing the formula based on symbols found on the main baseline for formula structure analysis. Later, Suzuki et al. improved the recognition rate in INFTYReader [119], by also utilizing extracted PDF character information from PDFMiner [121]. Some PDF characters are composed of multiple glyphs, such as large braces or square roots (commonly drawn with a radical symbol connected to a horizontal line). These 'compound' characters were identified by Baker et al. [12] using overlapping bounding boxes in modern PDF documents containing Type 1 fonts.

Context-free grammars (CFGs) became a popular choice, with several systems extending this idea through Stochastic CFGs to allow probabilistic parsing, as demonstrated by Alvaro et al. [4]. These grammars were often used alongside parsing algorithms to produce structured output such as parse trees or LaTeX strings [4, 12, 89, 125].

Grammar-based methods were particularly suited to mathematical recognition due to the domain's inherent structure, fixed symbol set, and recursive nature, all of which lend themselves to well-defined syntax rules [15]. For example, Baker et al. [12] applied a coordinate grammatical approach to recognize formulas from PDF documents, using syntactic pattern recognition with an expression

grammar.

While syntactic methods offer precise parsing capabilities, they face several limitations. Mathematical notation often varies widely, making it difficult to design universal grammars that can accommodate all symbol sets and structural variations [165]. The need to redefine formula symbols and structures for different cases further complicates the creation of robust grammars [152]. Moreover, syntactic approaches tend to be computationally expensive, as searching through rules to identify valid parses can be time-consuming, especially for complex expressions. They also require careful handling of failures when inputs do not conform to predefined grammar rules, limiting their robustness and scalability in diverse real-world scenarios.

Relation to our work. While grammar-based approaches have played a foundational role in mathematical expression recognition, our work diverges from these methods by employing data-driven techniques for parsing. Specifically, the Line-of-Sight Graph Attention Parser (LGAP) avoids predefined grammar rules and instead leverages a CNN-based network with attention mechanisms to directly learn from data.

Although LGAP utilizes PDF information for representing visual primitives and their spatial relationships, the parsing process is entirely data-driven. The LOS graph captures potential relationships between primitives, and the CNN with attention mechanisms classifies nodes and edges to refine the graph into a structured Symbol Layout Tree (SLT). This approach bypasses the rigidity and computational challenges of grammar-based parsing, offering greater flexibility and scalability for diverse inputs. Unlike grammar-based methods, which rely on predefined rules, LGAP learns the structural and spatial dependencies directly from training data, making it more adaptable to real-world variations in mathematical formulas.

Deep Neural Network Models

Deep neural network models have significantly advanced the field of mathematical expression recognition by leveraging powerful data-driven approaches for capturing complex patterns and relationships in input data. Unlike syntactic approaches, which rely on pre-defined grammar rules, deep learning models learn directly from data, offering flexibility and scalability across diverse input types, including handwritten, typeset, and born-digital formats. These approaches are generally categorized into encoder-decoder models and graph-based models, including graph neural network (GNN) models. Encoder-decoder models, including transformer-based variations, translate input images or sequences into structured representations, often leveraging self-attention mechanisms for

improved contextual understanding. On the other hand, graph-based models focus on capturing spatial and relational dependencies among symbols using graph structures.

Encoder-Decoder models are deep neural network architectures in which an encoder produces a feature embedding for input data in a lower-dimensional space, while the decoder converts the embedded input representation back into the original input, or into another representation. Generally, a Convolution Neural Network (CNN) is used for encoding pixel-level image features, and Recurrent Neural Networks (RNNs) are used as decoders to produce a string representation of a formula (e.g., in LaTeX) with some form of attention mechanism. Encoder-decoder models are used in other sequence prediction problems such as image caption generation [144], speech recognition, and scene text recognition [151], and have produced state-of-the-art results for these problems as well as math formula recognition [123,136,160,161]. The end-to-end Track, Attend and Parse (TAP) system [160] is a popular example. TAP obtained state-of-the-art results for many years, mostly because of the intelligent use of ensemble models that combine online and offline features, as well as different types of attention mechanisms.

Wu at al. [136] aim to improve generalization and interpretability in sequence-based encoder-decoder models by utilizing the hierarchical tree structure of mathematical expressions through a tree-based decoder with attention, and improving the generalization by removing the need for the spatial relationships to be in strict order. A few other variations include a counting aware network [62] using symbol counting as a global symbol-level position information to improve attention, and a stroke constrained network [133] which use strokes as input primitives for their encoder-decoder, to improve the alignment between strokes and symbols.

A recent transformer-based model by Zhu et al. [170], focused on capturing and using more detailed contextual information to improve accuracy. Their framework, called Implicit Character-Aided Learning (ICAL), introduces two key modules: one that predicts and models implicit characters (hidden symbols in mathematical expressions, including "^", "_", "{", "}") and another fusion model that merges this information with the transformer decoder's output for more accurate predictions. By enhancing how global structural information is processed, their method improves recognition performance over previous state-of-the-art techniques on the CROHME 2014/2016/2019 datasets [73, 82, 83].

Anitei et al. [7] introduce a method leveraging a Convolutional Neural Network (CNN) for initial feature extraction and a transformer encoder for modeling global context. The model, trained with Connectionist Temporal Classification (CTC) loss, operates as an encoder-only architecture, enhancing both speed and performance over traditional encoder-decoder systems. The CNN ex-

tracts local patterns while the transformer encoder, captures structural and spatial relationships in mathematical expressions through multi-head self-attention. The final CTC-based decoding step generates LATEX sequences by collapsing repeated symbols and removing blanks, They outperform the previous state-of-the-art techniques on typeset formula datasets, including IBEM [8] and Im2Latex-100k [29].

A limitation of encoder-decoder models is the interpretibility of their results. Error diagnosis in these models is challenging due to the lack of a direct correspondence between the input (image regions or strokes) and the symbols and relationships produced in an output LaTeX string or SLT: the exact correspondence between input primitives (pixels, CCs, or strokes) and the output is unknown. Hence, there is no way to identify errors at the input primitive level.

Relation to our work. Encoder-decoder models have inspired aspects of our approach to parsing mathematical formulas. Specifically, their ability to model complex, sequential dependencies using attention mechanisms and hierarchical representations aligns closely with the needs of parsing tasks. Building on these principles, we plan to incorporate self-attention mechanisms inspired by transformer-based encoder-decoder models into our parsing frameworks. This integration aims to enhance the contextual representation of symbols and relationships, improving both segmentation and classification.

Graph-based Models. Representing mathematical formulas as graphs or trees is more natural than images or one-dimensional strings. For example, LaTeX expresses formulas as a type of SLT with font annotations. Mathematical expression recognition can alternatively be posed as filtering a graph to produce a maximum score or minimum cost spanning tree (MST) representing symbols and their associated spatial relationships. Eto et al. [39] were the first to take this approach, creating a graph with symbol nodes containing alternative labels, and candidate spatial relationships on edges with associated costs. Formula structure was obtained from extracting a minimum spanning tree amongst the relationship edges, along with minimizing a second measure of cost for the global formula structure.

For graph-based parsing, an input graph defined by a complete graph connecting all primitives is natural. However, this leads to lots of computation, and makes statistical learning tasks challenging due to input spaces with high variance in features: this motivates reducing variance through strategic pruning of input graph edges. Line-of-sight (LOS) graphs select edges where strokes in a handwritten formula or connected components in an image are mutually visible [52, 71]. Systems such as Hu et al. [50], LPGA [71], and QD-GGA [72] use this LOS input graph constraint, and select the final interpretation as a directed Maximum Spanning Tree (MST). QD-GGA [72] extracts formula

structure using an MST over detected symbols, extending previous approaches [71,119] by adding a multi-task learning (MTL) framework, and graph-based attention used to define visible primitives in images used to generate visual features for classification. QD-GGA uses Edmond's arborescence algorithm [38] to obtain a directed Maximum Spanning Tree (MST) between detected symbols, maximizing the sum of spatial relationship classification probabilities obtained from an end-to-end CNN network with attention. Using directed MSTs allows many invalid interpretations to be pruned, as the output graph is a rooted directed tree (as SLTs are).

The use of graph neural networks with attention has been seeing more use in math parsing to capture context between primitives using graph edges directly. For instance, Peng et al. [97] use a gated graph neural network (GGNN) in the encoder stage as a message passing model to encode CNN features with visual relationships in the LOS graph. Wu et al. [137] use GNN-GNN encoder-decoder (modified Graph Attention Network [129] encoder, modified Graph Convolution Network [56] decoder) to utilize graph context. Tang et al. [123] aims to learn structural relationships by aggregating node and edge features using a Graph Attention Network to produce SLTs by simultaneous node and edge classification, instead of the sequence representation used for encoder-decoder models. They produce the output SLT by filtering 'Background' nodes and 'No-Relation' edges. Note the contrast with the use of Edmonds' algorithm in QD-GGA and LGAP, where all primitives are assumed to belong to a valid symbol, and an MST algorithm selects directed edges between symbols obtained after merging primitives predicted to belong to the same symbol to produce an SLT.

Xie and Mouchère [139] present a novel stroke-level graph labeling approach for handwritten mathematical expression recognition using an edge-weighted graph attention network (EGAT). This method builds a graph representation of handwritten expressions where nodes represent individual strokes and edges capture their spatial and temporal relationships. Unlike encoder-decoder architectures, this end-to-end model directly extracts features from strokes using a graph embedding network and refines node and edge attributes through EGAT, effectively capturing structural dependencies. This approach demonstrates improved accuracy in both stroke classification and relation prediction on CROHME 2023 [140] dataset without pre-training. Building on this, the same authors extend their model to incorporate both local and global context through a stroke-level Graph-to-Graph Modeling framework using a master node architecture (GGM-EGAT) [141]. Their approach integrates node and edge embeddings using an edge-weighted attention mechanism that dynamically includes edge features during message passing and feature aggregation. The attention mechanism jointly updates node and edge representations, with a message concatenation step that reinforces mutual dependency between connected elements. The global variant further introduces a

virtual master node to facilitate long-range interactions, improving structure recognition accuracy on complex handwritten expressions.

Graph-based parsing is more natural for mathematical expressions. Unlike encoder-decoder models, the mapping between the input and the output can be obtained from labels assigned in the output to the nodes (strokes or connected components) and edges provided in the input [155]. Error metrics at the symbol and primitive levels for segmentation, symbol classification, and relation classification can be computed directly from the labeled output graphs. Also, like encoder-decoder models, these techniques do not require expression grammars, requiring only a vocabulary of symbol and relationship types. However, these models, although fast with easier interpretability, have not been able to match the accuracy of encoder-decoder models. Possible reasons may include a lack of global context, attention, and spatial information in the current models.

Relation to our work. In our work, the Line-of-Sight Graph Attention Parser (LGAP) formulates the problem of parsing mathematical formulas as a graph search problem. Input diagrams are represented as Line-of-Sight (LOS) graphs, where nodes correspond to visual primitives (e.g., strokes, connected components, or characters), and edges represent potential spatial relationships between these primitives and merge edges. By leveraging CNNs and Graph Attention Networks (GATs), the model classifies segmentations, node types (symbols) and edge labels (spatial relationships).

The parsing process involves reducing the LOS graph to a Symbol Layout Tree (SLT) by retaining the most probable edges and their associated classifications. This is achieved using Edmonds' algorithm, which extracts a Maximum Spanning Tree (MST) from the LOS graph by maximizing relationship probability distributions. The MST construction ensures that the resulting graph is rooted and directed, representing the correct hierarchical structure of the expression.

Our approach captures both local and global dependencies through graph-based modeling, combining the strengths of spatial relationships encoded in LOS graphs with the contextual representation power of GATs. This integration enables accurate classification of nodes and edges while reducing the complexity of graph search, resulting in interpretable and efficient parsing of mathematical formulas.

Summary and Baseline Selection. For final comparisons in our evaluation of mathematical formula parsing, we select representative baseline systems across both handwritten and typeset domains, chosen for their performance, relevance, and methodological diversity. On the CROHME 2019 benchmark for handwritten expressions, we include USTC-iFLYTEK [73] and ICAL [170] as strong encoder-decoder baselines, along with the top-performing rule-based systems such as

MyScript and Samsung [73]. We also compare against QD-GGA [72], as we build upon this model. Likewise, we include other GNN based models, including GGM-EGAT [141], which incorporates joint edge-aware attention with global context propagation via a master node. These systems provide coverage of both symbolic sequence generation and node-edge classification paradigms. On the Im2LaTeX-100K dataset for typeset formula recognition, we compare against a range of encoder-decoder models including MathNet [110], and EDPA [69], selected for their state-of-the-art BLEU and image-level accuracy. We also include a GNN-based model—Im2Latex-GNN [97] to situate our graph-based EGATv2 model within the broader landscape. These baselines were selected to test scientific hypotheses concerning model interpretability, data efficiency, structural recognition, and the tradeoffs between symbolic sequence decoding and graph-based structural prediction.

2.2.2 Chemical diagram parsing

We begin by surveying approaches to parsing molecular structure, categorizing them into (1) rule-based systems, and (2) neural-based systems. For neural-based systems, we further divide these into methods that produce string representations of structure (e.g., SELFIES [57], DeepSMILES [91], or InChI [47, 48]) and methods that produce graph representations of structure. We conclude by comparing and contrasting previous work with the ChemScraper parsers.

Rule-Based Parsers. The earliest parser for chemical diagrams in printed documents we know of is a rule-based parser by Ray et al. from the late 1950's [104]. This approach first detected atoms in scanned document images, and then connections between atoms were identified in the regions between atoms. connections for atoms were used to determine the type of bonds, which worked well for common compounds.

An important later development was the creation of the Kekulé system [78]. Kekulé adds additional pre-processing and improved visual detection of bond types over previous methods. Kekulé used thinning and vectorization of raster scans to eliminate variations in bond lines and characters, and ensured that a consistent set of characters and lines were recovered. Once a connection between a pair of atoms was established, the system visually detected the bond type instead of using chemical rules as Ray et al. did. In the same period, CLiDE [53] added the use of connected component analysis in disconnected bond groups to identify bond types. The final adjacency matrix for structure was created similar to Kekulé. Another system by Comelli et al. [26] used additional processing to identify charges as subscripts or superscripts attached to atoms.

A still-popular open-source system extending the rules of CLiDE and Kekulé is OSRA by Filipov

et al. [41]. OSRA refined processing of raster images generated from born-digital documents, which tend to have clearly rendered text lines, characters, and graphics. A similar system is MolRec [108], which uses horizontal and vertical grouping to detect connected atoms, their charge, and stereochemical information. The more recent CSR system [19] also uses rule-based graphical processing to output SMILES representations for molecules, using the *OpenBabel* [92] toolkit to generate a valid connectivity table.

Relation to our work. Our MST-based approach builds upon traditional rule-based systems, employing geometric rules and constraints to ensure chemically valid parsing. The process starts with constructing an MST using proximity and angular constraints to connect primitives. Transitioning to the visual graph involves refining edges to handle over-segmentation and disconnected components. In the tokenized visual graph stage, primitives are grouped into symbols like atom labels and bond types using rules for line lengths, character placements, and stereochemistry detection. Finally, semantic analysis converts the tokenized visual graph into a molecular graph, identifying hidden elements (e.g., implicit carbon atoms) and mapping named structures (e.g., "NO_2") to their subgraphs.

This rule-driven pipeline ensures robust parsing while maintaining interpretability. To handle diverse diagram styles, we use the parser to generate ground truth data for training neural network-based visual parsers, enabling improved generalization and accuracy across varied molecular diagrams.

Neural Networks.

String Output. Recent advances in neural networks have proven effective for parsing chemical diagrams. For example, Staker et al. [117] use an end-to-end model for extracting molecular diagrams from documents and converting them into SMILES strings. For diagram extraction, they used a U-Net [105] to segment diagrams, which were then passed through an attention-based encoder network [128] to generate a SMILES string representing molecular structure from the segmented image.

DECIMER [103] also uses an encoder-decoder model for extracting molecular structure from raster images. In their work they explored using different structure representations, including SMILES, DeepSMILES, and SELFIES. They found that SELFIES produced stronger results because of the additional information encoded in comparison with SMILES strings. Rajan et al. present an improved encoder-decoder model [102] for recognizing hand-drawn chemical structures. The model integrates a CNN encoder with a transformer decoder to convert images into SMILES strings. Synthetic datasets were generated using the RanDepict toolkit [16] to mimic hand-drawn styles,

enabling training on a diverse array of molecular representations. This enhanced DECIMER architecture improves robustness against variations in handwriting styles, line thickness, and background noise, achieving 73.25% accuracy and a Tanimoto similarity of 0.94 on the DECIMER Hand-Drawn dataset.

Additional encoder-decoder parsers include IMG2SMI by Campos et al. [20] which uses a Resnet-101 [45] backbone to extract image features. Li et al. [64] modified a TNT vision transformer encoder [43] by adding an additional decoder. This use of a vision transformer was made possible by the BMS (Bristol–Myers–Squibb) dataset [1] released by Kaggle, which provided a larger baseline for the conversion of molecule images to InChI (International Chemical Identifier names). The training dataset used by Li et al. contained 4 million molecule images. Similarly, SwinOCSR by Xu et al. [146] used the Swin transformer to encode image features and another transformer-based decoder to generate DeepSMILES, and used a focal loss to address the token imbalance problem in text representations of molecular diagrams.

MPOCSR [65] introduces a multi-path Vision Transformer (MPViT) [61] as the backbone for feature extraction and combines it with a transformer-based encoder-decoder architecture to generate SMILES sequences. The multi-path design integrates local features from convolutions with global information from the transformer, improving the representation of both coarse and fine-grained image details. To handle the long-tail distribution of chemical elements in datasets, the model employs a class-balanced (CB) loss function, ensuring improved predictions for rare elements like 'Br' and 'Cl'. The training dataset consists of 2 million molecule images, including Markush and non-Markush structures, generated using tools like CDK and Randepict [16]. The MPOCSR achieves superior performance with an accuracy of 90.95%, outperforming previous models such as Image2SMILES [55], DECIMER-V2 [103], and SwinOCSR [146].

Graph Output. String representations of molecular structure lack direct geometric representation between input objects (e.g., atoms and bonds) and the output strings, and models trained upon them require extensive training data [79]. In recent years, molecular diagram parsers that combine rule-based and neural-based approaches and generate graph representations have emerged. These methods usually employ a graph decoder or graph construction algorithm.

MolScribe [100] uses a SWIN transformer to encode molecular images and a graph decoder consisting of a 6-layer transformer to jointly predict atoms, bonds, and layouts, yielding a 2D molecular graph structure. They also incorporate rule-based constraints for chirality (i.e., 3D topology) and algorithms to expand abbreviations.

MolGrapher [79] is another method employing a graph-based output representation. It utilizes a ResNet-18 backbone to locate atoms, and constructs a supergraph incorporating all feasible atoms and bonds as nodes, which is then constrained. Subsequently, a Graph Neural Network (GNN) is applied to the supergraph, accompanied by external Optical Character Recognition (OCR) for node classification. Both these systems utilize multiple data augmentation strategies, including diverse rendering parameters, such as font, bond width, bond length, and random transformations of atom groups, bonds, abbreviations, and R-groups (i.e., abbreviations for 'rest of molecule') to bolster model robustness.

Likewise, Yoo et al. [149] and OCMR [134] produce graph-based outputs directly from molecular images. Yoo et al. [149] leverage a ResNet-34 backbone, followed by a Transformer encoder equipped with auxiliary atom number and label classifiers. A transformer graph decoder with self-attention mechanisms is used for bonds. In contrast, Wang et al. [134] employ multiple neural network models for different parsing steps. These steps include key-point detection, character detection, abbreviation recognition, atomic group reconstruction, atom and bond prediction. A graph construction algorithm is subsequently applied to the outputs.

Chen et al. [24] introduce MolNexTR, a model combining ConvNext [68] and Vision Transformer (ViT) [35] in a dual-stream encoder for molecular structure recognition. The encoder captures local atom-level features and long-range interatomic relationships. A two-stage decoder predicts atoms and bonds to construct molecular graphs, while post-processing integrates chemical rules for stereochemistry and abbreviation expansions, ensuring valid SMILES outputs. They use diverse augmentation strategies (e.g., rendering, image perturbation, and contamination algorithms) and integrate chemical knowledge. They achieve state-of-the-art results on the standard datasets, including USPTO, CLEF, UOB, JPO, Staker, ACS, showcasing its robustness to diverse molecular styles and noise.

These graph-based methods offer improved interpretability and robustness, and represent chemical structures naturally. In particular, atom-level alignment with input images facilitates easy examination, geometric reasoning, and correction of predicted results.

Relation to our work. Our approach bridges born-digital and visual parsers by representing inputs and outputs consistently as graphs. For the visual parser, we utilize a CNN-based segmentation-aware network that processes molecular diagrams as graphs with visual primitives like lines and characters as nodes, and spatial and relational connections as the edges. The network iteratively classifies nodes and edges to construct the tokenized visual graph, which is converted to a molecular graph as output. Unlike string-based methods, the graph representation provides a direct mapping

between input primitives and predicted chemical structures, facilitating geometric reasoning, error correction, and a more natural representation of molecular diagrams.

Summary and Baseline Selection. For final evaluation, we compare our chemical diagram parser against a broad spectrum of systems spanning rule-based, neural network-based, and graph-based approaches. On the USPTO synthetic dataset, we include rule-based systems such as MolVec [41], OSRA [41], and Imago [19], which historically achieve high SMILES match accuracy and serve as strong traditional baselines. Among neural network models, we benchmark against DECIMER [102, 103] and Img2Mol [117], both representative of modern encoder-decoder approaches that convert diagram images to SMILES strings. These baselines are chosen for their accessibility and widespread adoption in open-source chemical structure recognition pipelines.

From the graph-based category, we include SwinOCSR [146], MolScribe [100], and MolGrapher [79], which represent state-of-the-art performance on benchmark datasets and produce explicit graph outputs. These systems were selected due to their alignment with our methodological goals: structure-level prediction, support for complex layout and stereochemistry, and interpretability of graph outputs. In addition, OCMR [134] is included as a hybrid model using multiple neural components and a graph construction algorithm. Our evaluations on both synthetic (USPTO) and scanned (CLEF-2012) datasets focus on exact SMILES accuracy and data efficiency, reflecting both structural fidelity and generalization under limited supervision. The selected baselines allow for a comprehensive comparison across accuracy, architectural design, and sample efficiency dimensions.

2.3 Techniques Relevant to Parsing Models

Parsing models for mathematical and chemical diagrams leverage a variety of techniques to enhance their accuracy, efficiency, and interpretability. These methods address challenges such as learning from multi-task objectives, incorporating global context, and effectively utilizing edge features and attention mechanisms in graph-based representations. Additionally, innovative approaches like local graph constraints, modified self-attention for transformers, and optimized loss functions contribute to improved performance. In this section, we discuss existing works that provide the foundation for these techniques, highlight their relevance to parsing tasks, and outline how our work builds upon or diverges from these methods.

2.3.1 Multi-task Learning and Interaction

Multi-task learning (MTL) is a machine learning paradigm where multiple related tasks are learned simultaneously, allowing the model to leverage shared representations across tasks. This approach has been shown to improve generalization and reduce overfitting by utilizing shared knowledge between tasks [21, 106]. MTL can be seen as a form of inductive transfer, where knowledge gained from auxiliary tasks provides a beneficial inductive bias. This bias encourages the model to favor hypotheses that can effectively address multiple tasks simultaneously. For instance, introducing sparsity through techniques like ℓ_1 regularization is a common example of inductive bias. In MTL, the auxiliary tasks serve as the source of this bias, guiding the model toward solutions that not only optimize the primary task but also generalize effectively across related tasks [106].

MTL enhances model effectiveness by leveraging various mechanisms. It serves as an indirect form of data augmentation, using multiple tasks to reduce the impact of noise and improve generalization. By combining information from related tasks, MTL helps the model prioritize important features, especially in noisy or high-dimensional data. It also allows the model to benefit from complementary tasks, where features that are easy to learn in one task can support another task's learning. Additionally, MTL encourages the development of shared representations that align with multiple tasks, enabling better adaptability to new scenarios. Lastly, MTL reduces overfitting by introducing constraints that limit the model's tendency to adapt to noise in the data.

In MTL, hard parameter sharing involves sharing hidden layers among all tasks while maintaining task-specific output layers, effectively reducing the risk of overfitting by encouraging a shared representation across tasks [14, 21]. In contrast, soft parameter sharing [36, 147] assigns separate models to each task and regularizes the distance between their parameters, promoting similarity while allowing greater flexibility in task-specific learning.

Recent MTL approaches typically generate predictions for all tasks directly from the input in a single processing cycle, either in parallel or sequentially. However, this approach often overlooks useful relationships between tasks, such as the alignment of depth discontinuities with semantic edges, potentially limiting performance improvements [142, 167]. To address this, newer methods employ initial task predictions to refine outputs in subsequent steps. For example, Xu et al. [142] used spatial attention to incorporate features from initial predictions into a residual for improving task outputs. Zhang et al. [167] proposed a sequential prediction method, leveraging earlier task predictions to refine features for other tasks iteratively. Building on this, later works employed recursive procedures to propagate cross-task and task-specific patterns, focusing on affinity matrices of initial predictions

rather than directly refining features [168]. MTI-Net [127] models task interactions across multiple scales through three key components: a multi-scale multi-modal distillation unit that explicitly models task interactions at each scale, a feature propagation module that transfers distilled task information from lower to higher scales, and a feature aggregation unit that combines refined task features across scales to produce final predictions. It demonstrates improved performance, reduced memory usage, and fewer computations compared to single-task learning methods, as validated on multi-task dense labeling datasets.

Relation to our work. Building on this foundation, we propose extending the MTL framework in new directions to further enhance the interplay between subtasks. Specifically, we plan to integrate self-attention mechanisms where classifications from the initial execution of the network are used as inputs for subsequent iterations. This iterative refinement process allows the model to progressively improve its predictions by learning from prior outputs. These enhancements are designed to exploit the interdependencies between subtasks further, enabling the model to capture contextual information and reduce ambiguities in parsing.

2.3.2 Local Constraints in Graph-Based Methods

In graph-based learning, incorporating local constraints has proven to be an effective strategy for reducing computational complexity and enhancing interpretability. These constraints focus on limiting the connections between nodes in a graph to those that are most relevant, improving the graph's sparsity and reducing noise. This paradigm has been explored across diverse applications, such as molecular graph parsing, road network extraction, and subgraph learning.

MolGrapher [79] and MolScribe [100] leverage local constraints to improve molecular graph parsing by limiting node connectivity based on domain-specific properties. For instance, nodes representing atoms and bonds are connected only if they meet spatial adjacency criteria, ensuring that the resulting graphs remain chemically valid. This approach allows the models to focus on meaningful local interactions, such as bond angles or connections, while ignoring irrelevant global connections.

Bahl et al. [10] apply local constraints to road network extraction by utilizing k-nearest neighbor graphs. By connecting each node to its k closest neighbors, the model captures the local structure of road networks, such as intersections and curves, while maintaining sparsity. This method effectively reduces the number of edges in the graph, leading to faster computation and better generalization.

Li et al. [63] propose sparse subgraph learning methods, which introduce local constraints by dynam-

ically pruning edges during training. This ensures that the graph retains only the most important connections, balancing the trade-off between sparsity and informativeness. Such techniques are particularly useful in high-dimensional datasets, where densely connected graphs can be computationally prohibitive and prone to overfitting.

Relation to our work. In parsing mathematical and chemical diagrams, local constraints are crucial for constructing sparse yet informative graphs. In our Line-of-Sight Graph Attention Parser (LGAP) [114], we apply LOS constraints [52, 71] to pruned edges from complete graph. LOS graphs include edges only between mutually visible strokes in handwritten formulas or connected components in images. This reduces computational complexity and improves statistical learning by lowering feature variance in the input space.

Similarly, in our Line-of-Sight Chemical Graph Parser (LCGP) [112], used for molecular diagram parsing, we restrict each primitive's LOS connections to its $k = 6$ nearest neighbors. This accommodates the four-bond constraint in molecular structures—ensuring that at most four lines or characters represent bonds—while accounting for potential over-segmentation of visual primitives, such as fragmented bond lines or characters. This approach enables the model to focus on relevant local interactions and effectively handle over-segmented inputs, ensuring both computational efficiency and improved accuracy.

2.3.3 Use of Edge Features in Graph Parsing

In graph-based parsing, the inclusion of edge features has emerged as a crucial advancement for improving model performance. Edge features provide additional contextual information about the relationships between nodes, enhancing the model's ability to capture structural dependencies within the graph. This idea has been effectively integrated into several models [33, 42, 139, 148].

Graphormer [148] incorporates edge features directly into self-attention layers using an edge encoding mechanism. This mechanism integrates the features of edges along the shortest path between nodes, represented mathematically as:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij},$$

$$c_{ij} = \frac{1}{N} \sum_{n=1}^{N} x_{e_n} \left(w_n^E\right)^T,$$

where: h_i and h_j are the hidden state vectors of nodes i and j, respectively, W_Q and W_K are learnable weight matrices for the query and key transformations, $\sqrt{d}$ is a scaling factor, where d is the dimensionality of the query and key vectors, and $b_{\phi(v_i,v_j)}$ is a bias term that encodes the shortest path distance $\phi(v_i, v_j)$ between nodes v_i and v_j, providing structural information.

The term c_{ij} aggregates the weighted edge features along the shortest path SP_{ij} as follows:

$$c_{ij} = \frac{1}{N} \sum_{n=1}^{N} x_{e_n} \left(w_n^E\right)^T,$$

Here, N is the total number of edges along the shortest path SP_{ij}, x_{e_n} is the feature vector of the n-th edge e_n in SP_{ij}, $w_n^E \in \mathbb{R}^{d_E}$ is the learnable embedding vector for the n-th edge feature, and d_E is the dimensionality of the edge feature embedding. By encoding edge features using this mechanism, Graphormer enhances attention weights with structural information, capturing connectivity patterns and improving the graph representation for downstream tasks.

Relational Transformers (RT) [33] extend transformer architectures by conditioning attention mechanisms on edge vectors. Specifically, RT concatenates the edge vector e_{ij} with the node feature vectors Q, K, V before performing linear transformations, effectively encoding the relationship between nodes directly into the attention computation. This approach not only improves performance but also introduces a multi-task interaction between nodes and edges, leveraging edge features to refine node embeddings.

Relation to our work. Building on this foundation, we propose enhancing our parsing models by incorporating edge-specific attention mechanisms. These mechanisms will dynamically update edge features during iterative passes, similar to nodes, refining the representation of relationships between symbols and components. Additionally, we plan to introduce a bidirectional flow of information by incorporating edge features into node representations and vice versa. This mutual exchange will strengthen the interaction between edges and nodes within the MTL framework

2.3.4 Graph Attention Methods

Attention mechanisms have been instrumental in advancing graph-based parsing by enabling models to dynamically focus on meaningful relationships between nodes. This subsection explores various graph attention methods, from standard transformer-based approaches to graph-specific adaptations, and their applications in parsing mathematical and chemical diagrams.

Standard Graph Attention Networks. Graph Attention Networks (GATs) [129] extend the traditional attention mechanism from transformer architectures to graph data. By dynamically computing attention weights based on node features and graph connectivity, GATs allow nodes to attend to their neighbors, effectively capturing local contextual relationships. This makes GATs particularly suitable for graph-based parsing tasks, where local dependencies are crucial. GATv2 [17] improves upon the original Graph Attention Network (GAT) by introducing a more expressive and dynamic attention mechanism. In GAT, attention scores are computed using static weights, limiting its ability to adapt to varying graph structures. GATv2 addresses this limitation by modifying the computation of attention coefficients, making them a function of the target and source node features, rather than the source features alone. This dynamic approach ensures that the attention mechanism captures richer relationships between nodes.

Mathematically, GATv2 computes the attention coefficients as:

$$\alpha_{ij} = \text{softmax}_j \left(\mathbf{a}^T \sigma \left(\mathbf{W} \cdot [\mathbf{h}_i \,\|\, \mathbf{h}_j] \right) \right),$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the feature vectors of nodes i and j, $\mathbf{W}$ is a learnable weight matrix, $\mathbf{a}$ is a learnable attention vector, σ is an activation function (e.g., LeakyReLU), and $\|$ denotes concatenation. This formulation ensures that the attention mechanism dynamically adapts to both the target and source nodes' features.

The dynamic nature of GATv2 allows it to better capture complex node relationships and adapt to heterogeneous graph structures. Despite these enhancements, GATv2 retains the same computational efficiency as GAT, making it a powerful tool for tasks requiring flexible and expressive graph representations.

Modified Self-Attention in Graph Transformers. Graph-structured data poses unique challenges for standard transformers due to its non-Euclidean nature, where structures lack a fixed order and dimensionality [37]. To address this, graph transformers incorporate graph-specific inductive biases, such as node position and edge structural information, into the self-attention mechanism [115]. These biases can be categorized into local and global attention mechanisms, which adaptively encode both fine-grained and high-level graph structures.

Graphormer [148] introduces centrality and spatial encodings into the attention computation to capture graph structure. Centrality encoding quantifies node importance using in-degree and out-degree, while spatial encoding integrates shortest path distances between nodes as a bias term in self-attention. By avoiding traditional positional encodings, Graphormer enables adaptive attention based on graph-specific relationships, effectively encoding structural information. Centrality

encoding captures the importance of a node based on its in-degree and out-degree:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)},$$

where x_i is the initial node feature, and $z^-_{\deg^-(v_i)}$ and $z^+_{\deg^+(v_i)}$ are learnable parameters corresponding to the in-degree and out-degree of node v_i, respectively.

Spatial encoding integrates shortest path distances (SPD) between nodes as a bias term in the self-attention mechanism:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)},$$

where $h_i W_Q$ and $h_j W_K$ are the query and key projections of nodes i and j, d is the dimensionality of attention vectors, and $b_{\phi(v_i,v_j)}$ encodes the SPD between v_i and v_j.

The GTMGC [143] model adapts the standard transformer self-attention mechanism to molecular graphs by introducing Molecule Structural Residual Self-Attention (MSRSA), which incorporates adjacency matrices (A) and row-subtracted interatomic distance matrices ($D_{\text{row-sub}}$) as residual biases in the attention scores. The local attention component uses adjacency matrix information scaled by a learnable parameter, ensuring local structural relationships are captured, while the spatial attention component incorporates distance relevance by applying a bias based on $D_{\text{row-sub}}$, modulated by another learnable parameter. The final attention computation combines these structural biases with global attention to adaptively weigh nodes and edges in the graph. This modification enables MSRSA to effectively model both local and global dependencies, making it particularly suited for molecular graph parsing.

Relation to Our Work. In our work, we leverage Graph Attention Networks (GATs) [129] in LGAP [114] to update node features using edge adjacency matrices. The aggregated features are concatenated with the original features before passing through linear layers to improve feature representation. Currently, LCGP does not employ graph attention mechanisms.

We propose to integrate GATv2 [17] and adapt the modified self-attention mechanism of graph transformers to enhance both LGAP and LCGP. Specifically, we aim to utilize the self-attention and cross-attention mechanisms from graph transformers within the MTL framework, enabling the model to dynamically capture task-level interactions and learn the importance of tasks for each other.

2.4 Evaluation Metrics

Evaluating the performance of a parsing system for mathematical formulas and chemical diagrams requires well-defined metrics that quantify the accuracy, completeness, and correctness of the generated outputs. These metrics are crucial in assessing both the structure and content of the parsed diagrams, ensuring that the system not only recognizes the individual components but also correctly captures the relationships between them.

In the context of mathematical formula parsing, evaluation typically involves measuring the accuracy of the Symbol Layout Tree (SLT), which represents the hierarchical structure of the formula. Metrics such as symbol recognition rates, spatial relationship detection, and overall structure accuracy are used to gauge the system's performance. For chemical diagram parsing, evaluation metrics focus on the correctness of the molecular structure, often represented in formats like SMILES or molecular graphs. The evaluation process may include string-based metrics, such as exact matches or normalized distances, as well as graph-based methods that assess the accuracy of the atom and bond relationships.

This section outlines the primary evaluation metrics used for both domains, including graph-based methods, string-based methods, and other task-specific metrics that help to assess the accuracy and reliability of the parsing systems. By employing these metrics, we can determine how well the systems perform across different types of input, such as PDFs, raster images, and handwritten strokes.

2.4.1 Graph-based Metrics (LgEval)

Evaluating the accuracy of parsed visual structures in both mathematical formulas and chemical diagrams involves the use of graph-based metrics. The LgEval library[4] [84,85], provides a framework for comparing labeled graphs by examining how well the nodes (representing primitives or symbols) and edges (representing relationships) align between the ground truth and the parser's output. This method enables a fine-grained analysis of recognition errors, allowing us to pinpoint mistakes in both node labeling and relationship detection.

Labeled graphs defined over the *same* nodes with known input locations can be directly compared using their adjacency matrix entries. Recognition errors are easily identified by differing labels

[4] `https://gitlab.com/dprl/lgeval`

in adjacency matrix cells, and located within an input image using the node locations. With a particular bottom-up representation for grouping nodes (i.e., segmentation), errors may be identified even when node groupings disagree, or nodes are missing in one or the other graph [157].

Handwritten math formula recognition was evaluated in this manner for the early CROHME competitions, with ground truth and recognizer outputs defined over the same handwritten strokes [85]. The LgEval library was used to compute metrics and visualize errors [84, 85, 113]. One can view all errors using the `confHist` tool including missing nodes and relationships. Repeated errors for nodes, edges, and subgraphs are compiled in histograms that may be explored in HTML pages.

For example, in mathematical formula parsing, the nodes in the graph represent individual symbols, and the edges represent spatial relationships such as superscripts, subscripts, or horizontal adjacency. In chemical diagram parsing, the nodes represent atoms or bond segments, and the edges represent the connections between these elements, such as bond types (single, double, triple, etc.) or spatial connections between visual elements in a tokenized graph.

Evaluation Process: The evaluation process begins by assigning unique identifiers to the nodes in both the ground truth and parser output graphs. In mathematical formula recognition, these identifiers may correspond to the connected components (CCs) or strokes forming a symbol, while in chemistry, the identifiers may correspond to atoms or bond segments in a molecule. Once the nodes are aligned based on spatial overlap (for visual parsing) or direct matching (for structured data), adjacency matrices are constructed for both the ground truth and parser output.

Each row and column of an adjacency matrix corresponds to a node in the graph. Diagonal entries represent node labels (e.g., the type of symbol in math, or the atom in chemistry), while off-diagonal entries represent the edges between nodes (e.g., spatial relationships in math, or bond types in chemistry). Differences between the adjacency matrices of the ground truth and the parser output highlight recognition errors. For example, a mislabeled node (such as a misclassified symbol or atom - e.g., 'two' misclassified as 'z' shown in Figure 2.7) would appear as a mismatch in a diagonal entry, while an incorrect or missing spatial relationship (e.g., 'Horizontal' misclassified as 'Rsub' in Figure 2.7) or bond type would appear as an off-diagonal mismatch.

For each graph, the evaluation metrics typically include:

- **Unlabeled Structure Accuracy**: This measures whether the predicted graph has the correct structure without considering the labels of the nodes and edges. It ensures that the relationships between the primitives are captured accurately in terms of their connections. In

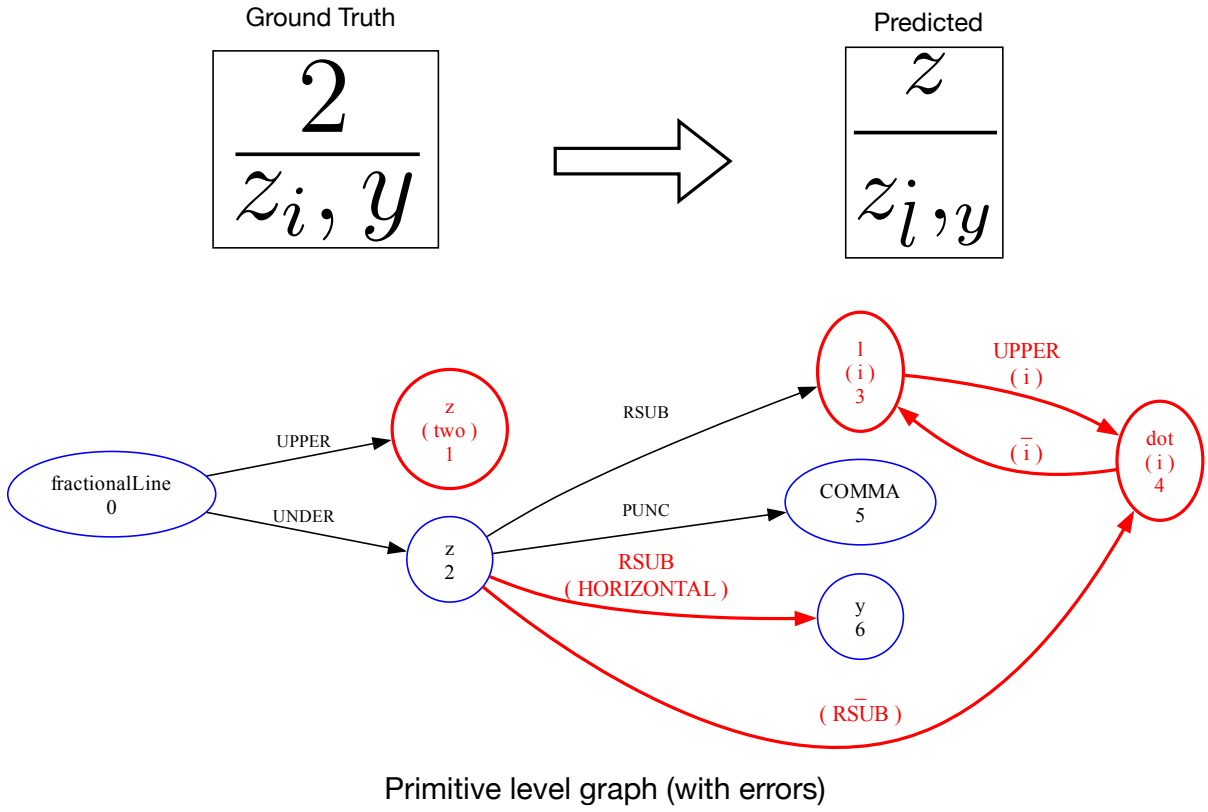


Figure 2.7: An example of a primitive-level graph representation highlighting errors in the predicted graph (right) compared to the ground truth (left) for a mathematical expression. The predicted graph shows incorrect or missing relationships (red edges) and mislabeled nodes (red nodes). Labels in brackets represent the ground truth, while underscores indicate missing elements or undefined labels.

Figure 2.7, the structure is incorrect, as there are missing edge between 'z' and upper part of 'i', due to over-segmentation error in 'i'.

- **Labeled Structure Accuracy**: This metric extends the unlabeled structure accuracy by evaluating whether the nodes and edges are correctly classified. For instance, in math, this means that the symbols are recognized correctly, and their spatial relationships (e.g., subscript, superscript) are accurate. In chemistry, this involves verifying that the atoms and bonds are labeled correctly in the tokenized visual graph or the molecular graph. In Figure 2.7, there are incorrect node and edge labels shown in red.

- **Detection F-scores**: This measures how well the system detects symbols (math) or atoms and bonds (chemistry) by comparing the number of correctly detected components to the total

number of components in the ground truth. The F-score is the harmonic mean of precision and recall, ensuring a balance between precision and recall. In Figure 2.7, we miss detecting the 'RSUB' edge between 'z' and upper part of 'i', reducing recall.

- **Detection + Classification F-scores**: This combines detection with classification. For a component to be considered correct, it must not only be detected but also classified accurately. This applies to both symbols and spatial relationships in math, and atoms and bond types in chemistry.

In chemical diagram parsing, this method is applied to both tokenized visual graphs and molecular graphs. For tokenized visual graphs, the relationships might represent binary connections between atoms and bonds, while for molecular graphs, the relationships represent specific bond types (e.g., single, double, or triple bonds). In mathematical formula parsing, the relationships are spatial relations like superscripts, subscripts, or horizontal adjacency.

A key difference between the two domains is how the relationships are interpreted. In math, spatial relationships between symbols play a central role in defining the formula's structure, whereas in chemistry, the connections between atoms and bonds define the molecular structure. However, both domains share the same underlying graph-based evaluation approach, making the LgEval framework versatile for assessing different types of visual structures.

By utilizing the LgEval library, we are able to carry out a comprehensive and detailed analysis of the recognition system's performance, identifying where errors occur and how they affect the overall accuracy of the parsing process.

2.4.2 String-based Metrics

In addition to graph-based metrics, string-based metrics are commonly used to evaluate how well systems translate visual or handwritten input into structured textual representations like LaTeX (for maths) or SMILES (for chemistry).

LaTeX Match

For mathematical formulas, the **exact LaTeX match** is a key metric. This metric measures the percentage of formulas where the automatically generated LaTeX code exactly matches the ground truth LaTeX representation. The comparison is strict, meaning even minor differences in formatting

(such as missing braces, incorrect subscript or superscript placement, or small spacing errors) will result in a mismatch.

Limitations of LaTeX Match: A key limitation of the LaTeX match metric is that the same formula can often be represented by multiple equivalent LaTeX strings. For instance, variations in spacing, the use of optional braces, or alternative ways of expressing subscripts and superscripts can result in different LaTeX code that still represents the same visual structure. Despite this, only an exact match will count as correct in this metric, making it overly rigid in some cases. Additionally, LaTeX match does not provide fine-grained feedback like graph-based metrics. It only indicates whether the entire formula was generated correctly or not, without giving insight into which specific parts of the formula (symbols or spatial relationships) were misrecognized.

SMILES Match and Normalized Levenshtein Distance

SMILES strings are compared by (1) the percentage of exact matches, and (2) the *inverse* of the average Normalized Levenshtein Distance (NLD). The *levenshtein distance* is the minimum number of insertions, deletions, or substitutions needed to convert one SMILES string to the other [111]. The distance is normalized to $[0, 1]$ using the minimum/maximum possible edits based on the SMILES string lengths. The inverse of the average NLD is given by subtracting the average NLD from 1, giving a *similarity* in $[0, 1]$, with 1 produced for identical SMILES strings.

Limitations.

Molecular formulas are naturally represented as graphs, where atoms and bonds have well-defined relationships and spatial arrangements. In contrast, SMILES representations are linear character strings *describing* graph structure. These SMILES characters have no direct connection with the atoms and bonds present in an input image (i.e., where atoms appear is not represented).

Levenshtein distances for SMILES strings may correspond to multiple operation sequences of the same length. In this case, Levenshtein-based SMILES metrics do not uniquely identify which parts of the input are incorrectly recognized. It is thus tempting to instead use graph edit distances over molecule structure graphs directly, with operations that insert/delete/relabel nodes and edges. Unfortunately, this can also result in ambiguous minimal edit sequences, and errors may again not be uniquely identified.

The main issue here is a missing correspondence between input image regions and the nodes/edges in a molecular structure graph representation. If molecular structure graphs include input image

locations (e.g., bounding boxes) their nodes may be aligned spatially and then compared using adjacency matrices. We describe the first application of this approach to chemical structure recognition evaluation next.

These string-based metrics are essential for assessing the accuracy and quality of systems that convert graphical or handwritten input into structured representations like LaTeX or SMILES. However, they are limited by their rigid nature and inability to provide detailed insights into specific recognition errors.

2.5 Summary

In this chapter, we explored the foundational elements required for understanding the parsing of both mathematical formulas and chemical diagrams. We first discussed the different representations and input types, including born-digital PDFs, typeset images, and handwritten strokes. Each input source brings unique challenges in extracting visual primitives, such as connected components (CCs), strokes, and lines, and forming them into structured graph representations. We introduced the notion of primitive-level graphs and how they differ between math and chemistry in terms of nodes and edges, as well as the construction of symbol layout trees (SLTs) for math and molecular graphs for chemistry.

We also outlined the evaluation metrics crucial for assessing the effectiveness of our parsing models. Graph-based metrics, using tools like LgEval, provide a fine-grained approach to evaluating structural and relational accuracy. In contrast, string-based metrics like LaTeX and SMILES provide quick, albeit limited, evaluations based on exact matches. These string-based methods, however, suffer from limitations in their ability to capture structural nuances, as discussed for both mathematical and chemical outputs. We also reviewed related works in the fields of mathematical formula and chemical diagram parsing, highlighting the evolution of techniques from syntactic and rule-based methods to more modern graph-based and neural network approaches. These works provide the foundation and context for our contributions, addressing limitations in existing models regarding accuracy, speed, and interpretability. This chapter provides the necessary background for understanding the technical and methodological context in which the proposed research operates.

In the next chapter, we delve into the born-digital parsing of formulas and diagrams directly from PDF symbols, leveraging the precise vector data available within PDFs to construct structured representations without relying on OCR. This transition forms the backbone of our approach to

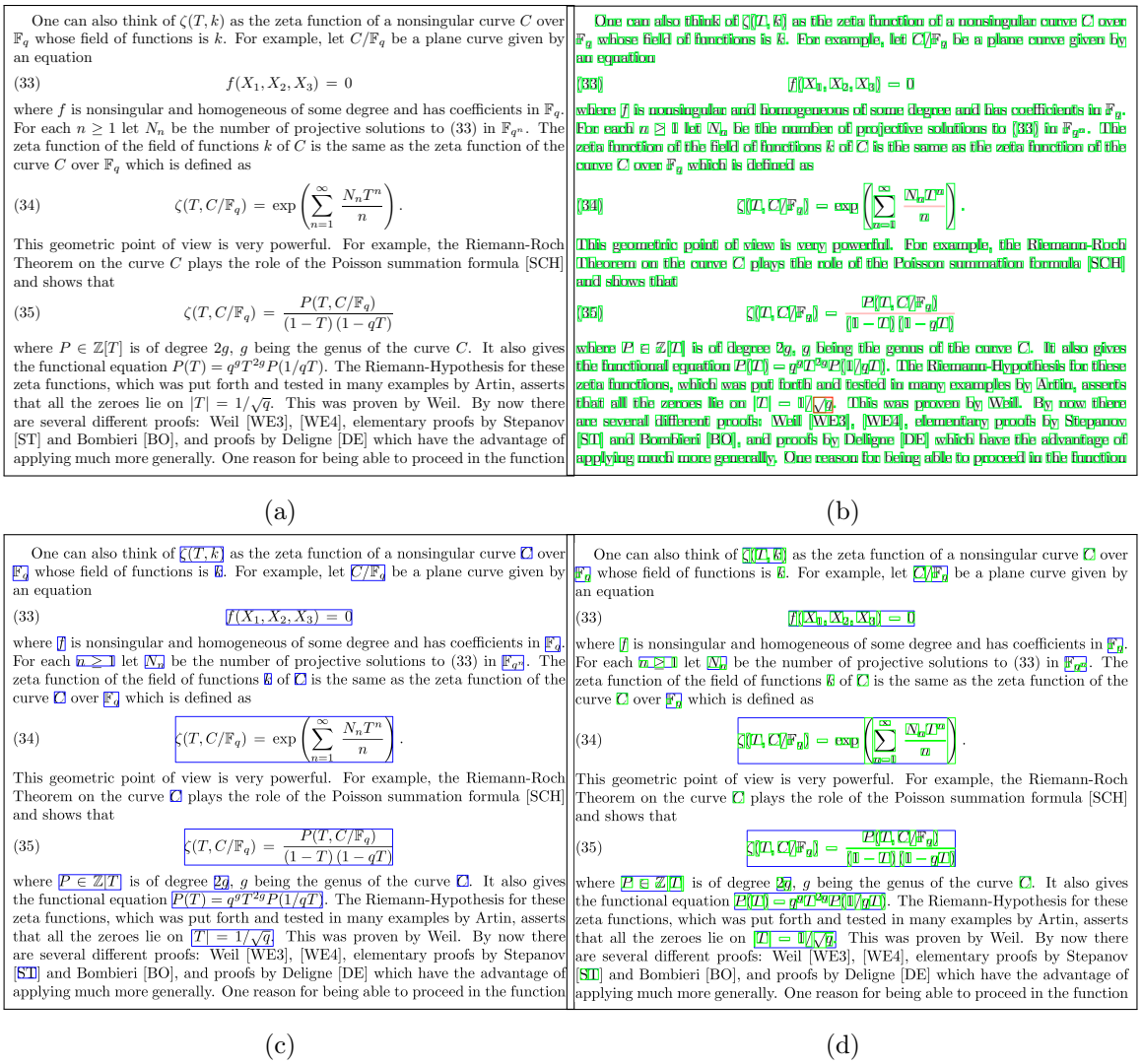
extracting high-quality data efficiently from documents.

Chapter 3

Born-digital Parsing from PDF symbols

This chapter focuses on the born-digital parsing of mathematical formulas and chemical diagrams from PDF files (ChemScraper [112]), utilizing vector-based information extracted directly from PDF drawing commands without relying on OCR. PDF documents encode rich graphical content, including symbols, lines, and other primitives, offering opportunities for precise and efficient extraction. By leveraging these native PDF instructions, we achieve higher parsing accuracy and efficiency compared to pixel-based methods. We were motivated to use PDF instructions by earlier math formula recognition work by Baker et al. using a combination of PDF instructions and image analysis [13]. In our approach, only PDF instructions are used, extracted by our `SymbolScraper` tool [113] without image processing.

For mathematical formulas, the goal is to recognize isolated formulas by constructing their structure as Symbol Layout Trees (SLTs), capturing the spatial arrangement of symbols such as superscripts, subscripts, and horizontal alignments [156] as described in Chapter 2. The isolated formulas are located using an existing tool–YOLOv8 [132] for formula detection, while precise symbol extraction, including bounding box locations and character labels, is handled by SymbolScraper [113]. SLTs provide a structured representation of formulas that can be converted into formats such as LaTeX or Presentation MathML. The challenge lies in accurately parsing complex formula structures with a large and diverse symbol vocabulary, while preserving spatial relationships among symbols. Recognizing these structures requires robust parsing techniques to interpret symbol arrangements and interactions effectively. To aid in diagnosing parsing errors and improving recognition accuracy, we utilize tools that visualize recognition outputs and errors through the LgEval library [73,83,85]. This library provides a convenient HTML-based visualization of recognition results, allowing for detailed



One can also think of $\zeta(T,k)$ as the zeta function of a nonsingular curve C over $\mathbb{F}_q$ whose field of functions is k. For example, let $C/\mathbb{F}_q$ be a plane curve given by an equation

$$f(X_1,X_2,X_3) = 0 \tag{33}$$

where f is nonsingular and homogeneous of some degree and has coefficients in $\mathbb{F}_q$. For each $n \geq 1$ let N_n be the number of projective solutions to (33) in $\mathbb{F}_{q^n}$. The zeta function of the field of functions k of C is the same as the zeta function of the curve C over $\mathbb{F}_q$ which is defined as

$$\zeta(T,C/\mathbb{F}_q) = \exp\left(\sum_{n=1}^{\infty} \frac{N_n T^n}{n}\right). \tag{34}$$

This geometric point of view is very powerful. For example, the Riemann-Roch Theorem on the curve C plays the role of the Poisson summation formula [SCH] and shows that

$$\zeta(T,C/\mathbb{F}_q) = \frac{P(T,C/\mathbb{F}_q)}{(1-T)(1-qT)} \tag{35}$$

where $P \in \mathbb{Z}[T]$ is of degree $2g$, g being the genus of the curve C. It also gives the functional equation $P(T) = q^g T^{2g} P(1/qT)$. The Riemann-Hypothesis for these zeta functions, which was put forth and tested in many examples by Artin, asserts that all the zeroes lie on $|T| = 1/\sqrt{q}$. This was proven by Weil. By now there are several different proofs: Weil [WE3], [WE4], elementary proofs by Stepanov [ST] and Bombieri [BO], and proofs by Deligne [DE] which have the advantage of applying much more generally. One reason for being able to proceed in the function

(a)

One can also think of $\zeta(T,k)$ as the zeta function of a nonsingular curve C over $\mathbb{F}_q$ whose field of functions is k. For example, let $C/\mathbb{F}_q$ be a plane curve given by an equation

$$f(X_1,X_2,X_3) = 0 \tag{33}$$

where f is nonsingular and homogeneous of some degree and has coefficients in $\mathbb{F}_q$. For each $n \geq 1$ let N_n be the number of projective solutions to (33) in $\mathbb{F}_{q^n}$. The zeta function of the field of functions k of C is the same as the zeta function of the curve C over $\mathbb{F}_q$ which is defined as

$$\zeta(T,C/\mathbb{F}_q) = \exp\left(\sum_{n=1}^{\infty} \frac{N_n T^n}{n}\right). \tag{34}$$

This geometric point of view is very powerful. For example, the Riemann-Roch Theorem on the curve C plays the role of the Poisson summation formula [SCH] and shows that

$$\zeta(T,C/\mathbb{F}_q) = \frac{P(T,C/\mathbb{F}_q)}{(1-T)(1-qT)} \tag{35}$$

where $P \in \mathbb{Z}[T]$ is of degree $2g$, g being the genus of the curve C. It also gives the functional equation $P(T) = q^g T^{2g} P(1/qT)$. The Riemann-Hypothesis for these zeta functions, which was put forth and tested in many examples by Artin, asserts that all the zeroes lie on $|T| = 1/\sqrt{q}$. This was proven by Weil. By now there are several different proofs: Weil [WE3], [WE4], elementary proofs by Stepanov [ST] and Bombieri [BO], and proofs by Deligne [DE] which have the advantage of applying much more generally. One reason for being able to proceed in the function

(c)

(b)

(d)

Figure 3.1: Detection of symbols and expressions. The PDF page shown in (a) contains encoded symbols shown in (b). (c) shows formula regions identified in the rendered page image, and (d) shows symbols located in each formula region.

examination of structure recognition errors organized by ground truth subgraphs. For example, symbol segmentation, symbol classification, and relationship classification errors are automatically highlighted and linked to specific input formula images in context, enabling targeted analysis and refinement of the recognition process.

In the case of chemical diagrams, current molecule structure recognizers generally parse images from pixel-based raster images, and produce chemical structure descriptions such as Simplified Molecular-

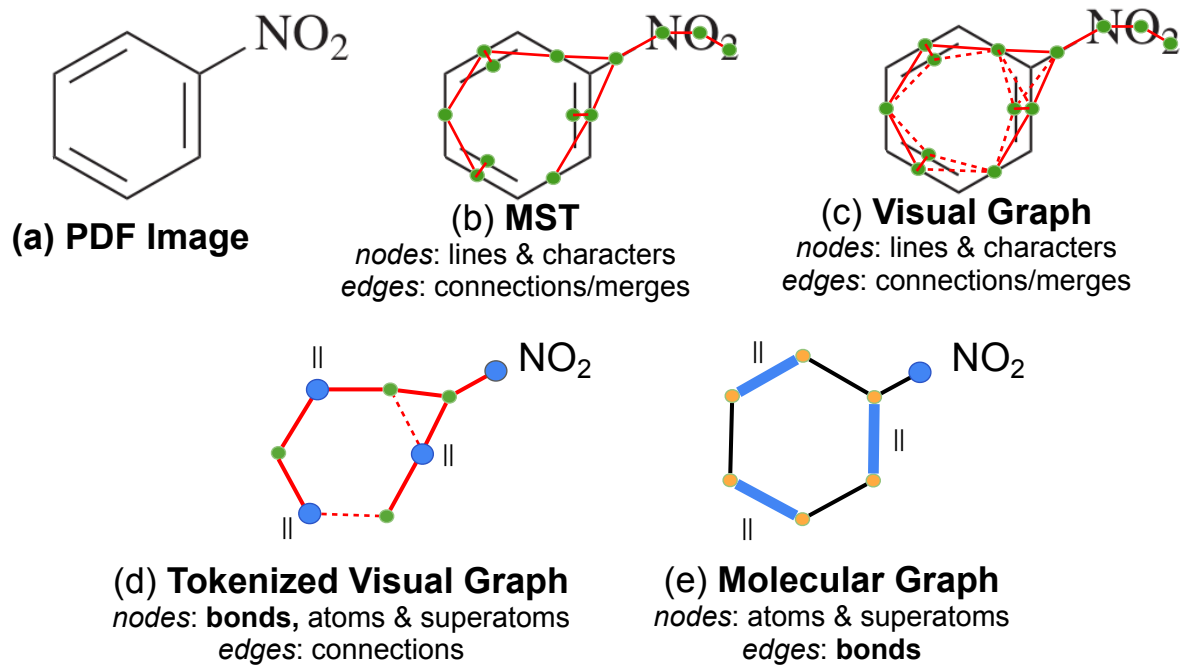


Figure 3.2: Parsing Nitrobenzene ($C_6H_5NO_2$) from a PDF image (a). (b) **Minimum Spanning Tree (MST)** over lines & characters. (c) **Visual Graph** with additional edges (dashed lines) (d) **Tokenized Visual Graph** with merged nodes (bonds and named groups). (e) **Molecular Graph**. Blue nodes show double bonds and atom/group names in (d) and (e). In (e) orange nodes are 'hidden' carbon atoms, and single/double bonds are converted from nodes to edges.

Input Line-Entry System strings (SMILES [135]) describing molecular structure as output. A number of these approaches work well, and some include modern variations of encoder/decoder models that recognize structure with high accuracy, as discussed in Chapter 2. However, modern documents often use vector images to depict molecules. Vector images encode diagrams as characters, lines, and other graphic primitives. So, parsing involves recognizing chemical elements, such as atoms, atom groups, and bonds, from PDF drawings. Unlike mathematical formulas, where symbols follow strict spatial hierarchies, chemical diagrams require identifying connections between atoms and bonds, potentially forming complex cyclic structures (e.g., aromatic rings). Our approach involves constructing a Minimum Spanning Tree (MST) over extracted primitives, followed by tokenization of molecular entities such as atoms and bonds. This tokenized visual graph is then transformed into a molecular graph representing the chemical structure, supporting downstream tasks such as chemical search and reaction planning.

The chemical diagram born-digital parser is used to annotate pixel-based raster images, to address

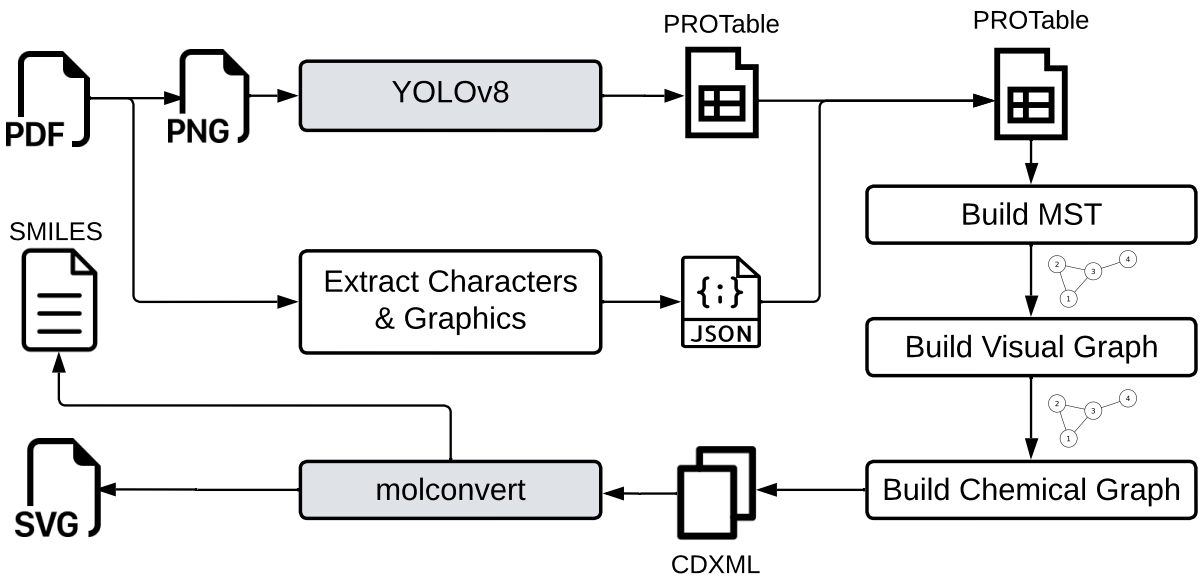


Figure 3.3: ChemScraper Born-Digital Pipeline. Molecules are detected in PNG page images, but symbols are extracted from PDF instructions. Page-Region-Object tables store bounding boxes and the graphics they contain. Molecules are recognized in three stages, producing CDXML containing the page location, appearance, and chemical structure for each. CDXML can then be converted to chemical structure file formats (e.g., SMILES) or rendered as images (e.g., SVG).

a shortage of training data. This includes annotations for all graphical primitives, atoms, and bonds (see Section 3.4). We use this data to train a new *visual* parser, a novel multi-task neural network for recognizing molecule diagrams in raster images (see Chapter 4). The born-digital parsers' use of Minimum Spanning Trees (MSTs) to recognize molecular diagrams is novel.

The *born-digital* vector image parser is one component in the online ChemScraper molecule extraction tool[1], which includes a YOLOv8 [132] detection module not described here. Figure 3.3 provides an overview of the full ChemScraper born-digital extraction pipeline. The model locates page regions where molecular diagrams appear, and then parses their structure. Recognized molecules are stored in ChemDraw[2] CDXML files [88]. CDXML represents both visual and chemical structure in molecular diagrams. The ChemAxon `molconvert` command line tool[3] is used to convert CDXML to vector images (SVG) and SMILES. Recognized molecules can then be used for editing, search, and other applications (e.g., in chemoinformatics).

[1] `https://chemscraper.frontend.staging.mmli1.ncsa.illinois.edu/configuration`
[2] `https://revvitysignals.com/products/research/chemdraw`
[3] `https://docs.chemaxon.com/display/docs/molconvert_index.md`

An important attribute of ChemScraper output graphs is that they contain both visual and chemical structure information. This allows output graphs to closely match their original appearance in addition to capturing chemical structure. The additional visual information is helpful both for reusing the appearance of molecules within documents, and for visualization and checking of recognition results.

In this chapter, we present the core components of the parsing pipelines for both mathematical formulas and chemical diagrams. For mathematical formulas, the process involves symbol extraction, formula region detection, and the generation of Symbol Layout Trees (SLTs) to capture structure. For chemical diagrams, characters and graphical elements such as lines are extracted to construct a tokenized visual graph, which is then converted into a molecular graph to represent chemical structures. Both approaches emphasize graph-based parsing techniques, leveraging vector-based PDF data for fast, accurate, and interpretable recognition.

The subsequent sections cover each step in detail, with the goal of producing structured graph representations: SLTs for mathematical expressions and molecular graphs for chemical structures. These graph representations not only capture the content and layout of the original diagrams but also enable various applications, including editing, searching, and chemical reaction planning, by converting them into formats such as LaTeX, MathML, CDXML, and SMILES.

In Section 3, we describe the Symbol Scraper, a tool designed to extract characters, graphics, and shapes from vectorized drawing instructions in PDF files. Section 3.1 introduces the MST-based math formula parser, which builds the Symbol Layout Tree (SLT) using Symbol Scraper and the QD-GGA approach. Section 3.3 focuses on the ChemScraper born-digital parser, which constructs molecular graphs through a streamlined and efficient design. The process begins with constructing a Minimum Spanning Tree (MST) from PDF graphical primitives, identifying neighboring elements, adding or removing edges to capture the diagram's visual structure, and grouping (or *tokenizing*) primitives into molecular entities, such as atoms and bonds. This tokenized visual graph is then transformed into a graph representing the molecular structure.

Section 3.4 explains the generation of annotated training data using the born-digital parser, focusing on extracting primitives for training visual parsers that recognize molecular structures from raw images. Finally, Section 3.5 presents the evaluation results for both the mathematical and chemical born-digital parsers. For chemical diagrams, we use two evaluation representations: SMILES and labeled directed graphs. A key contribution of this work is the direct comparison of molecular structure graphs, which enables detailed detection of structural differences that are often missed by traditional SMILES-based evaluations.

3.1 SymbolScraper: Symbol Extraction from PDF

`SymbolScraper` is a tool for extracting characters and shapes from vectorized drawing instructions in PDF files, ignoring embedded images [113]. This requires identifying and extracting character shapes (*glyphs*) embedded in font profiles, as well as instructions for other graphics such as lines and polygons. Unless formula images are embedded in a PDF file (e.g., as a .png), born-digital PDF documents provide encoded symbols directly, removing the need for character recognition [12]. In PDF documents, character locations are represented by their position on writing lines and in 'words,' along with their character codes and font parameters (e.g., size).

For the math parser, we simply extract all character locations and labels from the math formula regions, detected by YOLOv8. These symbol-level details from born-digital PDFs are directly utilized; however, in cases where symbols are absent or incomplete, characters are recognized from connected components (CCs) in images using a visual parser called QD-GGA [72].

For the chemical parser, the extraction process is more complex, as it involves identifying and tokenizing additional graphical elements such as polygons, curves, lines, and rectangles present in chemical diagrams. PDF graphics are defined primarily by instructions for lines, rectangles, and Bezier curves. We use these as graphical primitives along with their parameters such as (x,y) points, line widths, whether objects are filled, etc. Graphical primitives are converted to *line strings* (polylines)[4], each of which is a sequence of straight line segments. We approximate Bezier curves in PDF as straight line segments, using a parameter to limit the maximum distance that a point on the original curve can deviate from the approximated line segments, in points (i.e., 1/72 of an inch).

For chemical diagram parsing, a small number of rules and additional parameters are used to extract the final input tokens (parameters shown in Table 3.1). Some straight lines are drawn as filled polygons, which are approximated by a line if the two longest lines cover more than a percentage of the polygon perimeter and have their angles within a small tolerance. Solid wedges (trapezoids) are identified in polygons based on the ratio of long:short side lengths. Positive charges are sometimes drawn with two overlapping lines tested for perpendicularity within an angular tolerance.

The final input tokens produced by SymbolScraper for the born-digital parser are bounding boxes, polygons, or polylines. Each have associated parameters, types, and labels.[5]

[4] Java Topology Suite: `https://locationtech.github.io/jts/`

[5] represented using the Python `Shapely` library

3.2 MST-based Math Formula Parsing

Parsing mathematical formulas from PDFs requires building a graph representation of the symbols and their spatial relationships. In our approach, we leverage the high-quality information available in born-digital PDFs by using SymbolScraper to extract precise symbol locations and labels directly from drawing commands. This eliminates the need for OCR and ensures accurate symbol extraction wherever PDF information is available. However, not all PDFs contain complete symbol data or maintain drawing instructions, necessitating the use of a neural network-based method as a fallback. For these cases, we employ QD-GGA [72], which is a convolutional neural network based parser, which segments connected components (CCs) and recognizes symbols from pixel-based data. These image-based models are discussed in detail in Chapter 4.

This parsing approach uses a Maximum Spanning Tree (MST) for determining formula structure. Specifically, we construct a line-of-sight (LOS) graph over symbol bounding boxes, which is then merged into a symbol-level LOS graph. The MST is extracted from this symbol-level LOS graph by maximizing the relationship probability distributions obtained from the CNN using Edmond's arborescence algorithm [38].

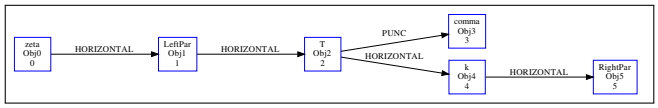
3.2.1 Identifying Extracted Symbols in Formula Regions

The first step in parsing is identifying and associating extracted symbols with their corresponding formula regions. We detect overlapping areas between the formula regions (detected using YOLOv8) and the bounding boxes of symbols extracted by SymbolScraper. Symbols outside the detected formula regions are discarded, as illustrated in Figure 3.1d. The combined information—formula regions and their corresponding symbols—is stored in a tab-separated variable (TSV) file in a hierarchical structure. This consolidated data serves as the input for graph-based parsing.

3.2.2 Parsing Formula Structure with SymbolScraper and QD-GGA

When symbol locations and labels are available from SymbolScraper, they are directly used for parsing, bypassing the need for character-level OCR or CC-based segmentation. Figure 3.4 shows an example of formulas parsed using SymbolScraper data. However, for cases where SymbolScraper data is incomplete or missing, QD-GGA [72] supplements the parsing process. QD-GGA uses connected component (CC) extraction and a CNN based network to identify symbols and spatial

Input: image w. extracted chars. (BBs shown)



Output: Symbol Layout Tree (SLT)

```
\(\zeta\left( {{T,}\left. k \right)} \right.\)
```

SLT in LaTeX

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
<mrow>
    <mi xml:id="0:">ζ</mi>
    <mrow>
        <mo xml:id="1:">(</mo>
        <mrow>
            <mrow>
                <mi xml:id="2:">T</mi>
                <mo xml:id="3:">,</mo>
            </mrow>
            <mrow>
                <mi xml:id="4:">k</mi>
                <mo xml:id="5:">)</mo>
            </mrow>
        </mrow>
    </mrow>
</mrow>
</math>
```

SLT in Presentation MathML

Figure 3.4: Parsing a formula image. Formula regions are rendered and have characters extracted when they are provided in the PDF. We produce a Symbol Layout Tree as output, which can be translated to LaTeX and Presentation MathML.

relationships in the formula.

We construct a graph in which the nodes represent extracted symbols or CCs, and the edges capture spatial relationships between these nodes. These edges are initially defined by a line-of-sight (LOS) graph computed over the connected components. In cases where CC-based symbols are segmented into multiple parts, binary 'merge' relationships are defined, and the symbol class is determined based on the highest average confidence score among merged CCs.

3.2.3 Building the Symbol Layout Tree (SLT)

Once the symbols and their relationships are identified, the next step is to construct a hierarchical graph representation, the Symbol Layout Tree (SLT). To build the SLT, we use Edmond's arborescence algorithm [38] to extract a Maximum Spanning Tree (MST) from the weighted relationship class distributions. This ensures that the final formula interpretation reflects the correct structure and spatial arrangement of the symbols. QD-GGA concurrently trains CNN-based features and attention modules for multiple tasks, including symbol classification, edge classification, and segmentation of CCs into symbols. The graph-based attention module allows multiple classification queries for both nodes and edges to be processed in a single feed-forward pass, leading to a fast training and inference.

The final output of the parsing process is an SLT, which encodes both the structure and the symbols

Table 3.1: Parameters for PDF Symbol Parsing Stages

Parameter (Value)		Primitive Graph (MST) 1. Extract Symbols	Primitive Graph (MST) 2. Build MST	3. MST → Visual Graph (a) -ve Charges	3. MST → Visual Graph (b) Restr. MST	3. MST → Visual Graph (c) Tokenization
PDF GRAPHIC PRIMITIVES						
BEZIER_FLATNESS_PTS	(0.25)	✓				
RECT2LINE_LONG_RATIO	(0.85)	✓				
RECT2LINE_ANGLE_TOLERANCE	(5.0)	✓				
ANGLES & PROXIMITY						
ANGLE_TOLERANCE_DEGREES	(3.0)	✓		✓	✓	✓
CLOSE_NONPARALLEL_ALPHA	(1.75)				✓	
CLOSE_CHAR_LINE_ALPHA	(1.5)				✓	
SYMBOLS						
S-WEDGE_LENGTHS_DIFF_RATIO	(0.7)	✓				
NEG-CHARGE_Y_POSITION	(0.5)			✓		
NEG-CHARGE_LENGTH_TOLERANCE	(0.5)			✓		
PRUNING EDGES						
ABS_COS_CHAR_PRUNE	(0.1)		✓			
CHAR_LINE_Z_TOLERANCE	(1.5)		✓			✓
MAX_ALPHA_DIST	(2.0)					✓

of the formula, along with the spatial relationship classifications. This SLT can be further converted into other machine-readable formats, such as Presentation MathML or a LaTeX string, for use in various applications (see Figure 3.4).

In cases where symbol data is incomplete or absent, our method seamlessly integrates QD-GGA as a fallback, ensuring robust formula recognition. Further details on the neural network-based parsing approach (QD-GGA) are provided in Chapter 4, where we explore how attention-based methods improve segmentation and classification accuracy.

3.3 MST-based Molecular Digram Parsing

In this section we present the ChemScraper born-digital parser for recognizing molecular diagrams directly from vectorized PDF images. As seen in Figure 3.5, our born-digital parser has four stages, including extracting graphics commands using an improved SymbolScraper [113], constructing a Minimum Spanning Tree (MST), rewriting the MST as a visual structure graph, and finally rewrit-

Input: Born-Digital PDF Molecule Image

1. **Extract Symbols from PDF**
   Characters and graphical objects (e.g., lines)
2. **Build Minimum Spanning Tree (MST)**
   Connect neighboring lines, shapes, & characters
3. **MST → Visual Graph**
   (a) Detect negative charges (vs. other lines)
   (b) Restructure MST
   (+) add edges: touching lines (e.g., in rings), adjacent parallel lines and char/line pairs
   (-) delete edges: 'floating' objects
   (c) **Tokenization**
   · Neighboring characters → name nodes
   · Neighboring parallel lines → bond nodes
4. **Visual Graph → Molecular Graph**
   *No tunable parameters
   (a) Convert line intersections into carbons
   (b) Replace bond nodes by edges
   (c) Annotate names with subgraphs (e.g., SO_2)
   (d) Generate CDXML

Output: Editable molecular diagram (CDXML)

Figure 3.5: Molecule Parsing from PDF Symbols. Symbol information is transformed into an MST (Figure 3.2(b)), a visual structure graph (Figure 3.2(c)), a *tokenized* visual graph (Figure 3.2(c), and finally a molecular structure graph (Figure 3.2(d))

ing the visual graph into a molecular structure graph. The final molecular graph replaces line intersections by carbon atoms, and all bond tokens/nodes (e.g., single, double, triple, solid/hashed wedge) are replaced by edges.

This is a compiler-like recognition architecture, with some similarities to the DRACULAE mathematical formula recognition system [152]. Using a compiler-based architecture provides a helpful separation of concerns that allows changes to be implemented and tested across smaller modules.

We provide an overview of the outputs and processing for stages shown in Figure 3.5. Each stage is

then described in more detail in the remainder of this section. The full parsing process has an asymptotic run-time complexity of $O(n^2 \log n)$ for n nodes in the input graph (PDF character/graphics primitives), reflecting the cost of MST construction.

Stages 1 & 2: Primitive Graph (MST). SymbolScraper recovers primitive symbols from PDF, for which neighboring objects are identified using an MST. Because molecule diagrams represent connections between atoms/groups using line intersections and line/character proximity, MSTs capture many valid connections. However MSTs prune cycles, some primitives must be merged, and some diagrams contain multiple molecules (e.g., parallel lines in bonds and floating ions).

Stage 3: (Tokenized) Visual Graph. To capture structure missing in the primitive MST, the MST is transformed to provide a two-dimensional syntactic analysis for the visible primitives. This is done by first adding/removing edges to correct MST structure producing a *visual structure graph* (Figure 3.2(c)), followed by grouping characters and lines into names and bond types (i.e., tokens) producing a *tokenized visual structure graph* (Figure 3.2(d)).

Stage 4: Molecular Graph. The final stage is semantic analysis: visual syntax is mapped to represented information/structure, including elements not visible in the diagram. This includes identifying hidden carbon atoms at line intersections, and structures represented only by name. In our system, names are mapped to molecular subgraphs using a dictionary. In Figure 3.2(e), NO_2 will be replaced by a subgraph with one nitrogen and two oxygen atoms connected to a hidden carbon.

This MST-based approach differs from that used in math parsing: instead of being the final structure, the MST here serves as an initial constraint that is iteratively refined through edge addition/removal to form the final chemical graph, allowing cycles where necessary for accurately representing molecular structures. Our semantic analyzer, designed for this parsing pipeline, can also be applied to other visual parsers generating compatible visual graph outputs, as demonstrated with the visual parser discussed later in Chapter 4.

3.3.1 Minimum Spanning Tree (MST)

MSTs are widely used for constraints and optimization tasks involving point sets and other geometric object collections in continuous space (i.e., $\mathbb{R}^n$), including agglomerative clustering. For graphics recognition, MSTs have been used to constrain symbol and spatial relationship types when recognizing handwritten math formulas, e.g., by Matsakis [76] and Eto and Suzuki [40].

As can be seen in Figure 3.2, chemical diagrams are even better suited to MST-based selection of spatial relationships than math formulas. The visual structure of math formulas may have as many as eight spatial relationship types, while molecule diagrams contain only one spatial relationship (`connected`). Symbols in formulas may be related at a distance, while connections in molecular diagrams are between neighboring symbols. Lines or other graphical objects that need to be combined into symbols (e.g., two parallel lines in a double bond) are also neighboring objects.

We construct an MST to connect graphical primitives with their nearest neighbor in a chemical diagram, breaking ties arbitrarily when two or more neighbors are equidistant. A complete undirected graph over all input PDF primitive pairs is generated first, with edges weighted by distance. By default, edge weights are the distance between the closest points on two objects; however, for line pairs we use their end-points to capture connection distances. This also prevents overlapping lines from having distance 0.

Invalid character connections are prevented by setting distances in our weighted adjacency matrix to ∞ when: (1) The absolute value of the cosine for the angle between characters falls between [0.1, 0.9], e.g., between [25.8, 84.3]°. This prevents (illegal) superscript or subscript character connections. (2) A line-character distance is more than 1.5 standard deviations from the mean line-character distance in the diagram. Pruning parameters are shown in Table 3.1.

We use Kruskal's algorithm to extract an MST with $n-1$ edges for n primitives, such that the sum of edge distances is minimal in the pruned adjacency matrix. An example MST over input graphics primitives is shown in Figure 3.2(b).

3.3.2 MST → Visual Structure Graph

While an MST over PDF graphical primitives includes many connections needed to recognize molecular structure, connections often need to be added or removed. For example, an MST cannot contain cycles, and so we need to insert edges when three or more lines intersect. These and other changes are needed to produce the final graph capturing the visual syntax of a molecular diagram, e.g., as seen in Figure 3.2(d). The steps used for this transformation are presented below; parameters are shown in Table 3.1.

Negative Charges. We first distinguish negative charges from other lines. Lines are considered negative charges if they are: (1) roughly horizontal (0°), (2) no longer than a fraction of the average line length in the diagram, and (3) right adjacent to a character, with the line's vertical center in

the upper half of the character's bounding box.

Restructure MST. Next we correct connections for 'floating' bond lines such as the double bonds in Figure 3.2. These floating lines may not connect with their corresponding parallel line in the MST when another line's endpoint is closer. We consider creating an edge between a candidate floating line with degree 1 (one edge) in the MST with another nearby overlapping parallel line if it is within the five nearest neighbors of the line, and the average endpoint distances between the two lines is smaller than for the current neighbor. If so, the line is disconnected from its current neighbor and connected to the closer parallel line.

We then use distance-based clustering to add and remove connections based on MST distances.

1. *Line Intersections.* Add missing non-parallel line intersections (e.g., for rings and multi-line intersections) where the lines' endpoints are within a ratio of the maximum distance between connected non-parallel lines.
2. *Character-Line Connections.* Filter MST char-line connection distances via Z-scores (i.e., standard deviations from the mean) before estimating the maximum char-line connection distance. Add all char-line edges within a ratio of this maximum distance.
3. *Split Floating Structures.* Prune edges with a distance larger than a ratio of a maximum distance. The connection type used to determine the maximum distance is selected in the following in order, based on first available distance type in the MST: (1) char-line distances, (2) parallel line distances, or (3) non-parallel line distances.

Tokenization. There are two steps for merging lines into bonds and characters into atom and group names: (1) merging adjacent characters and parallel lines, and (2) labeling bond types.

Merge Characters and Parallel Lines. Characters connected by edges are merged into text tokens, using the location of the nearest character as the connection point for a bond, if present (see Figure 3.2(d)). Double bonds, triple bonds, and hashed wedge bonds are represented by adjacent parallel lines. Hashed and solid wedge bonds have a shorter side that begins the bond and a longer side that ends the bond, indicating the bond direction. Solid wedge bonds are trapezoids, while hashed wedge bonds are drawn as parallel lines of increasing length. All neighboring parallel line groups in the MST are merged, and annotated by the number of lines they contain. For example, in Figure 3.2(d), three pairs of parallel lines representing double bonds will each be merged and annotated with '2'.

Label Bonds in Line Groups/Wedges. Annotated line groups can then be labeled as *single*, *double*, or *hashed wedge* bonds by the number of lines they contain (i.e., 1, 2, or >3). Three parallel lines are a special case: both triple bonds and hashed wedge bonds may contain 3 parallel lines. We distinguish these by sorting the 3 lines topologically (i.e., top-down, left-to-right), and then determine whether these lines uniformly increase or decrease in size within the sorted list.

For wedge bonds, we need to identify new endpoints on the longest and shortest sides (for solid) or longest and shortest lines (for hashed) and restructure the final visual structure graph accordingly. Bond endpoints are important in the semantic analysis step, which we describe next.

3.3.3 Visual → Molecular Structure

In the final stage of the born-digital parser, visual structure is converted to molecular structure, and chemical information not directly visible in the diagram is added to produce a chemical graph. The chemical graph is then represented in a CDXML file capturing *both* visual and chemical structure. Note that this stage uses a deterministic process that involves no tunable parameters.

We first need to define explicit intersection points where line endpoints meet. These intersection points are defined by the midpoint between adjacent endpoints for connected lines in the visual structure graph. 'Hidden' carbon atoms are then inserted as nodes at bond line intersections, and at line endpoints without a neighbor. Nodes for bonds in the tokenized visual structure graph are removed, and replaced by edges labeled with the same bond type (see Figure 3.2(d) and (e)).

CDXML Generation. CDXML is a file format representing molecules and reactions along with related text on a canvas or series of pages. For molecular data, both chemical structure and the appearance of molecules on a 2D canvas are encoded in CDXML files. The format was created for the ChemDraw chemical diagram editor.

In CDXML tags define molecules, nodes (e.g., atoms, named groups), and bond connections in the diagram, along with annotations for node positions and appearance. We encode the locations of nodes on their associated page, so that the appearance and location of recognized molecules match the original document. Positions are also helpful with accurate conversion to other chemical formats (e.g., SMILES), and to capture spatial information in the chemical structure (e.g., for wedge bonds).

Annotate Names with Subgraphs: Molecules are often represented more compactly using chemical formulas or other names for substructures. For example, Figure 3.2 shows an abbreviation node NO_2, a nitro group with an external connection available. We use a manually compiled dictionary

of 612 common abbreviations with their associated subgraphs collected from the RDKit Python library[6], ChemDraw, and our own work. For the abbreviation NO_2, we insert the full structure ($* \rightarrow N_1, N_1 \rightarrow O_1, N_1 \rightarrow O_2$) into the CDXML as a nested molecule 'fragment.' $*$ represents where the structure can be connected to other structures; O_1 and O_2 represents two oxygen atoms connected to the nitrogen N_1 through a single and double bond respectively.

3.4 Generating Training Data for Visual Parser

In designing ChemScraper, we noticed that authors often copy molecular diagrams directly into their documents as raster images, which become embedded in PDFs. To create parsers for raster images with easily interpreted results, we require *explicit* correspondences between image regions and molecular symbols in generated visual structure graphs. Unfortunately, there is a shortage of training data with direct annotations of raster images. In addition to fast and accurate recognition, this was the second key motivator for creating our born-digital parser.

While one can create large datasets from SMILES using their rendered raster images, the correspondence between image regions and portions of SMILES strings is absent in such datasets. One can also generate molecular diagram images from MOL files, which include explicit molecular structure (e.g., atoms and their connection by bonds), along with optional 3d spatial positions. However, MOL files were not designed to describe image regions for characters, bonds, or other visual primitives in an image. For example, MOLs identify spatial locations of atom groups such as CH_3, but do not the locations for its constituent `H` and `3` in an image.

A new data generation technique is required. First, we sought a stable visual primitive in pixel-based (raster) molecule images that would avoid merging symbols, and found that we could extract a type of line primitive reliably for this purpose (see Figure 4.5(b)). Given the born-digital parse results for a molecule in PDF, we extract these line primitives from the rasterized PNG for the molecule, and align them with the PDF primitives based on maximum overlap.

The born-digital visual graphs annotated with line primitives can then be used for training models using the same line primitives as input. For these parsers, the visual primitive extraction replaces the first step of the born-digital parsing pipeline seen in Figure 3.3, where rather than extract characters and lines directly, we may also extract image regions that over-segment (i.e., split) lines and characters.

[6] `https://www.rdkit.org`

3.4.1 Visual Primitives (Lines)

From a raster image (PNG) for a PDF molecule rendered by the Indigo chemoinformatics toolkit,[7] we extract connected component (CC) contours, and convert these to polygons using a simplification algorithm (provided by `Shapely`). These polygons are transformed into a set of skeletal lines using pairs of adjacent parallel lines on the contour boundary. Each pair of parallel lines is replaced by their medial axis (i.e., line between the middle of the parallel lines' endpoints).[8] After the medial axis lines have been identified, pixels in CCs are segmented by assignment to the nearest axis line using a distance transform.

The resulting 'visual' line primitives can be seen in Figure 4.5(b). Some CC shapes such as curved lines and closed curves are unaltered by the process. The 2 is unsegmented because after identifying all skeletal lines for CCs in a molecule, to avoid segmenting small CCs, we test whether the average skeletal line length in a CC is less than the average for all skeletal lines. If this average length is smaller than the global average, we do not segment the CC. We also remove skeletal lines within CCs that are smaller than the global average skeletal line length, which avoids over-segmenting lines at dense intersections (e.g., at the connection point between two single bonds and a double bond). We split a long line in a triple or double bond by projecting the floating line onto it, and then testing if the overlap ratio r for the longer line is in the interval of one third to one half, with a margin of 10% ($\frac{1}{3} - \frac{1}{10} \leq r \leq \frac{1}{2} + \frac{1}{10}$).

For illustration, here we have manually broken the N into three parts; in practice, both characters and lines may be over-segmented. In Figure 4.5(b) there are 15 visual primitives, versus 13 graphical primitives for the original PDF in Figs. 3.2(a) and (b). 10 primitives are straight bond lines, and 5 primitives are for the characters in NO_2.

3.4.2 Visual Graph Generation

We now annotate raster images using our visual primitives and visual graphs before tokenization (see 3.2(c)) from our born-digital parser. We use Indigo to render PDFs from SMILES rather than PNG images as done in previous methods (e.g., MolScribe [100]). The born-digital parser is then run on the PDF images, and where the recognized SMILES and original SMILES match (i.e., the result is correct), we use the resulting visual graph as our preliminary ground truth data (e.g., see Figure 3.2(c)).

[7] `https://github.com/epam/Indigo`

[8] parameters in Table 3.1 constrain angles and min. overlap

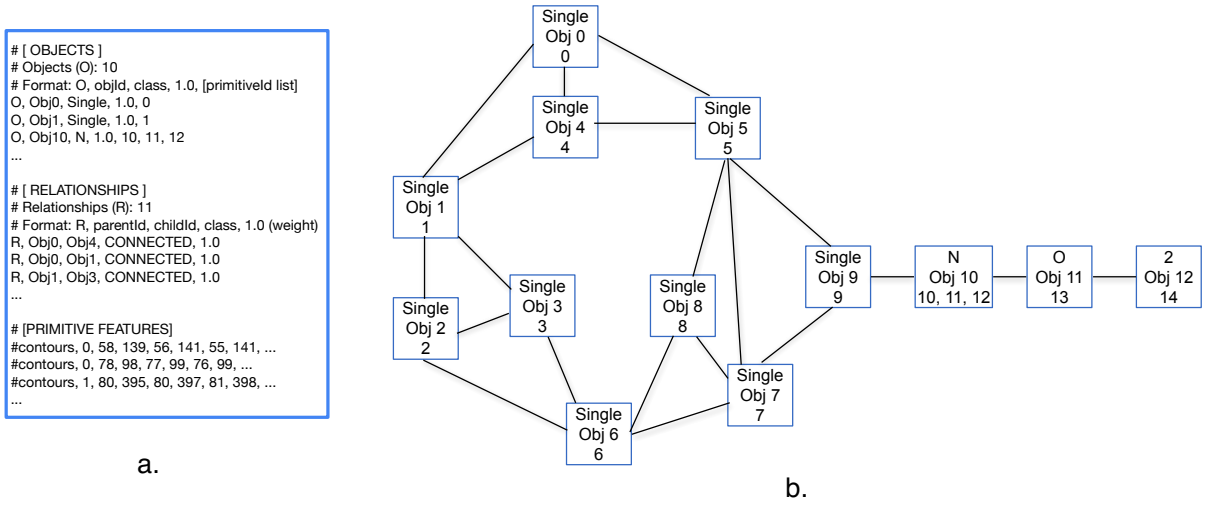


Figure 3.6: Ground Truth Visual Graph Generated for Figure 3.2(c). (a) Label graph file with Objects (`O`), Relationships (`R`) and Visual primitives with contour points (`#contours`). (b) Visualization showing primitive identifiers, node labels, and edges (all edges labeled as `CONNECTED`. Objects for single bond contain one line primitive each, while the character `N` contains three line primitives. A second file for the PNG is created using 13 PDF primitives (vs. 15 visual line primitives shown here).

We next assign visual line primitives to PDF graphical primitives in the born-digital visual graph. PDF images are converted to 256 DPI PNG images, and we extract visual line primitives as described above. The assignment of visual primitives to PDF primitives/symbols is determined by maximum overlap. In Figure 4.5(c), 1 line primitive is attached to each line node, 3 line primitives are attached to N, and one primitive is attached to each of the O and 2. Finally, we validate bonds between atoms against a MOL connection table generated from SMILES using Indigo.

To store visual graphs, we create label graph (Lg) files [84, 85] for both PDF primitives and visual line primitives. An example is shown in Figure 3.6(a). Primitives are represented by numeric identifiers and image contours, while typed objects are comprised of one or more primitives (e.g., `Single` bond: one line, character `N`: three lines).

A label graph file defines structure over declared primitives, using primitive groups (*objects*) and their relationships. In our label graph files, only `CONNECTED` relationships are explicitly defined, however `MERGE` relationships are defined implicitly between all primitive pairs in an object. In Figure 3.6 `MERGE` edges exist between primitives 10, 11, and 12 for `N` (`Obj10`), and the connection between this character and the `Single` bond `Obj9` is represented by `CONNECTED` edges for (9,10), (9,

11) and (9,12). Similarly, all primitives in an object share a label (e.g., for `Obj10`, primitives 10, 11, and 12 are labeled `N`).

3.5 Evaluation and Results

This section presents the evaluation metrics and experimental results for the parsing methods developed for both mathematical formulas and chemical diagrams. The goal of this evaluation is to assess the accuracy, robustness, and overall effectiveness of the born-digital parsers across both domains, using the graph-based and string-based metrics introduced in Chapter 2. For mathematical formulas, we primarily rely on raster image evaluations due to the lack of standardized datasets for evaluating formulas extracted directly from PDFs. Here, the parsing pipeline leverages SymbolScraper to extract symbol locations and labels from PDFs wherever possible, with QD-GGA [72] used as supplementary models when PDF information is missing. The evaluation covers the accurate reconstruction of Symbol Layout Trees (SLTs). Similarly, for chemical diagrams, we measure the parser's ability to extract graphical primitives and convert them into molecular graphs. The results are presented separately for math and chemistry, with a detailed analysis of system performance, limitations, and potential error sources.

3.5.1 Evaluation of Math Formula Recognition

We do not have results specifically for the born-digital math parser, as there are no standard datasets available for benchmarking formulas extracted directly from PDFs. Therefore, we present our evaluation results based on raster images, using QD-GGA [72], the neural network-based parser employed in cases where PDF information was unavailable in our math parsing pipeline.

Dataset. We evaluated the parser on the InftyMCCDB-2 dataset,[9] a modified version of InftyCDB-2 [118]. The test set consists of 6830 mathematical expressions derived from scanned article pages. The parser achieved a structure recognition rate of 92.56% and an expression recognition rate (Structure + Classification) of 85.94%.

Implementation/Systems. SymbolScraper is built in Java using Apache's `PDFBox` and the `Java Topology Suite`. The experiments were conducted on a desktop system with a hard drive (HDD), 32GB RAM, an Nvidia GTX 1080 GPU, and an Intel(R) Core(TM) i7-9700KF processor. The

[9] `https://zenodo.org/record/3483048#.XaCwmOdKjVo`

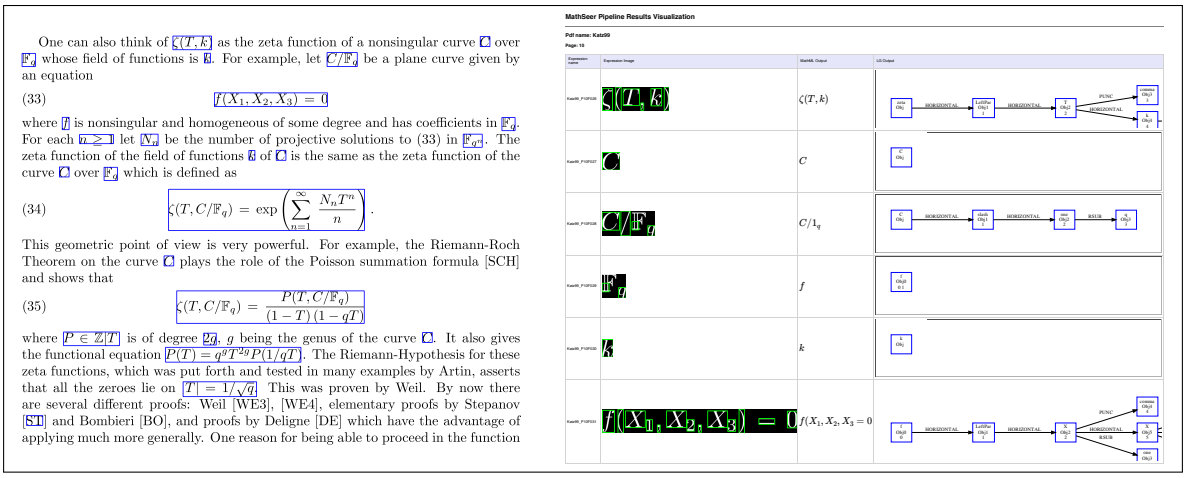


Figure 3.7: HTML visualization for formulas extracted from a sample PDF page with detected formula locations (left), and a table (right) showing extracted formulas and recognition results as rendered MathML and SLT graphs.

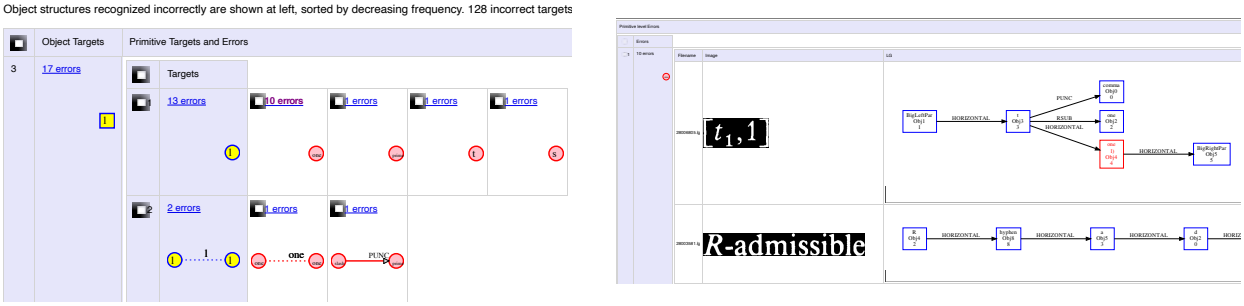
total processing time for the 6830 formula images was 26 minutes and 25 seconds, averaging 232 milliseconds per formula.

An important contribution of this work is new tools for visualizing recognition results and structure recognition errors, which are essential for efficient analysis and error diagnosis during system development and tuning. We have created these in the hopes of helping both ourselves and others working in formula extraction and other graphics extraction domains.

Recognition Result Visualization (HTML + PDF). We have created a tool to produce a convenient HTML-based visualization of the detection and recognition results, with inputs and outputs of the pipeline for each PDF document page. For each PDF page, we summarise the results in the form of a table, which contains the document page image with the detected expressions at the left and a scrollable (horizontally and vertically) sub-table with each row showing the expression image, the corresponding MathML rendered output, and the SLT graphs (see Figure 3.7).

We created HTMLs for easy evaluation of the entire pipeline, to identify component(s) (symbol extraction, expression detection, or structure recognition) producing errors on each page. This makes it easier to diagnose the causes of errors, identify the strengths and weaknesses of different components, and improve system designs.

Improved Error Visualization. The LgEval library [84, 85] has been used for evaluating formula structure recognition both for recognition over given input primitives (e.g., strokes) and for output



(a) (b)

Figure 3.8: Error analysis (errors shown in red). (a) Main error table organized by decreasing frequency of errors. (b) Specific instances where 'l' is misclassified as 'one,' seen after clicking on the '10 errors' link in the main table.

representations without grounding in input primitives [73] . We extend LgEval's error summary visualization tool, `confHist`, to allow viewing all instances of specific errors through HTML links. These links take the user directly to a list of formulas containing a specific error. The tool generates a table showing all error types for ground truth SLT subtrees of a given size, arranged in rows and sorted by frequency of error, as shown in Figure 3.8a (for single symbols). Each row contains a sub-table with all primitive level target and error graphs, with errors shown in red. Errors include missing relationships and nodes, segmentation errors, and symbol and relationship classification errors - in other words, all classification, segmentation, and relationship errors.

New links in the error entries of the HTML table open HTML pages, containing all formulas sharing a specific error along with additional details arranged in a table. This includes all expression images containing a specific error along with their SLT graph showing errors highlighted in red Figure 3.8b. The new tool helps to easily identify frequent recognition errors in the contexts where they occur. For example, as seen in Figure 3.8b, we can view all expression images in which 'l' is misclassified as 'one' by clicking the error entry link (10 errors) in Figure 3.8a and locate the incorrect symbol(s) using the SLT graphs.

Both visualization tools load very quickly, as the SLT graphs are represented in small PDF files that may be scaled using HTML/javascript widgets.

3.5.2 Evaluation of Chemical Diagram Recognition

We next evaluate the accuracy of our parsers. It is important to remember that the ChemScraper born-digital parser utilizes PDF information for characters, lines, and other graphical objects that parsers working from raster (pixel) images do not. Our analysis includes a graph-based analysis of recognition errors at the level of molecule structure present that provides information missing in standard SMILES-based evaluation methods.

Datasets. For tuning born-digital parser parameters and generating visual parser training data, we use 5000 molecules (46 unique SMILES characters) extracted from PubChem[10] prepared by the MolScribe team [100]. For benchmarking, we use three datasets: (1) the USPTO synthetic dataset with 5,179 PNG images generated by the Indigo toolkit from SMILES strings (37 unique SMILES characters) [101], (2) UoB (5,740 molecule PNG images + SMILES: 33 unique characters [107]), and (3) CLEF (992 molecule PNG images + SMILES: 71 unique characters [99]).

The born-digital parser is run on Indigo-rendered PDFs from SMILES ground truth, including for the UoB and CLEF datasets. For the USPTO synthetic set, the rendered PNG and PDF images are essentially identical, but this is not true for the CLEF and UoB data sets where scanned images of molecules were annotated with SMILES; in this case rendering the SMILES using Indigo may produce images in different styles, fonts, and orientations than the scanned molecule images.

Implementation/Systems. ChemScraper born-digital parser is implemented in Python using the `Shapely` (2d geometry), `networkx` (graphs), `numpy`, and `mr4mp` (map-reduce) libraries. The ChemScraper born-digial and visual parsing pipelines are Python-based, along with the visual line primitive extractor.

Born digital parsing runs were made on a Ubuntu 20.04 server, with a Intel(R) Xeon(R) CPU E5-2667 v4 (3.20 GHz) and 512 GB RAM. Experiments for the visual parser were run on another Ubuntu 20.04 server with hard drives (HDD), an A40 (48GB) GPU, a 64-core Xeon Gold 6326 (2.9 GHz), and 256 GB RAM.

SMILES-Based Evaluation

ChemScraper-generated CDXMLs are first translated to SMILES using ChemAxon's `molconvert` tool. After this, we canonicalize both CDXML and benchmark SMILES to remove differences in

[10] `https://pubchem.ncbi.nlm.nih.gov`

their atom order, which can vary for the same molecule. SMILES canonicalization is performed using the RDKit library via the function `CanonSmiles()`, with `ignore_chiral=False`.

Parameter Tuning and Rendering. Each molecule in our 5,000 PubChem molecules for parameter fitting was rendered with Indigo with 3 randomly selected parameters. The rendering parameters are described below. For benchmarking the born-digital parser, we use the Indigo default rendering parameters. This is done to insure PDF molecules for the born-digital parser have the same appearance as PNG images in the USPTO dataset, which is our primary collection for benchmarking.

The final parameter values seen earlier in Table 3.1 are obtained using grid search, with the exception of the PDF GRAPHICS PRIMITIVES group belonging to `SymbolScraper`. To keep the tuning process manageable, we divided the grid search into 3 stages, one per group in the order given in Table 3.1. Initial default values were identified. After each parameter group's grid search was complete, learned values replaced the default values. Value ranges and defaults are shown in Table 3.2.

We also tested the effect of the MST pruning parameters discussed in Section 3.3.1: removing them harms accuracy. For the USPTO dataset removing the absolute cosine angle threshold for characters produces 93.72% SMILES matches, removing the threshold for line-character distances produces 97.06% SMILES, matches and removing both produces 93.20% matches. Including the pruning parameters produces 98.16% exact SMILES matches.

Benchmarking Born-Digital Parser. Table 3.3 compares ChemScraper and existing molecule parsing models. For the USPTO dataset, we see that the born-digital parser obtains the highest rates. Note that the 'rendering failure' for USPTO applies to all systems, because the SMILES for these 15 molecules are missing in the collection itself. Given this, the born-digital parser working from PDFs outperforms the neural models working from raster images by nearly 1%, and rule-based system working from raster images by roughly 3%. The strong performance of the born-digital parser is because of the additional information is available from PDF instructions, and the robust design of the born-digital parser.

The model also obtains competitive rates for CLEF and UoB, but note that this is for Indigo-rendered SMILES, and not the provided PNGs because PDF images are not provided in these collections.

In terms of execution time, running the born-digital parser on the USPTO-Indigo dataset (5,719 molecules) with a single process took 28.01 mins (293.39 ms/formula), i.e., 3.4 molecules/sec, with

Table 3.2: Grid Search Parameters. Values tested are shown, with default values in bold.

1. ANGLES & PROXIMITY	
ANGLE_TOLERANCE_DEGREES	{1, 3, **5**, 10, 15}
CLOSE_NONPARALLEL_ALPHA	{1, 1.25, 1.5, **1.75**, 2.0}
CLOSE_CHAR_LINE_ALPHA	{1, 1.25, **1.5**, 1.75, 2.0}
2. SYMBOLS	
S-WEDGE_LENGTHS_DIFF_RATIO	{0.70, 0.85, **0.90**, 0.95}
NEG-CHARGE_Y_POSITION	{0, **0.25**, 0.5}
NEG-CHARGE_LENGTH_TOLERANCE	{0.33, **0.5**, 0.66}
3. PRUNING EDGES	
ABS_COS_CHAR_PRUNE	{0.10, **0.15**, 0.20}
CHAR_LINE_Z_TOLERANCE	{1.0, **1.5**, 2.0}
MAX_ALPHA_DIST	{2.0, **2.5**, 3.0}

Table 3.3: Molecular Structure Recognition Benchmarks. Percentages of generated SMILES matching ground truth are shown. For USPTO both PNG and PDF images are rendered using Indigo, but rendered SMILES PDFs may differ from scanned PNGs for CLEF and UoB (indicated by italics).

		Synthetic Image	*Scanned Image	
Models		USPTO (5719)	CLEF-2012 (992)	UoB (5740)
Rule-based	MolVec 0.9.7	95.40	83.80	80.60
	OSRA 2.1	95.00	84.60	78.50
	Imago 2.0	-	68.20	63.90
Neural Network	Img2Mol	58.90	48.84	78.18
	DECIMER	69.60	62.70	88.20
Graph Outputs	OCMR	-	65.10	85.50
	SwinOCSR	74.00	30.00	44.90
	Image2Graph	-	51.70	82.90
	MolScribe	**97.50**	88.90	87.90
	MolGrapher	-	**90.50**	**94.90**
ChemScraper	**Born-Digital Parser (PDF input)**			
	(PDF rendering errors)	(15) **98.16**	*(71) 89.32*	*(0) 94.41*
	*Skipping rendering errors	**98.42**	*96.20*	*94.41*
	Visual Parser (PNG input)	85.02	-	-

a peak CPU memory use of 230 MB. With multiple processes (32) the total time is reduced to 1.81 mins (19.04 ms/formula), i.e., 52.5 molecules/sec. Performance benchmarks from Rajan et al. [101] show that on a Linux workstation with Ubuntu 20.04 LTS, two Intel Xeon Silver 4114 CPUs and 64 GB of RAM, processing the USPTO-Indigo dataset took 28.65 minutes for MolVec 0.9.7, and 145.04 minutes for OSRA 2.1. Thus, on comparable systems, our born-digital parser operates at similar or faster speeds compared to other rule-based methods.

Rendering: Sensitivity Analysis. To check the robustness of the born-digital parser, we used the rendering parameters of Indigo to perform a sensitivity analysis. We tested three rendering parameters visualized in Figure 3.9. Parameters/values considered are:

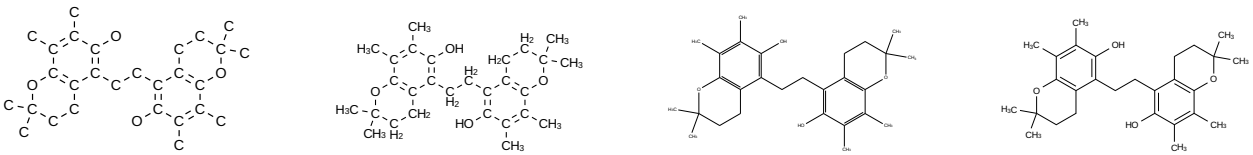
1. `relative-thickness`: Boldness of graphic and text objects. Values considered: {0.5, 1, 1.5}. The default is 1.
2. `render-implicit-hydrogens-visible`: Whether to show implicit hydrogens. Default is True.
3. `render-label-mode`: Which atom labels to show: {hetero, terminal-hetero, all}. *all* shows all atoms. There is a *none* option we omit because it leads to ambiguous molecules. Default is *terminal-hetero*.

This produces 18 parameter combinations for rendering. We evaluated our parser with each of them for the USPTO Indigo dataset, using SMILES matches and inverse normalized levinshtein distances for evaluation.

Figure 3.10 shows how different atom labelings affect performance of the parser. Including all atom labels slightly hurts performance, in part because the more dense a molecule becomes, the more probable it is for the parser to connect atoms incorrectly. Figure 3.11 then shows the effect of rendering with different thicknesses. Lower thicknesses produce stronger results, again because this decreases the density of the molecule. As seen in Figure 3.9, lower thickness increases the distance between unconnected objects.

Figure 3.12 compares performance when rendering molecules with or without implicit hydrogens. The difference between the conditions is minimal, with 14 fewer exact matches (roughly 0.06%) than when showing implicit hydrogens. This difference is due to merging errors of different groups that are close, similar to the crowding of Figure 3.9b.

Overall, the born-digital parser is quite robust to these changes in rendering parameters. This robustness was achieved by gradually increasing the reliance of the born-digital parser on graph



(a) (`all`, `F`, `1.5`) (b) (`all`, `T`, `1.5`) (c) (`term-h`, `T`, `0.5`) (d) (`term-h`, `T`, `1.0`)

Figure 3.9: Rendering a molecule with different parameters (Indigo toolkit). Each of (a)-(d) indicate the label mode, whether implicit hydrogens are shown, and the relative thickness. Parameters in (d) are the defaults. The born-digital parser recognizes all four versions correctly.

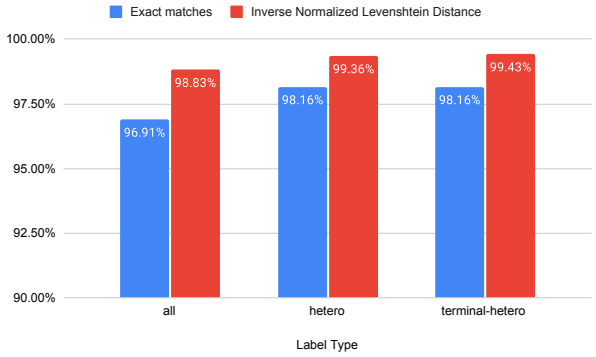


Figure 3.10: Sensitivity of Born-Digital Parser to Label Rendering Parameter. SMILES-based evaluation is used. Other parameters have default values, with `render-implicit-hydrogens-visible` as True and `render-relative-thickness` to 1.

Table 3.4: Born-Digital Parser Label Graph Metrics for Different Rendering Parameters (5719 molecules). Shown are F_1 measures for symbol labels, correct labels, and complete graphs.

	Rendering Parameters			Correct node	Correct edge	Molecules	
Render	`label_ mode`	`implicit_ hydrogens_ visible`	`relative_ thickness`	(labels) F_1	(labels) F_1	Struct.	+Class
Default	terminal-hetero	true	1	99.96	99.84	98.49	97.62
Hardest	all	true	1.5	99.65	99.01	81.89	81.12

properties while reducing the number of parameters used; additional reductions in parameters are likely possible.

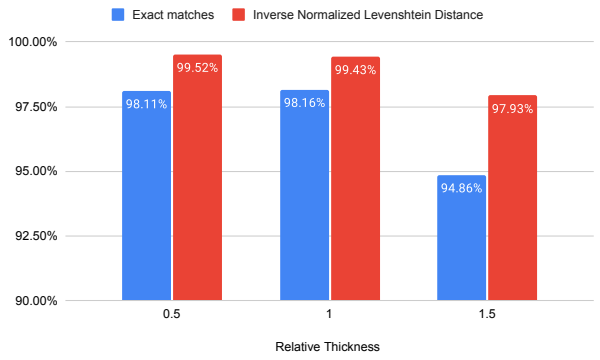


Figure 3.11: Sensitivity of Born-Digital Parser to Thickness Rendering Parameter. Higher thickness reduces accuracy. Other parameters: `render-implicit-hydrogens-visible` is True, `render-label-mode` is terminal-hetero.

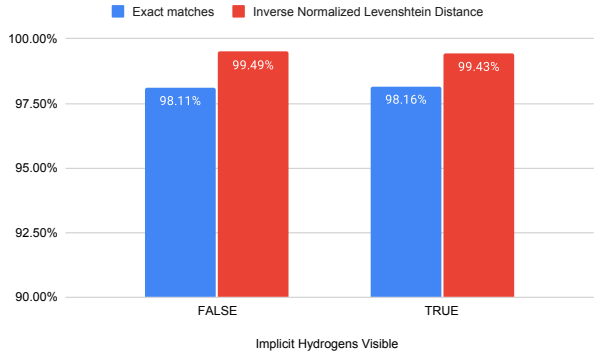


Figure 3.12: Sensitivty of Born-Digital Parser to Showing Implicit Hydrogens. Other parameters: `render-label-mode` is terminal-hetero and `render-relative-thickness` is 1.

Graph-Based Evaluation

Example molecular structure graphs are shown in Figs. 3.2(e) and 4.5(e), which are equivalent.[11] For the ChemScraper parsers, molecular structure graphs produced using born-digital primitives (see Figure 3.2(b)) contain polygons representing the image locations for hidden carbons and atom/group labels. We use these graphs directly for evaluation.

As outlined in Chapter 2, graph-based comparisons are facilitated when both the ground truth and output graphs share the same set of nodes with known input locations. These graphs can be analyzed

[11] **Note:** The graphs are mostly undirected, but wedge bonds going 'in'/'out' of a page require directed graphs.

through their adjacency matrices, where any discrepancies in cell entries indicate recognition errors. By referencing node locations, these errors can be precisely pinpointed within the corresponding input image. Furthermore, even when the grouping of nodes (segmentation) varies between the graphs—whether due to over-segmentation or missing nodes—such differences can still be detected through this bottom-up comparison approach [157].

Here, we take a slightly different approach. Rather than graphs sharing nodes, corresponding ground truth and output nodes in molecular structure graphs are aligned (i.e., assigned the same identifier) based on spatial overlap in a PDF image. After this alignment, we apply the same adjacency matrix-based evaluation metrics and tools used for CROHME.

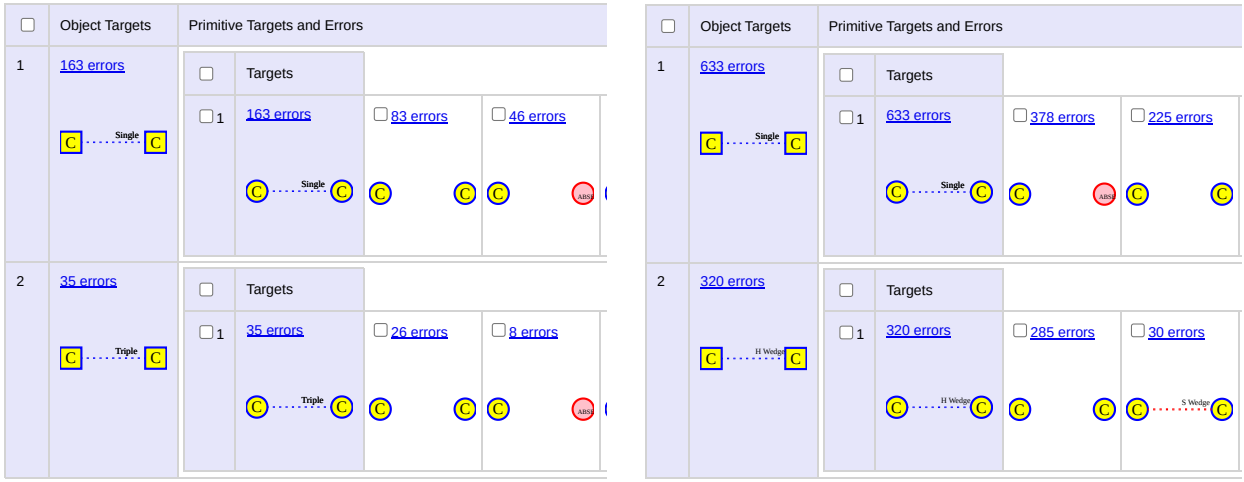
We first assign identifiers to nodes in the ground truth graph, which are atoms or named groups (e.g., SO_2) and hidden carbons at line intersections. the Indigo graph using code provided by MolScribe. We have adapted MolScribe code to locate atom/group names and hidden carbons in a PDF image for a molecular diagram generated using Indigo. Then, parser output graph nodes are given the identifier of the ground truth node that they have maximum overlap with, breaking ties arbitrarily. Where multiple output nodes overlap one ground truth node, or an output node does not overlap a ground truth node (e.g., missed line intersections produce extra hidden carbons), additional unique identifiers are created. Bonds are then defined using labeled edges between nodes using these bond types: (`single, double, triple, wavy, solid wedge, hashed wedge`).

After alignment, adjacency matrices are used to identify *all* structural differences from the labels in corresponding cells. Both rows and columns of adjacency matrices for: (1) ground truth, and (2) parser output, are labeled by the node identifiers obtained during alignment. Node labels are located in diagonal entries (e.g., (n_1, n_1)) and edge labels are provided in the off-diagonal entries (e.g., (n_1, n_2)). For nodes, we compute the percentage of ground truth nodes aligned with an output graph node with the same label (i.e., (R)ecall), and the percentage of output nodes aligned with an identically labeled ground truth node (i.e., (P)recision). We combine Recall and Precision using their harmonic mean F_1:

$$F_1 = \frac{2RP}{R+P} \ .$$

We also report the analogous F_1 measure for edges (bonds). An output edge is correct if its end nodes and label match ground truth. Finally, we report the percentages of molecules with correct structure (i.e., correct `MERGE` and `CONNECTED` relationships), and with both correct structure and node labels.

For fine-grained evaluation of ChemScraper, we require molecule graph representations for both ground truth and the predicted molecules. Given we have already created chemical structure graphs



(a) Default Rendering Parameters

(b) Hardest Rendering Parameters

Figure 3.13: Relationship Confusion Histograms for Renderings in Table 3.4 (truncated at right for space). Hyperlinks show molecules with specific errors, check boxes allow selecting molecules with errors for export. **Default rendering:** the top 2 errors are missing single and triple bonds. We can observe that in both cases, at times a missing (ABSENT) hidden carbon is the cause. **Hardest rendering:** missing single bonds are again the most frequent error, caused half of the time by a missing carbon. The second most-frequent error is missing hashed wedges between carbons, where no bond is detected, or because of misclassification of hashed wedges as solid wedges.

subsequently converted to CDXML format, we can readily employ these graphs for evaluation. It is important to note that the molecular graphs utilized for evaluation differ from the visual graphs created in Section 3.4 to annotate raster images.

Molecular Graphs for Evaluation. The predicted graph corresponds to the final stage in the parsing algorithm, shown in Figure 3.2(e). These graphs are generated in the final step of the born-digital parsing pipeline (see Figure 3.5). This graph assumes the representation of atoms or atom groups as nodes, with edges representing bond types associated with nodes, which may have one of the following types: `{Single, Double, Triple, Solid Wedge, Hashed Wedge}`. To construct a ground truth molecular structure graph, we use a `MOL` object generated by Indigo from the corresponding SMILES representation. We then extract atom positions along with the adjacency matrix for bonds between atoms using MolScribe code [100] with minor modifications.

We identify correspondences between nodes in parser output and ground truth graphs using atom coordinates from Indigo (ground truth) and Symbol Scraper (parser output). Minor discrepan-

cies in atom coordinates are resolved using minimum distances between corresponding atom pairs. Corresponding nodes are giving the same identifiers.

Finally, we create object-relationship label graph files (Lg files) as described in Section 3.4. 'Object' entries represent individual atoms or atom groups, and the 'Relationship' entries denote bond edges with bond type labels between the atoms, as opposed to specifying the type of connections between visual elements.

Analysis. We use `LgEval` to compare molecular graphs to obtain the metrics in Table 3.4. The table shoes a disparity between recognition rates when using labeled graphs (last column) vs. the exact SMILES matches shown in Table 3.3. This arises because SMILES string-based metrics lack sensitivity to direction and errors for 3D bonds, such as hashed and solid wedge bonds. In this way, SMILES exact matches may be misleading in terms of identifying correct molecular structures. In contrast, our graph-based metrics readily identify such errors.

Table 3.4 show a large decline in recognition rates when using the hardest rendering condition for the parser, despite only a 0.83% reduction in accurate detection of edges in molecular graphs. This is mainly due to the intricate network of edges and relationships, particularly in large structures with rings. Even a 1% error in relationships, as seen in the USPTO-Indigo dataset with 382,058 target relationships for 5,719 molecules, substantially affects accuracy.

In the `confHist` tool error summary (an excerpt is shown in Figure 3.13), common errors for the default rendering include missed single and triple bonds. The run for the hardest rendering parameters produces a notable increase in the count for the most frequent errors, including missing single and hashed wedge bonds. This unexpected difficulty with easier-to-detect bonds is due to the density of molecules in the hardest rendering condition, which produces short bond lines and a compact structure (See Figure 3.9(b)). This poses challenges for our graph transformations using thresholds to accurately detecting bonds or establishing correct connections between entities. This illustrates where greater use of visual features may be beneficial within the born-digital parser itself.

3.6 Summary

We presented two specialized born-digital parsing systems: one for extracting mathematical formulas and the other for molecular diagrams, both leveraging PDF graphics data without relying on OCR. SymbolScraper plays a crucial role by extracting precise character labels and locations from PDF rendering commands, bypassing traditional OCR methods. The mathematical parser integrates this

symbol data to construct Symbol Layout Trees (SLTs), which may be converted to formats like LaTeX or Presentation MathML. In the absence of complete PDF information, we use neural network-based models such as QD-GGA [72] and LGAP [114] to ensure formula structure recognition, focusing on segmenting connected components and classifying symbols.

For chemical diagrams, the ChemScraper parser efficiently tokenizes graphical elements like lines, polygons, and curves into molecular structure graphs, which can then be converted into chemical representations like SMILES and CDXML. This parser also addresses the challenge of training data scarcity by generating annotated raster images for visual parsing tasks. Both parsers provide outputs that are compatible with widely-used tools like ChemDraw, enabling seamless integration into chemical workflows.

We adapted adjacency matrix-based evaluation metrics from CROHME for both mathematical formulas and molecular graphs, ensuring detailed assessments of parsing accuracy. These metrics highlight structural errors not captured by simpler string-based evaluation methods like SMILES matching.

Limitations: While these parsers offer robust and accurate extraction capabilities for documents with vector-based content, they rely heavily on the availability of clean vector information. Many real-world documents contain a mix of vector and raster graphics or lack vector information entirely. This limitation underscores the need for extending our parsing approach to handle raster images through visual parsing, a problem we address in the next chapter.

In the next chapter, we extend our exploration to visual parsing from raw images. Here, we shift focus from PDF-based symbol extraction to image-based parsing, leveraging neural networks to analyze and reconstruct formula and diagram structures directly from pixel-based data.

Chapter 4

Visual Parsing from Raster Images

Documents include both born-digital PDF documents (e.g., where symbols in a formula may be available, but not formula locations [12, 113]) as seen in the previous chapter, and scanned documents, for which OCR results for formulas can be unreliable. In this chapter, we focus on the task of parsing the visual structure of formulas and diagrams from raster images after formulas have been detected/isolated, and parsing isolated formulas from connected components or visual primitives in binary images in particular.

The Line-of-Sight with Graph Attention Parser (LGAP) [114] introduced in this chapter builds upon the Query-Driven Global Graph Attention formula parser (QD-GGA) [72]. Both systems use line-of-sight (LOS) graphs to structure their input and generate SLTs that represent mathematical expressions. However, LGAP improves upon QD-GGA by incorporating additional context through LOS neighbors [50,52], enhancing spatial information via spatial pyramidal pooling [44], and refining representation of punctuation relationships to avoid requring edges that are missing in many LOS graphs. This chapter shows how LGAP increases the recognition accuracy for math formulas and addresses the challenges inherent in parsing mathematical expressions.

In parallel, we introduce Line-of-Sight Chemical Graph Parser (LCGP)—a neural network-based chemical diagram parser that extends the concepts of LGAP for molecular structures. LCGP is specifically designed to handle the intricacies of chemical diagrams by segmenting primitives such as lines, polygons, and curves from raster images, generating visual graphs, and converting them into molecular structure graphs. LCGP also employs iterative segmentation-aware learning, where the network refines input features across multiple runs until the merging of primitives stabilizes. The advancements made in LCGP for chemical diagram parsing can also be adapted for mathematical

formulas, and this will be discussed in Chapter 5.

The chemical visual parser introduced here recognizes molecules from raster images (pixel-based). Similar to the born-digital parser, ChemScraper, described in Chapter 3, the visual parser operates in a compiler-style multi-step architecture, starting with (1) primitive identification and progressing through (2) *visible* diagram structure recovery to (3) chemical structure generation.

The chemical visual parser starts by creating line-shaped contour primitives from a raster image that over-segment lines and characters. Just as for the born-digital parser, the visual parser creates a visual graph providing an explicit correspondence between an input image and recognized structure. The same tokenization and molecular graph generation steps used for the born-digital parser are then performed to produce the final molecular graph, which can be converted into formats like SMILES and CDXML for chemical applications. The LCGP network, based on a ResNet backbone, uses discrete attention mechanisms and dynamically updates classifier inputs with refined primitive contours during recurrent runs.

Both LGAP and LCGP benefit from taking a divide-and-conquer approach, with structure recovery in mathematical formulas driven by LOS-based neighbors, and molecular diagrams processed through overlapping windows. The parsers are evaluated using adjacency matrix-based metrics for graph comparisons, providing detailed insights into recognition errors. For chemical diagrams, we also employ SMILES-based evaluation to report structural differences missed by graph-based methods.

In summary, this chapter demonstrates the evolution from QD-GGA to LGAP for math parsing, and the transition from LGAP to LCGP for chemical diagram parsing. These parsers exemplify the potential of graph-based methods for extracting meaningful structure from visual data and offer avenues for future extensions. The methods introduced here set the stage for developing more unified parsing systems that can seamlessly handle both math and chemical structures from raw images.

4.1 The Parsing Model – LGAP and LCGP

In this section, we provide an in-depth overview of two parsing models, LGAP [114] and LCGP [112], both designed for visual parsing tasks but with specific adaptations for math formulas and chemical diagrams, respectively. By examining each model across their various components, we aim to clarify their similarities and differences in detail. Both LGAP and LCGP share a common

backbone and parsing strategy, grounded in the use of line-of-sight (LOS) features and multi-task neural networks, but diverge in their feature representations, input handling, constraints, and post-processing steps. The progression from QD-GGA to LGAP for math parsing, and subsequently from LGAP to LCGP for parsing chemical diagrams, reflects targeted improvements to accommodate the unique requirements of each domain.

LGAP (Line-of-Sight Graph Attention Parser) is an MST-based parser designed to improve mathematical formula parsing. Evolving from the QD-GGA model, LGAP enhances parsing accuracy by incorporating line-of-sight (LOS) context not only in constructing input graphs but also in supplementing visual features through neighboring primitives. This additional context helps capture local relationships between symbols, which are essential for accurate formula parsing. LGAP also addresses the QD-GGA model's spatial information loss by employing spatial pyramidal pooling [44] rather than isolated average pooling. This preserves critical spatial features, especially useful in recognizing complex math expressions with hierarchical structures like superscripts and subscripts. Furthermore, LGAP corrects limitations in representing punctuation, which previously relied on 'PUNC' edges between adjacent symbols. In many cases, the line-of-sight between parent symbols at left and punctuation marks at right, such as '.' or ',' is blocked by other symbols, causing removal of a ground truth edge in the graph.

LCGP (Line-of-Sight Chemical Graph Parser) is a multi-task neural network adapted from LGAP to parse chemical diagrams from raster images. While LGAP was tailored to the demands of math formulas, LCGP extends this approach to handle the complexities of molecular structures. Like LGAP, LCGP utilizes LOS-based features but modifies its feature representations, input processing, and constraints to reflect the distinct requirements of chemical diagrams. The model constructs visual structure graphs for molecular diagrams by classifying individual *queries* – line primitives and relationships through multi-task queries, similar to LGAP, but with modifications to accommodate chemical structures. The parser is trained using our ground truth representation for raster images illustrated in the previous chapter (3). Also, another additional strategy is used in LCGP for formulas that contain `MERGE` edges. We use two versions of the input: (1) with no labels, relations, or `MERGE` edges defined (i.e., raw primitive input), and (2) with no labels or relations, but *all* ground-truth `MERGE` edges provided. This allows the model to learn more quickly how to classify symbols and relationships from whole objects rather than their parts.

While LGAP and LCGP both build visual structure graphs, LCGP's focus on chemical diagrams necessitates further processing beyond MST-based restructuring. Compared to the born-digital parser, LCGP uses line primitives to replace the first stage of the pipeline in Figure 3.5, while

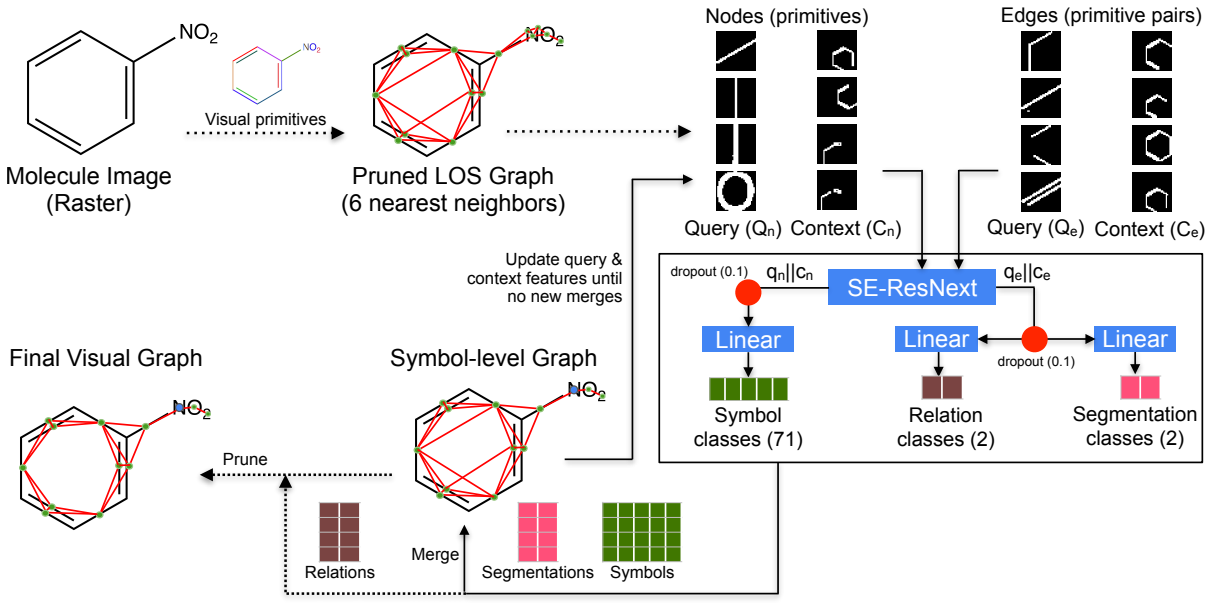


Figure 4.1: Parsing a Raster Image of Nitrobenzene ($C_6H_5NO_2$). Line contours are extracted as primitives, over which a pruned LOS graph is built. At top-right, four node and four edge queries are shown, at bottom-left their classification tensors (rows: queries, columns: classes). (Q)uery and (C)ontext features enter an SE-ResNext block. Two-layer Multi-Layer Perceptrons (MLPs) estimate probabilities for symbol, segmentation (MERGE), and relationship (CONNECTED) probabilities. Merges are applied (e.g., for 'N'), with symbol/relationship probabilities averaged across primitives. The model runs recurrently, updating queries and their contexts until no new merges are found (e.g., two passes for this example).

LCGP replaces the second and third stage up to step 3(b) to produce a visual graph (restructured MST). The remaining tokenization and semantic analysis steps (steps 3(c) and 4) are unchanged.

Through the evolution from QD-GGA to LGAP for math formulas, and further to LCGP for chemical diagrams, this section presents a unified parsing approach that adapts well across domains by building on a shared LOS-based parsing model and progressively refining feature and structural adaptations specific to each task. We delve into the parsing model by examining its key components, including the inputs and outputs, feature representations, and the multi-task CNN-based classification framework used for both LGAP and LCGP. We then explain how the models transform input graphs into output structures, specifically, a Symbol Layout Tree (SLT) for math formulas and a molecular graph for chemical diagrams. Each subsection will highlight how LGAP and LCGP address their domain-specific requirements while building on a shared foundation. Both models were implemented in PyTorch, with Python's NetworkX library facilitating graph representations and

Edmonds' algorithm to obtain MSTs/SLTs, where relevant. The open-source implementations for LGAP and LCGP are available for further development and research.[1]

4.1.1 Inputs

In both LGAP and LCGP, the initial input to the parsing model is a Line-of-Sight (LOS) graph constructed from visual primitives. This LOS graph effectively prunes edges between primitives by connecting only those pairs where an unobstructed line can be drawn from the center of one primitive to a point on the convex hull of another [71]. This approach helps reduce the number of edges while retaining essential spatial relationships, which is crucial for efficiency in graph-based parsing.

In **LGAP** for mathematical formulas, the input primitives are either handwritten strokes or connected components (CCs) from typeset images (see Figure 2.1). A handwritten stroke is represented by a list of 2D (x,y) coordinates obtained from sampled positions of a pen, trackpad, or other touch device. When parsing online handwritten formulas, our input is a graph over strokes images: strokes are individually rendered using their stroke coordinates. For typeset formulas, connected components derived from binarized formula images act as input primitives. For both handwritten and typeset formulas, we assume that primitives (stroke images or CCs) belong to exactly one symbol.

An initial complete graph is created with edges between every pair of primitives, but only LOS edges remain after pruning as suggested by Hu et al. [52]. This selective LOS graph [28], as shown in Figure 4.4, reduces the space of output graphs and number of classification decisions needed by eliminating irrelevant connections with few deletions of pertinent edges [52]. It focuses the model on key relationships—such as the relationship between a fraction line and '2'—by removing unnecessary edges from '2'. Figure 4.4, all edges between the '2' and other CCs other than the fraction line are pruned.

In **LCGP** for molecular diagrams, the input consists of line primitives obtained from rasterized molecular images, which often include lines, curves, and character components. Connections and merges exist only between nearby primitives in molecular diagrams, as reflected by our use of MSTs in the born-digital parser. Similar to LGAP, LCGP starts with a complete graph of visual primitives and then applies LOS constraints to retain edges only between directly visible primitives. An additional step here involves limiting each primitive's LOS connections to its $k = 6$ nearest neighbors to accommodate a four bond constraint in molecular diagrams – there can be at most 4

[1] LGAP/LCGP open-source implementation: `https://gitlab.com/dprl/lgap-parser`

lines or characters in a bond; we choose 6 neighbors to accommodate over-segmentation in visual primitives, such as when bond lines or characters may be broken into multiple segments. This adjustment ensures that the model remains focused on relevant local interactions, particularly in cases of over-segmentation.

Modified Punctuation Relationship in Ground Truth SLTs (LGAP). Mahdavi et al. [71] added a new spatial relationship called *PUNC* in the InftyMCCDB-2[2] dataset, for distinguishing horizontal relationships with punctuation sybmols from other horizontal relationships to make their visual features more consistent and thereby identifiable. However, this modification can lead to missing ground truth punctuation edges in the input LOS graph, often when a symbol has a subscript and is followed by a punctuation symbol on its writing line. This happens since there is good chance that the line of sight between the parent symbol and the punctuation is blocked by one or more primitives between them, usually a large subscript. For the example in Figure 4.2, the ground truth edge between the symbols 'z' and 'COMMA' is blocked by the primitives corresponding to 'i'. Here the *PUNC* relationship will be missed, due to the missing edge in the input LOS graph representation. Additional heuristics in LPGA [71] modify the LOS graph to add missing punctuation based on angles and relative sizes of symbols, which are not very robust.

To address this issue, we modified the assigned parent node for *PUNC* edges as shown in Figure 4.2. Instead of the original parent, which may be far left on the writing line and/or blocked in the LOS graph, we assign the PUNC parent node to the symbol immediately at left of the punctuation symbol. This modification ensures all *PUNC* edges are included in input LOS graphs. This also improves training/learning, as relationships between *PUNC* parent and child nodes using more consistent visual features (see Section 4.2).

4.1.2 Features

In both LGAP and LCGP, we develop feature representations based on the line-of-sight (LOS) context, with each model using unique strategies to improve accuracy and efficiency in parsing mathematical formulas and chemical diagrams.

In LGAP, for formula images, we extract CC contours by first smoothing them and then sampling contours to produce trace points. For handwritten formula images, trace points are already available. These trace points are interpolated, normalized, and scaled to a height of 64 pixels while preserving the aspect ratio: this results in formula images with a fixed height but varying width. The full

[2] `https://www.cs.rit.edu/~dprl/data/InftyMCCDB-2.zip`

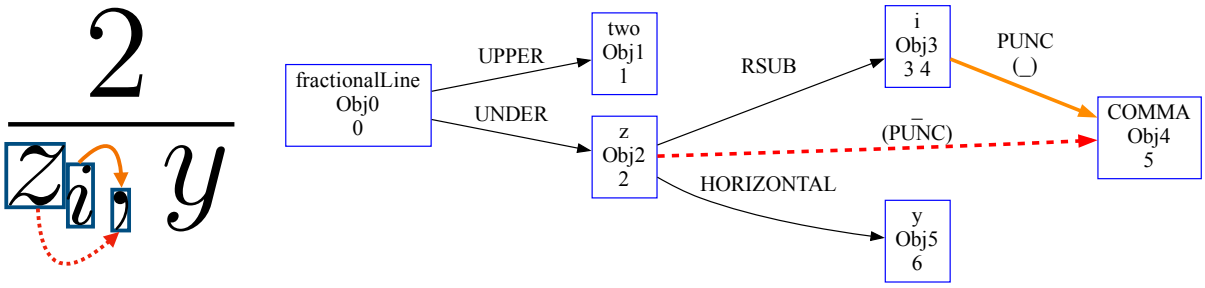


Figure 4.2: Modifed puncutation (*PUNC*) ground truth representation. The old *PUNC* edge is shown using red dashed arrows, and its corresponding new edge is shown with solid orange arrows. The original *PUNC* edge between nodes 'z' and 'COMMA' is missing in the LOS graph, as can be seen in Figure 2.3.

normalized formula image is fed into a CNN encoder backbone to compute the global visual features, and points on sampled contours are used to construct the LOS input graph (described below).

Input regions are processed to produce binary attention masks created for input queries corresponding to individual primitives (nodes) or primitive pairs sharing an edge in the LOS graph. Binary masks for directed edges between primitive pairs are concatenated with the binary mask for the parent primitive, to increase identifiability and thus classification accuracy [72]. Query binary masks filter the CNN global feature map, providing focus on image regions pertinent for three classification tasks: (1) masking individual CC/strokes for symbol classification (for nodes), and masking CC/stroke pairs for (2) segmentation and (3) spatial relationship classification (for directed edges).

We introduce an additional binary LOS mask, which consists of a binary sum (OR) of the query mask with the binary mask of all neighboring primitives in the LOS graph (see Figure 4.3). These additional LOS binary masks serves to provide visual spatial context information from the neighbors in the LOS graph. As for primitive and primitive pair binary masks, LOS masks are downsampled and weight the global feature map produced by the encoder in order to focus on the relevant parts of the input image for the three classification tasks (symbols, segments, relationships).

However, a significant limitation in LGAP is that these binary masks often have sparse information due to the large image size relative to the localized regions with active pixels. Only the query primitives or primitive pairs contribute active pixels within the full formula image, resulting in a high proportion of zero-valued pixels. This sparsity limits the network's ability to generalize meaningful patterns efficiently and can lead to suboptimal feature representations.

In LCGP, we addressed these limitations by shifting to a more compact and context-focused fea-

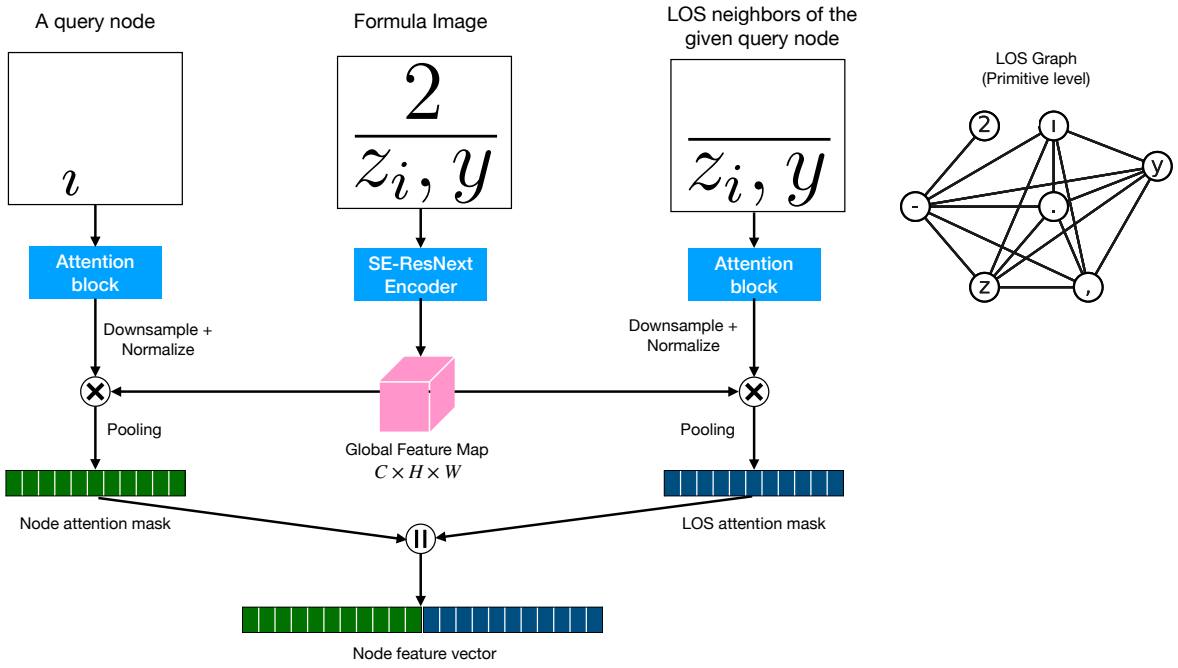


Figure 4.3: Binary attention masks in LGAP. The input primitive query mask (represented here by the base of the letter 'i') and its corresponding LOS masks are used to generate the attention masks by performing element-wise multiplication with the global feature map. The two attention masks are concatenated to get the node feature vector that is utilized for symbol classification. Note that the same process is applied to the primitive pair binary masks and LOS mask to generate the primitive pair feature vector for classifying directed edges.

ture representation. Instead of processing entire formula windows, visual features are created by drawing line primitive contours directly into 28×28 binary images for (1) individual primitives (node queries), (2) primitive pairs (edge queries), and (3) context images containing the $k = 6$ nearest neighbors centered around each query (one per node/edge query). This approach significantly reduces the amount of zero-padding and allows each primitive or pair to be tightly centered, minimizing irrelevant areas and focusing the network on the actual region of interest. This setup is similar to the MNIST digit classification task [60], where the network processes compact, centered images of individual symbols, which aligns well with LCGP's symbol classification task and improves the network's ability to recognize symbols accurately.

LCGP's fixed-size feature vectors eliminate the variable-width issue in LGAP, where varying formula widths made batch processing inefficient and required extensive packing. With uniformly-sized 28×28 feature maps, LCGP achieves efficient batch processing, allowing for better generalization and

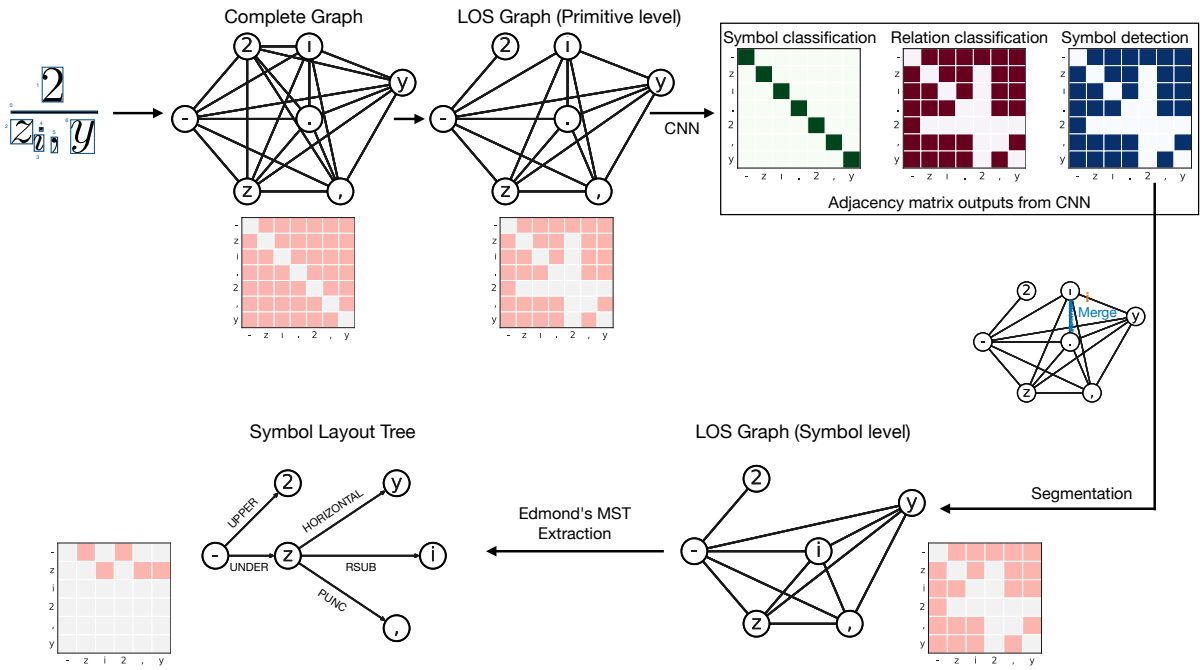


Figure 4.4: LGAP formula parsing example. A complete graph over input primitives (here, CCs) is pruned, sub-images corresponding to CCs and CC LOS edges are given symbol, merge/split, and spatial relationship class distributions. Based on merge/split probabilities primitives are merged into symbols (here, for the 'i'), and finally an SLT is produced from remaining spatial relationship edge by extracting a **directed MST** (arrows omitted for legibility). Symbol and relationship class probabilities are averaged when merging primitives into symbols.

reduced memory requirements. Additionally, the queries are shuffled throughout the dataset—even across formulas—treating each query as independent. This strategy promotes robust learning, as the network processes a diverse set of primitives and primitive pairs across different formula contexts.

In summary, LCGP enhances the feature representation by focusing on tightly-cropped query and context images, reducing sparsity, and optimizing for batch efficiency. These features are compact yet effective, capturing essential information while reducing noise, resulting in a more efficient and accurate parsing approach compared to LGAP.

4.1.3 Multi-Task CNN for Classifying Primitives and Primitive Pairs

In this subsection, we delve into the multi-task CNN architecture used for classifying primitives and primitive pairs in both mathematical formulas and chemical diagrams. The goal is to segment

symbols, classify their types, and determine the spatial or structural relationships between them. The evolution from QD-GGA to LGAP and subsequently to LCGP highlights the enhancements made in contextual feature extraction, attention mechanisms, and efficient processing. This section will describe how the CNN model processes visual features derived from input primitives and their pairs, performing symbol classification for individual primitives and segmentation and relationship classification for primitive pairs. Through this approach, the model captures both local details and contextual information, enabling accurate parsing of visual structures within mathematical and chemical diagrams.

The LGAP model evolved from the end-to-end trainable multi-task learning model, QD-GGA [72], focusing on parsing mathematical formulas through improved attention mechanisms and feature utilization. In LGAP, image features for primitives and edges within the Line-of-Sight (LOS) graph are extracted using a CNN architecture, facilitating the classification of three primary tasks: symbol segmentation (detection), symbol classification, and spatial relationship classification. This multi-task learning framework builds upon the QD-GGA model, where the estimated probability distributions for each classification task are stored in three adjacency matrices defined over the LOS graph. Symbol classification probabilities for primitives are represented along the LOS adjacency matrix diagonal, and merge/split and spatial relationship probabilities are represented for directed LOS edges in off-diagonal entries, as seen at top-right in Figure 4.4.

The CNN contains an SE-ResNext backbone [49, 138] used to compute a global feature map from the input formula image along with attention modules to produce attention relevance maps from the binary masks described in the previous Section. The SE-ResNext backbone and the attention modules are trained concurrently using a combined cross entropy loss function for all three classification tasks. Attention modules are used to produce 2D relevance maps from the 2D binary masks for nodes, edges and LOS neighbors separately by convolving them through 3 convolutional blocks, trained for each task, where each block has 3 kernels of size 7×7, 5×5, and 5×5 [72].

For this work, we modified the QD-GGA SE-ResNext encoder by reducing the number of output channels from 512 to 256, to reduce the number of parameters requiring training. This results in a 256-dimensional feature vector for symbol classification, and two 512-dimensional feature vectors for segmentation and relation classification, respectively.

In LCGP, we introduce enhancements to address some limitations in LGAP and better adapt the model for chemical diagram parsing. Unlike LGAP, LCGP does not generate a global feature map for the entire formula image, nor does it use separate attention modules to produce relevance maps. Instead, each query and its context are represented by independent, fixed-size 28×28 binary images,

which significantly reduces the proportion of zero-valued pixels and centers the focus on relevant primitives. This approach eliminates the sparsity and variability of the input image sizes present in LGAP, allowing for more compact and effective feature extraction.

In LCGP, query and context images are passed separately through the same SE-ResNext backbone. This backbone generates 32 feature maps for each 28×28 input. We further refine the architecture by replacing the large 7×7 convolutional kernel with a smaller 3×3 kernel, applying a stride of 1 and using `same` padding, while removing the first maxpool layer to preserve more detailed features since input images are already small. Note that the number of output channels is further reduced from 256 in LGAP to 32 in LCGP, reflecting the improved feature representation and reducing model complexity without compromising accuracy.

By treating each query independently and shuffling queries across the dataset, LCGP achieves robust learning while maintaining uniform and efficient processing through fixed-size features. This approach, inspired by the MNIST-like classification of primitives [60], allows LCGP to maintain local context while focusing on critical elements for parsing.

Spatial Pyramidal Pooling and 1-D Context Module.

Average pooling uses a single average activation value to represent the convolutional response in a region [2]. Without the use of windowing within an input image, a large amount of important spatial information is lost during average pooling. Jose et al. [54] use a pyramidal pooling method, which integrates spatial information into the feature vectors, producing more compact and location invariant feature vectors. He et al. [44] introduced the spatial pyramid pooling (SPP) strategy for deep CNNs by adopting the widely used spatial pyramid matching (SPM) strategy [58]. SPP captures spatial information by pooling features within equal-sized regions of the feature map for increasing numbers of sub-regions, forming a pyramid of overlapping sub-regions (e.g., whole image, left-right, top-down, 3 horizontal regions, etc.). SPP is used to capture spatial information across multiple horizontal and vertical regions, providing more spatial information and lower variance in features.

For LGAP, we use 5 levels with 11 regions in pooling outputs: this includes 1 full feature map, 2 vertical bins, 3 vertical bins, 2 horizontal bins, and 3 horizontal bins. To reduce the growth in parameters due to increasing pooling regions from 1 to 11, we also reduce the number of output channels in the SE-ResNext encoder from 512 to 64 (a factor of 8). The resulting node and edge feature vectors have lengths of 704 (i.e., 64×11) and 1408 (i.e., 128×11), respectively. With the

new LOS attention masks, the feature lengths are 1408 (i.e., 704×2) and 2816 (i.e., 1408×2) respectively. The edge features have an additional factor of 2 due to the concatenation of parent primitive attention masks, as mentioned earlier.

1-D Context Module in LGAP. The weighted feature maps are pooled to produce feature vectors. A '1D context' module then performs a 1-by-3 convolution along the sequence of query feature vectors used as classification input [72]. The convolution aggregates features from the previous $(i-1)^{th}$ and next $(i+1)^{th}$ query in input order for the i^{th} query. In the input, queries are spatially sorted top-down, left-right by the top-left coordinate of a query's bounding box. Feature vectors are passed to one of three fully-connected output layers for three classification tasks: segmentation and relationships (edges), and symbol classification (nodes). We also examine removing the 1-by-3 convolutional context module, since its notion of neighbor is not based directly on spatial proximity as discussed earlier (see Section 4.2).

For LCGP, feature maps are average pooled in 7 pyramidal regions (image, 3 vertical, 3 horizontal). The final query visual features are the pooled convolution responses for a node/edge and its associated context (i.e., $q_n||c_n$ or $q_e||c_e$). For 32 features maps with 7 average-pooled regions, the query and context images produce $2 \times 224 = 448$ features. We add three positional encodings to query vectors in the form of bounding boxes (BBs) $(x_{min}, y_{min}, x_{max}, y_{max})$ with coordinates normalized to be percentages of width/height:

1. Query BB relative to the formula window
2. Query BB relative to the context window
3. Context window BB relative to the formula

For edge queries we use the combined primitive pair position as the query position. Adding these three BBs each query vector contains $448 + (3 \times 4) = 460$ features. Dropout is applied for regularization (rate of 10%).

1-D Context `REMOVED` in LCGP. A key improvement in LCGP is the elimination of the 1D context module, which was used in LGAP for aggregating local context along a query sequence. This removal is justified because LCGP already achieves context-aware representations through the use of tightly cropped query and context images and the inherent spatial pooling mechanism within the SE-ResNext backbone. Additionally, removing the 1D context module simplifies the architecture, reduces computational overhead, and allows for direct input of query and context

feature vectors into linear layers for classification tasks. This change leads to more efficient and consistent processing without compromising the model's ability to capture relevant context.

Multi-Task Cross Entropy Loss Function and Training.

In both LGAP and LCGP models, we utilize a multi-task learning approach, applying a cross entropy (CE) loss function with a softmax activation for each classification task. This approach, originally used in QD-GGA [72], combines losses for symbol segmentation, symbol classification, and relationship classification.

For LGAP, the data instances correspond to individual formulas, where each formula contains N nodes (primitives such as strokes or connected components) and E edges (between primitives in the line-of-sight graph). The total loss for a formula is computed as:

$$\delta(N, E) = \sum_{n \in N} CE(n, S) + \sum_{e \in E} \left(CE(e, D) + CE(e, R) \right), \tag{4.1}$$

where D denotes segmentation ground truth edge labels, R denotes relationship ground truth edge labels, and S represents the ground truth symbol labels for primitives (nodes). The loss is optimized per formula, ensuring that the relationships within each formula are treated as a coherent unit. For backpropagation, the Adam optimizer is used with a learning rate of 0.0005, momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and a weight decay (L2 regularization) of 0.1.

For LCGP, the data instances differ in that they consist of independently shuffled queries for nodes (primitives) and edges (primitive pairs). Unlike LGAP, where the total loss is computed for each formula, LCGP computes the total loss over batches of independent queries, making it more robust by allowing shuffling across the entire dataset, including different formulas within the dataset. The total loss for a batch of queries is computed as:

$$X(Q_n, Q_e) = \sum_{q_n \in Q_n} X_S(q_n) + \sum_{q_e \in Q_e} \left(X_M(q_e) + X_C(q_e) \right), \tag{4.2}$$

where X_S, X_M, and X_C are the cross entropy losses given the correct target response vectors (1-hot) and softmax distributions for (S)ymbol classification, primitive (M)ERGE, and primitive (C)ONNECTED outputs, respectively. Here, Q_n represents the set of node queries (primitives) and Q_e represents the set of edge queries (primitive pairs) within a given batch. Random over-sampling of node and edge queries is employed to balance positive and negative examples, particularly for `MERGE` and `CONNECTED` edges, ensuring equal numbers of positive and negative samples.

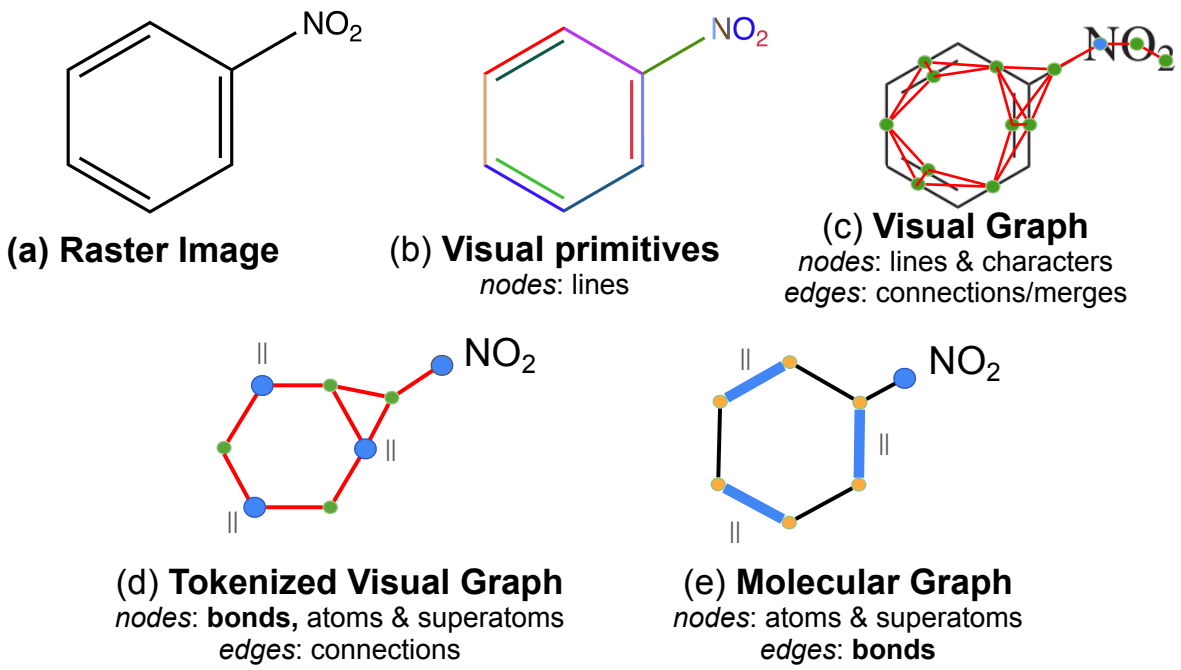


Figure 4.5: Parsing Nitrobenzene ($C_6H_5NO_2$) from a raster image (a). (b) **Visual primitives.** The N is split into 3 lines. (c) **Visual Graph** extracted from visual parser. (d) **Tokenized Visual Graph** with merged nodes (bonds and named groups). (e) **Molecular Graph**. Blue nodes show the primitives of N merged into a character (c) and double bonds and atom/group names in (d) and (e). In (e) orange nodes are 'hidden' carbon atoms, and single/double bonds are converted from nodes to edges.

Node and edge queries are processed together in batches, with a batch size of 64. The Adam optimizer is used for backpropagation with a learning rate of 0.0005, β values of (0.9, 0.999), and no weight decay. This batch-based approach improves model robustness by diversifying the context and increasing generalization through the shuffling of queries across different formulas.

4.1.4 Parsing: Transforming Input Graphs into Output Graphs

The transformation of input graphs into output graphs represents a critical step in both mathematical formula parsing and chemical diagram recognition. This process involves interpreting node and edge classifications to build meaningful structures that represent the underlying content. For LGAP, the focus is on constructing a directed Symbol Layout Tree (SLT) from connected components or strokes representing mathematical symbols and their spatial relationships. In contrast,

LCGP constructs a tokenized visual graph for chemical diagrams by merging and connecting primitives such as atoms, bonds, and other graphical elements. Both approaches utilize a line-of-sight (LOS) graph as a foundational representation, but differ in how they transform this input graph into their respective output structures. While LGAP relies on a directed Maximum Spanning Tree (MST) approach with specific constraints, LCGP employs a more flexible pruning strategy with recurrence to handle the cyclic nature of chemical diagrams. The following section outlines the methodologies for each model.

In LGAP, after the LOS graph for mathematical formulas is constructed with nodes representing primitives (e.g., connected components or strokes) and edges representing spatial relationships, the multi-task CNN model assigns class probability distributions to each node and edge. The model handles three main tasks: symbol detection (segmentation), symbol classification, and spatial relationship classification across 207 symbol classes (including digits, Latin and Greek characters, trigonometric variables, fraction lines, etc.), nine relationship classes (e.g., horizontal, right subscript, right superscript, under, upper, punctuation, no relation), and two primitive merge classes (merge/split decisions).

The parsing begins by merging all LOS segmentation edges with a higher probability of being classified as `MERGE` rather than `SPLIT`. This effectively merges all connected components over primitives in the resulting segmentation graph into symbols. For the example in Figure 4.4, primitives corresponding to the symbol 'i' are merged into a single symbol node, along with their corresponding relationship edges. For merged primitives, symbol and relationship probabilities are averaged.

To construct a Symbol Layout Tree (SLT) representing the hierarchical structure of a formula, we apply Edmond's arborescence algorithm [38] to produce a directed Maximum Spanning Tree (MST). The MST maximizes the sum of relationship probabilities in selected directed edges, where the highest-probability relationship is chosen for each edge. To avoid duplicate edges with the same spatial relationship from one parent symbol to multiple child symbols, we enforce a constraint that a parent symbol may have at most one edge per relationship type (e.g., to prevent having two horizontal relationships from one symbol to two symbols at right). We replace edges that duplicate a relationship by removing the lower-probability edge, and replacing the lower probability edge's label with its next-highest relationship class probability, and then rebuild the MST, repeating until this unique relationship constraint is satisfied.

Figure 4.4 shows how a complete graph over primitives is constrained and annotated with classification probabilities in stages, producing an SLT as output [155]. There are bidirectional edges between strokes or CCs belonging to the same symbol, with the edge label matching the symbol

class. Spatial relationships are represented by directed labeled with their spatial relationship type. Relation types include *Horizontal*, *Rsub* (right subscript), *Rsup* (right superscript), *Upper*, *Under*, and *PUNC* (for punctuation).

In LCGP, the approach is adapted for parsing chemical diagrams. Similar to LGAP, LCGP utilizes a multi-task neural network to assign class probabilities, but operates with a different set of classes. As seen in Figure 4.1, node and edge queries are classified using three two-layer multi-layer perceptrons (MLPs): 71 node symbol classes, including chemical elements (characters), digits, straight lines, charges (+,-), parenthesis, etc.), two edge primitive merge classes (`MERGE` and `SPLIT`), and two edge primitive connection classes (`CONNECTED` and `NOT_CONNECTED`). For each classification, a hidden linear layer (512 units) is fully connected to the class output layer. For node queries the Unlike LGAP, the final graph structure for chemical diagrams is not constrained to be a tree, so Edmonds' algorithm is not applied. Instead, the parser employs a simpler edge pruning strategy.

Classification outputs guide the merging of primitives into symbols and establish connections between symbols. The LCGP parser uses a recurrent execution strategy to iteratively refine symbol segmentation.

Recurrent Execution in LCGP

The parser segments symbols bottom up from input primitives, updating query and context images during recurrent execution. Execution is performed recurrently until edge queries classified as `MERGE` with probability > 0.5 are unchanged from the previous pass (i.e., a fixed point is reached). On a recurrent execution, query images, context images, and positional encodings are all updated for merged primitives. Merges are identified by connected components along `MERGE` edges.

Note that this is not a conventional recurrent neural network (RNN) where a state vector is updated across executions. Instead, we simply update input features directly as the segmentation changes. For example, an N broken into three primitives may be merged in the second pass to produce three node queries containing all three primitives. This allows the N to be classified in a single query, rather than in three parts within the first iteration. Here the query images are identical for each merged primitive, but note the context image for the primitives will differ because they are centered on the original input primitive associated with each query. This addresses class imbalance by representing multi-primitive symbols multiple times, each with a slightly different context image.

Recurrent execution stops when no change in `MERGE` decisions is identified. Edges identified with a

probability of being `CONNECTED` > 0.5 are selected; any edges not selected for `MERGE` or `CONNECTED` are removed. Symbol and relationship probabilities are then computed by averaging them across primitives in segmented symbols and their connections.

This simpler, less constrained approach avoids the complexities of Edmonds' algorithm and is well-suited for chemical diagrams, where cyclic graph structures may be present.

After constructing the final tokenized visual graph, the graph undergoes a transformation to a chemical graph using the same deterministic steps described in Stage 4 of the born-digital parser outlined in Chapter 3. This process, known as semantic analysis, maps the visual syntax of the graph into a structured chemical representation. The semantic analysis stage includes identifying hidden elements such as carbon atoms at line intersections and mapping structures represented solely by name to their corresponding molecular subgraphs. For instance, names like NO_2 are replaced by subgraphs that contain one nitrogen and two oxygen atoms connected to a hidden carbon atom. This step ensures that the output graph accurately captures the chemical structure beyond what is visually represented, creating a molecular graph for downstream applications.

4.2 Evaluation and Results

This section evaluates the proposed LGAP and LCGP models on math and chemical datasets respectively. It explores the effects of modifications to punctuation relationship representations, changes to CNN architectures, and improvements in visual feature representations for mathematical formula parsing using the LGAP model. We also present initial benchmarking and analysis of the LCGP model on chemical diagram datasets. The evaluation focuses on measuring recognition accuracy, computational efficiency, and identifying areas for further improvement. This section is divided into two parts: the evaluation of the LGAP model for mathematical formula images and the evaluation of the LCGP model for chemical diagrams.

4.2.1 Evaluation of Math Formula Recognition (LGAP)

In this section, we evaluate the LGAP model's performance on mathematical formula images using a well-defined dataset, and rigorous evaluation metrics to measure the model's effectiveness and accuracy in parsing mathematical expression images.

Dataset. We use InftyMCCDB-2[3], a modified version of InftyCDB-2 [120]. The dataset contains binary images for isolated formulas from scanned articles, with formulas containing matrices and grids removed. The training set has 12,551 formulas, and the test set contains 6830 formulas. The dataset includes 207 symbol classes, and 9 relationship classes: *Horizontal*, *Rsub* (right subscript), *Rsup* (right superscript), *Lsub* (left subscript), *Lsup* (left superscript), *Upper*, *Under*, *PUNC* (for punctuation), and *NoRelation* (see Figure 4.2). The training and test sets have approximately the same distribution of symbol classes and relation classes.

Evalation Metrics. We report expression recognition rates for (1) *Structure:* unlabeled SLTs with correct nodes (CC groups for symbols) and edges (for relationships), and (2) labeled SLTs where symbol classes and relationship types must also be correct. For a finer-grained analysis, we also report detection and classification F-scores for symbols and relationships using the LgEval library originally developed for the CROHME handwritten formula recognition competitions [85]. F-scores are the harmonic mean of recall and precision for the detection of target symbols and relationships (2RP / (R+P)). We report both: (1) *detection f-scores*: quantify properly detected/segmented CC groups for symbols, and the presence of an edge (relationship) between two properly segmented symbols, and (2) detection + classification f-scores: here a symbol or relationship is correct if has been detected correctly and assigned the correct symbol or relationship label.

Implementation Details. Experiments were run on two servers and two desktop machines, all with hard drives (HDD):

1. 2 x GTX 1080 Ti GPUs (12GB), 8-core i7-9700KF (3.6 GHz), 32 GB RAM
2. 2 x GTX 1080 Ti GPUs (12GB), 12-core i7-8700K (3.7 GHz), 32 GB RAM
3. 4 x RTX 2080 Ti (12GB), 32-core Xeon E5-2667 v4 (3.2 GHz), 512 GB RAM
4. A40 (48GB), 64-core Xeon Gold 6326 (2.9 GHz), 256 GB RAM

Effect of Modifying the PUNC Relationship in Ground Truth

As described in Section 4.1.1, we adjusted the ground truth representation for punctuation relationships to better capture spatial nuances and correct labeling issues. Specifically, this involved redefining how punctuation was represented relative to neighboring symbols to minimize ambiguity.

[3] `https://www.cs.rit.edu/~dprl/data/InftyMCCDB-2.zip`

Table 4.1: Effect of modifying *PUNC* relationship representation for InftyMCCDB-2. F1 and expression rate metrics are defined in Section 4.2.1

	Symbol (F1)		**Relation (F1)**		**Expression Rate**	
	Detect	**+Class**	**Detect**	**+Class**	**Structure**	**+Class**
Original relationships	**98.23**	95.21	94.63	94.28	89.21	81.77
Modified PUNC	98.22	**95.23**	**94.93**	**94.56**	**90.45**	**83.00**

Table 4.2: Effect of LGAP Spatial Pyramidal Pooling (SPP) and 1D context module. Feature vector sizes given as *(node-size, edge-size)*. Modified PUNC representation used

Feature Vectors			**Symbol (F1)**		**Relation (F1)**		**Expression Rate**	
Pool	Sizes	1D-context	**Detect**	**+Class**	**Detect**	**+Class**	**Structure**	**+Class**
Avg	64,128	True	96.99	90.24	91.18	90.62	85.58	66.41
SPP-Avg	704,1408	True	**98.30**	**95.25**	**94.50**	**94.09**	**88.64**	**80.78**
Avg	64,128	False	89.84	34.68	67.68	65.23	64.32	17.57
SPP-Avg	704,1408	False	95.64	87.55	86.16	84.49	77.76	68.80

This change led to a 1.23% improvement in the formula recognition rate (Structure + Classification), as shown in Table 4.1. The outputs were generated using a reduced encoder architecture with 256 channels and a context module, as detailed in Section 4.1.3. Error analysis with the `confHist` tool in the LgEval library confirmed that the revised representation more accurately identified and labeled punctuation relationships that were previously missed or misclassified.

Enhancing Visual Features: SPP and Increased LOS Context

To further improve feature representation, we introduced modifications in feature extraction using Spatial Pyramidal Pooling (SPP) and incorporated context by using additional LOS (line-of-sight) neighbors, as described in Sections 4.1.1 and 4.1.3. These changes aimed to capture richer spatial information and enhance context sensitivity for improved classification and segmentation. Our experiments evaluated whether these adjustments led to improved recognition accuracy across the three tasks, including symbol detection, segmentation, and relationship classification.

Spatial Pyramidal Pooling. Table 4.2 shows that when the 1D context module is used, the formula recognition rate improves 14.37% after replacing average pooling by spatial pyramidal pooling (first two rows in Table 4.2). This difference in expression rate increases dramatically to 51.23% when the context module is removed (last two rows in Table 4.2). These results suggest

Table 4.3: Effect of Adding LOS Neighborhood Masks to LGAP SPP-Avg Model. Original ground truth used (without PUNC modification)

Feature Vectors			**Symbol (F1)**		**Relation (F1)**		**Expression Rate**	
LOS	Sizes	1D-Context	**Detect**	**+Class**	**Detect**	**+Class**	**Structure**	**+Class**
False	704,1408	True	97.96	94.48	94.36	93.89	88.70	78.90
True	1408,2816	True	98.32	**95.66**	**94.85**	94.35	89.27	**83.27**
False	256,512	True	98.23	95.21	94.63	94.28	89.21	81.77
True	512,1024	True	**98.39**	95.49	**94.85**	**94.46**	**89.36**	81.89
False	256,512	False	95.16	83.78	86.48	85.22	79.72	65.26
True	512,1024	False	95.40	85.97	86.27	85.07	80.23	70.09

that spatial pyramidal pooling greatly improves visual features, allowing us to obtain recognition rates close to the QD-GGA model with 256 encoder channels using only 64 encoder channels. However, removing the 1-D context module reduces the expression rate using the new SPP features by 13.17%, and the original single-region average pooling features reduces dramatically (48.83%). This demonstrates that the new SPP features are beneficial, and shows the importance of context.

Adding LOS Context through Neighborhood Embeddings. Next we evaluate the impact of including additional LOS neighbors in selecting image regions for queries in the attention modules, as outlined in Section 4.1.2. We hypothesized that incorporating the context from LOS neighbors would reduce ambiguity for visually similar symbols/relationships. Experiments were performed using both the 64-channel encoder output with spatial pyramidal pooling (using 11 bins, resulting a feature vector of lengths 1408 and 2816 (when LOS context is used), as well as the original 256-channel output with average pooling and the original representation. We also investigated the effect of using LOS masks when the context module was removed. The results in Table 4.3 show improvements in recognition rates at the symbol, relation, and formula levels when LOS neighborhood masks are used. Further, the expression rate accuracy is increased 2.49% over the best SPP model in Table 4.2.

Error Analysis.

An error analysis using the `confHist` tool in LgEval on our best performing model (LOS context, spatial pyramidal pooling, and 1D context; second row in Table 4.3) reveals the majority of symbol classification errors occur between visually similar symbols, such as $(i, j), (m, n), (l, 1)$, (α, a), and (LeftParanthesis, Vertical), in decreasing order of frequency. This highlights needed improvements in

Table 4.4: Benchmarking MST-based Parsing Models on InftyMCCDB-2

MST Model	Symbols		Relationships		Formulas	
	Detect.	+Class	Detect.	+Class	Structure	+Class
LGAP (this paper)	98.32	95.66	94.85	94.35	89.27	83.27
QD-GGA	98.50	94.54	94.43	93.96	87.77	76.72
LPGA_{RF}	99.34	98.51	97.83	97.56	**93.81**	90.06
LPGA_{CNN}	**99.35**	**98.95**	**97.97**	**97.74**	93.37	**90.89**

local visual features for symbols. For relationships, the most frequent errors are missing relationship edges in wide formulas containing a large number of symbols. This type of error can be attributed to the preprocessing step used for inputs: with the height of all formulas fixed at 64 pixels and the aspect ratio preserved, image resolution is noticeably reduced for wide formulas. This leads to features extracted from low-resolution input images being used to locate spatial relationships between connected components for very wide formulas.

We also note that expression rates are influenced by small changes in symbol and relationship recognition accuracy, which may amplify expression rates differences between conditions. For example, if a formula has just one symbol or relationship wrong, it is not counted as correct in the expression rate.

Benchmarking MST-Based Math Parsers

As seen in the previous experiments, the LGAP model that obtained the highest expression rate used a combination of Spatial Pyramidal Pooling, line-of-sight attention masks, and a modified punctuation representation in ground truth. We next benchmark this best LGAP model against previous MST-based visual parsers applied to InftyMCCDB-2. Results are presented in Table 4.4. Performance for LGAP relative to the QD-GGA model it extends is better in every metric, aside from a very small decrease in symbol detection F1 (-0.18%). The expression rate has increased 6.55% over QD-GGA. Unfortunately, performance is weaker than the LPGA models, with an expression rate that is 7.62% lower.

Despite LGAP's slightly weaker performance than LPGA [71], the LGAP offers substantial benefits in speed and efficiency. The encoder and attention modules are trained end-to-end on a joint loss for multiple tasks in a single feed-forward pass, making the training and execution process much faster than LPGA. Running on a desktop system with two GTX 1080 Ti GPUs (12GB), an 8-core i7-9700KF processor (3.6 GHz) and 32 GB RAM, LGAP requires 25 minutes per epoch to train on

12,551 training images and 11 minutes, 12 seconds to process the 6,830 test formula images (98.4 ms per formula).

Opportunties for further improvements include improving context usage through graph neural networks, as well as more sophisticated graph-based attention models to replace the current 1D context module from QD-GGA.

4.2.2 Evaluation of Chemical Diagram Recognition (LCGP)

In this section, we evaluate the performance of our Line-of-Sight Chemical Graph Parser (LCGP) for recognizing chemical diagrams from raster images. The evaluation covers datasets used, metrics employed, benchmarking against existing methods, and analyses of strengths, limitations, and areas for improvement. This analysis focuses on demonstrating LCGP's ability to extract accurate molecular structure graphs and highlights the impact of training data limitations and structural diversity on recognition outcomes.

Dataset. We generate our own visual parser training data for LCGP. We use 5000 molecules (46 unique SMILES characters) extracted from PubChem[4] prepared by the MolScribe team [100]. For benchmarking, we use the USPTO synthetic dataset with 5,719 PNG images generated by the Indigo toolkit from SMILES strings (37 unique SMILES characters) [101].

Additionally, as described in Chapter 3, we generate annotated visual graph data for training our visual parser that recognizes from raster images. This comprises 3,416 label graph files from the original pool of 5,000 molecules sourced from PubChem that could be accurately converted into exact SMILES strings. Errors include 240 diagrams mis-recognized from valid visual primitives by the born-digital parser, and 1,344 diagrams with errors produced in primitive extraction, alignment, and converting visual graphs to SMILES strings. This training dataset includes molecules represented by 32 unique symbol classes. A limitation that there are test set symbols that are missing in this training set. For the USPTO dataset 4 symbols are absent (`1, a, D, b`), from CLEF 26 symbols are absent (including `*, R, X, 0`), and from the UoB dataset 2 symbols are missing (`:, 0`).

Evaluation Metrics. For evaluating the performance of the chemical diagram recognition system using LCGP, we rely primarily on SMILES-based metrics. The primary benchmark is the *Exact SMILES Match* metric, which measures the accuracy of the recognized molecular structure by comparing the generated SMILES string with the ground truth string, both after canonicalization.

[4] `https://pubchem.ncbi.nlm.nih.gov`

This provides a direct measure of how accurately the chemical structures are reconstructed from raster images.

In addition, we perform error analysis using `confHist` on the graph representation, which helps identify and categorize common types of recognition errors in terms of symbol and relationship classifications within the visual structure graphs. However, it is important to note that graph-based evaluation metrics were not directly applied to the visual parser in this study. Instead, the emphasis was on understanding structural errors and classification performance in the context of chemical diagram recognition.

Implementation/Systems. The ChemScraper visual parser (LCGP) is implemented in Python using the `PyTorch`, `networkx` (graphs), `numpy`, `mr4mp` (map-reduce) libraries. visual parsing pipelines are Python-based, along with the visual line primitive extractor. Experiments for the visual parser were run on an Ubuntu 20.04 server with hard drives (HDD), an A40 (48GB) GPU, a 64-core Xeon Gold 6326 (2.9 GHz), and 256 GB RAM.

Benchmarking Chemical Parsers

For the synthetic USPTO dataset, our visual parser trained using outputs from our born-digital parser, obtains a recognition rate of 85.02%. While this rate is lower than that seen for transformer-based methods like MolScribe [100] and rule-based methods such as MolVec and OSRA [41], this result still demonstrates potential. Notably, MolScribe is trained on 1.68 million examples with various chemical structure-based and image-based augmentations, and employs a SWIN transformer model with 88 million parameters. In contrast, our visual parser was trained on a much smaller dataset of 3,416 annotated images, without augmentation, and using a simpler SE-ResNeXt model with 4 million parameters. Despite these differences, our parser outperforms SWIN-OCSR [146], which also uses a SWIN transformer but is trained on 4.5 million molecules.

We have omitted results for the real datasets (CLEF and UoB) due to limitations in our initial training dataset, which is missing symbols from these sets and it trained using a single set of Indigo rendering parameters as mentioned earlier. This first training set does not adequately capture the diverse styles and structural variations seen in the non-synthetic data sets. We will address this in future work. We will note here however, that the visual line primitives extracted from the real images are accurate.

We conducted training runs on the Pubchem dataset, which consisted of queries for 3,416 molecules

in three forms: primitives, whole symbols, and symbols detected during training. Each epoch averaged 155.6 minutes, with the model completing 19 epochs in about 49 hours. This training time is notably shorter than other systems, such as DECIMER [103], which required 27 days to converge on 15 million structures, demonstrating efficiency with fewer data to achieve comparable results.

However, testing on the synthetic USPTO dataset (5,719 molecules) took 18.6 hours (11.74 secs/-molecule), which is slower compared to systems like MolGrapher [79] and OCMR [134] that process a single molecule in less than a second. The slow inference time is due to inefficiencies in our first implementation. In particular, re-assembling query outputs for formulas and writing visual graphs are currently slower than they could be. Future versions will accelerate these components.

Error Analysis

For molecular diagrams produced by the visual parser for USPTO, symbols including different characters, numbers, and wedges are often misclassified as `Single` bonds. This is mainly due to class imbalance in the training data that predominantly features `Single` lines (roughly 70% of symbols in training are single lines). Errors also include incorrect segmentations, particularly for characters like `N, H` that are frequently over-segmented. This is also likely due to their rarity in the training data. Additionally, relationship errors, notably missed connections between lines and characters, are comparatively more common due to the predominance of line-line connections over line-character connections.

The class imbalance in symbols and relationships, especially the predominance of the `Single` class and line-line connections, highlights the need for better recognition of less frequent classes to improve the parser's performance on diverse molecular structures. Additionally, the training set does not include all symbols present in the test sets, which impacts the parser's ability to accurately recognize and interpret a full range of molecular symbols. Addressing this imbalance and coverage is important for future enhancements.

4.3 Summary

In this chapter, we introduced the Line-of-Sight with Graph Attention Parser (LGAP) for parsing mathematical formulas from typeset images and its extension, the Line-of-Sight Chemical Graph

Parser (LCGP) model for chemical diagrams. LGAP builds on and improves the MST-based QD-GGA parser through enhanced visual feature representations, incorporating line-of-sight neighborhoods and spatial pyramidal pooling (SPP) for better spatial context capture. Additionally, we modified ground truth representations of spatial relationships connecting punctuation with parent symbols in Symbol Layout Trees (SLTs). These enhancements enabled a more comprehensive context representation, minimized spatial information loss compared to single-region average pooling used in QD-GGA, and preserved valid punctuation relationships that might otherwise have been pruned.

The LCGP model extends LGAP to parse chemical diagrams from raster images, leveraging annotated visual graph data from our born-digital parser for robust training. LCGP enhances feature representation by directly drawing line primitives into fixed-size 28×28 binary images, improving focus and reducing noise. Tightly cropped context images provide more precise visual context, while a recurrent multi-task neural network iteratively refines symbol segmentation and classification, handling over-segmentation more effectively. The model outputs molecular structure graphs in CDXML, compatible with tools like ChemDraw and Marvin, and easily convertible to other formats such as SMILES, MOL, and InChI. These advancements make LCGP more adaptable to complex chemical structure recognition and a versatile framework for broader visual parsing tasks.

Our experiments demonstrated the effectiveness of LGAP and LCGP in addressing complex visual parsing tasks for both math and chemical structures. LGAP showed improvements in parsing mathematical formulas with enhanced context usage, while LCGP demonstrated potential in parsing chemical diagrams using graph-based representations. However, limitations remain, including handling noisy images and the need for improved generalization of the visual parser for real-world data.

Limitations of our approach include:

1. The images evaluated were limited to relatively clean vector and rasterized-vector images generated from a single rendering model (Indigo). Real-world applications, including noisy scanned documents, would require adaptations in primitive extraction, annotation strategies, and parser designs.

2. The initial implementation of the visual parser for chemical diagrams is slower in inference and struggles with generalization due to limited class coverage and variability in the training dataset.

In the next chapter (Chapter 5), we address **RQ2** and **RQ3** by systematically studying the effect of input graph representations and contextual interaction mechanisms in visual parsing. We compare different graph representations across math and chemical domains, analyzing trade-offs in edge recall, precision, and expression coverage. We also introduce different improvements in visual features, including a better backbone encoder architecture. Finally, we develop an edge-aware graph attention mechanism (EGATv2) that integrates multi-hop message passing and task-level interaction, enabling joint updates of node and edge representations for improved parsing accuracy.

Chapter 5

Input Graph Representations and Context

Accurate parsing of structured visual notations such as mathematical formulas and chemical diagrams requires understanding not only the appearance of individual visual primitives but also their spatial and relational context. In graph-based parsing frameworks, this context is modeled via a structured input graph, where nodes represent over-segmented visual primitives and edges represent hypothesized relationships. The design of this input graph is critical: it constrains the space of learnable structures and determines the scope of relational reasoning the model can perform.

In this chapter, we address two central research questions related to graph-based input and context modeling. First, we address **RQ2**, which investigates the effectiveness of different input graph representations in enabling accurate parsing for both math and chemical domains. Prior work by Hu et al. [51] emphasized the importance of achieving high edge recall while maintaining low graph density to balance structural completeness and learning efficiency. Motivated by this, we evaluate multiple graph topologies including complete graphs, line-of-sight (LOS) graphs [52], and k-nearest neighbor (KNN) graphs [39], and propose domain-specific graph representations that attain a balance between high expression coverage and sparsity. We also partly address **RQ1** by introducing an efficient and robust method for extracting visual line primitives from raster chemical diagrams, which effectively separates over-segmented components.

Second, we address **RQ3**, which studies the role of attention-based mechanisms and intermediate predictions in modeling contextual dependencies across tasks. Inspired by Mahdavi et al. [72],

who discussed the possible benefits of using task interaction through class distribution vectors, we propose a two-stage attention model that incorporates classification outputs with visual features into graph attention to improve cross-task learning. To further enhance context aggregation, we introduce a novel edge-aware graph attention architecture based on GATv2 [17], which supports multi-hop message passing and jointly updates both node and edge embeddings.

5.1 Input Graph Representations (RQ2)

In our graph-based parsing framework, structure recognition is decomposed into three interrelated stages: segmenting primitives into symbols, selecting edges to form a symbol layout graph or tree, and classifying both the edges and associated symbol nodes. These stages are not strictly sequential; they influence each other during training through shared representations and recurrent message passing. Consequently, the design of the input graph representation becomes a critical factor in the overall parsing performance. Importantly, the input graph serves as a hypothesis space for possible relationships and defines which edges the model can classify. This graph is fixed before learning and determines the connectivity constraints for downstream edge and node classification.

A well-constructed input graph should therefore contain all ground-truth edges (or as close to 100% recall as possible), ensuring that the model is not structurally limited [51]. Near-perfect recall is sufficient because extra edges can be corrected by learning to classify it as a `NoRelation` edge. But missing edges are not recoverable, and they represent structural gaps that the model cannot fix. At the same time, the number of invalid or extra edges should be minimized to avoid overwhelming the model with irrelevant options, which increases computational cost and class imbalance during learning.

5.1.1 Types of Graph Representations

To balance the concerns above, we investigate *strategic pruning* of edges, not present in the ground truth graphs to reduce variance and improve computational efficiency while maintaining high recall. Our goal is to identify graph types that maximize recall of true edges in the ground truth while enabling effective learning. In both domains, each graph is constructed over a set of visual primitives, which serve as the graph nodes. For mathematical formulas, primitives correspond to connected components (CCs) obtained from binarized formula images. For chemical diagrams, primitives are the over-segmented visual elements extracted through contour and line segment detection, as

described in Section 3.4.

For analysis and evaluation, we operate on symbol-level nodes, as they are easier to interpret and understand when comparing different graph representations. We consider three types of input graphs: Complete, Line-of-Sight (LOS), and K-Nearest Neighbor (KNN), described below.

Complete Graphs. While complete graphs trivially ensure perfect recall by connecting every pair of nodes, they introduce an overwhelming number of invalid edges. For an expression with N symbols, a complete graph contains $N(N-1)$ directed edges. In contrast, the ground truth edges for mathematical expressions, represented as trees in our parser, are limited to $N-1$ edges between the symbol objects. This results in a precision that is at best $1/N$ [51]. In the case of chemical diagrams, the average number of ground truth edges is higher, $2.5N$ on an average in our PubChem-5k dataset, but still significantly lower than the number of edges in a complete graph. This yields an average precision of roughly $\frac{2.5}{N-1}$.

For typical expressions with more than 10 symbols, these values correspond to a precision below 10% for math formulas and below 28% for chemical diagrams. Such low precision hampers both inference and learning. The parser must evaluate a large number of irrelevant edges, increasing computational cost, and must also learn under conditions of extreme class imbalance, where valid edges are vastly outnumbered by invalid ones.

Line-of-Sight (LOS) Graphs. Line-of-sight (LOS) graphs select edges where visual primitives (e.g., strokes, connected components, or contours) are mutually visible without obstruction [52, 71, 72, 114]. This constraint has been widely used in math expression parsers such as Hu et al. [50], LPGA [71], and QD-GGA [72]. In these models, the final interpretation is selected as a maximum spanning tree (MST) over the LOS graph hypotheses as described in Chapters 3 and 4.

On average, LOS graphs contain approximately $4.5N$ directed edges per expression in the InftyMCDB-2 (math) dataset and $14.8N$ directed edges per expression in the PubChem-5k (chemistry) dataset, where N is the number of symbols. This leads to an approximate edge precision of $\frac{N-1}{4.5N}$ for math formulas and $\frac{2.5N}{14.8N} \approx 0.169$ for chemical diagrams. For typical expressions with more than 10 symbols, this results in a precision of roughly 20% in the math domain and 17% in the chemical domain.

K-Nearest Neighbor (KNN) Graphs. To balance completeness and computation, we pruned the complete graph using a k-nearest neighbor strategy, ensuring coverage of all ground truth edges while reducing edge count. In a K-Nearest Neighbor (KNN) graph [39], each visual primitive is connected to its K closest neighboring primitives based on spatial distance. The spatial distance is defined by the nearest end-point distances between the primitives. This structure ensures strong local connectivity while keeping the overall graph sparse and computationally manageable.

For KNN graphs, we evaluate $K = 2$ and $K = 6$: lower values like $K = 2$ emphasize highly local structure, while higher values such as $K = 6$ help capture ground-truth edges between primitives that are slightly farther apart. These settings allow us to examine the trade-off between graph sparsity and expression coverage, where all ground truth edges are present in a graph.

For the chemical domain (PubChem-5k), 2NN graphs yield approximately $2.8N$ directed edges per expression and achieve a edge precision of around $\frac{2.5N}{2.8N} \approx 0.89$. Increasing K to 6, the number of edges increase to about $6N$ per expression, causing the precision to drop to around $\frac{2.5N}{6N} \approx 0.42$. In the mathematical domain (InftyMCDB-2), 2NN graphs similarly result in about $2.3N$ edges per expression and has a precision of $\frac{N-1}{2.3N} \approx 0.39$ for $N = 10$ symbols. 6NN graphs expand the edge set to approximately $5.2N$ edges per expression, and precision drops to $\frac{N-1}{5.2N} \approx 0.17$ for $N = 10$ symbols.

5.1.2 Comparison and Analysis

We compare the three graph representations on both mathematical formulas and chemical diagrams using our PubChem-5k dataset [112] for chemical diagrams and the InftyMCDB-2 dataset [114] for math formulas.

We evaluate each graph type based on three criteria: edge recall, edge precision, edge F1 and expression coverage rate. These metrics jointly characterize the suitability of a graph representation by balancing ground-truth structural coverage with input sparsity and complexity. All graph representations use the same set of ground-truth nodes and their corresponding symbol labels. This analysis focuses exclusively on the effect of graph connectivity (i.e., edge sets).

Edge recall is the proportion of ground-truth edges that are present in the input graph. It is computed across all math expressions or chemical diagrams in the dataset. High recall is essential to avoid unrecoverable parsing failures, since missing edges cannot be reconstructed in later stages

in our parser.

$$\text{Edge Recall } \% = \frac{\text{\# ground-truth edges in the input graph}}{\text{\# edges in the ground truth graph}} \times 100$$

Edge precision is the proportion of edges in the input graph belonging to true relationships. Maintaining high precision reduces the number of invalid edges, lowering parser complexity and mitigating the effects of class imbalance during edge classification.

$$\text{Edge Precision } \% = \frac{\text{\# ground-truth edges in the input graph}}{\text{\# edges in the input graph}} \times 100$$

Edge F1 score is the harmonic mean of edge precision and recall, and provides a balanced measure that penalizes both missed edges and false positives:

$$\text{Edge F1 } \% = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

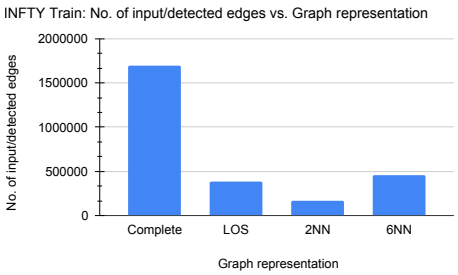
The harmonic mean prefers precision and recall values that are close to each other, with the F1 score reaching its maximum when precision equals recall.

Expression coverage rate. This measures the percentage of math expressions or chemical diagrams in the dataset in which all ground-truth edges are present in the input graph. This reflects the upper bound on correct expressions under perfect node and edge classification for the input graph representation, and serves as a global indicator of structural completeness in the input graph.

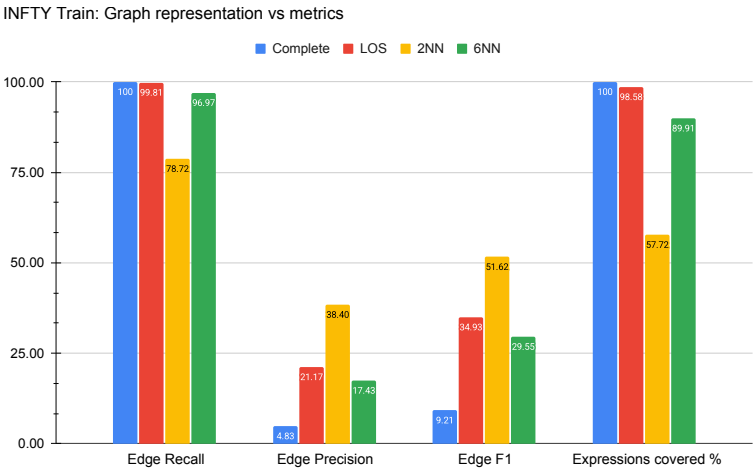
$$\text{Expression Coverage Rate } \% = \frac{\text{\# expressions where all ground-truth edges are present}}{\text{\# total expressions in the dataset}} \times 100$$

Analysis. Figure 5.1 present a comparative evaluation of input graph representations—Complete, LOS, 2NN, and 6NN—on the InftyMCDB-2 (math) and PubChem-5k (chemistry) training sets. Each representation is assessed using the metrics: edge recall, precision, F1 score, and expression coverage rate, along with the total number of candidate input edges.

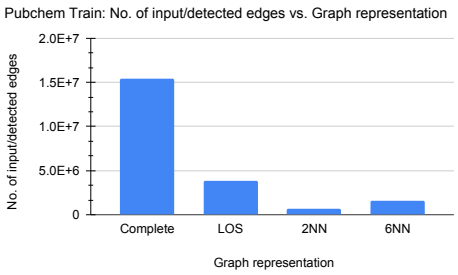
Math formulas. For mathematical expressions, the LOS graph offers the best trade-off among the evaluated representations. As shown in Figure 5.1b, it achieves near-perfect edge recall (99.66%), high expression coverage (98.58%), and much higher precision (21.17%) than the complete graph (4.83%). In contrast, while the complete graph achieves perfect recall and expression coverage,



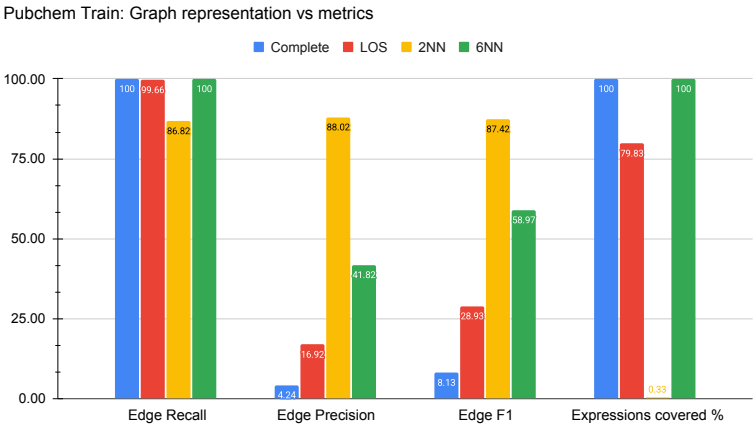
(a)



(b)



(c)



(d)

Figure 5.1: Comparison of input graph representations on the InftyMCDB-2 training set (a-b) for math formulas and Pubchem-5k training set (c-d) for chemical diagrams. (a), (c): Number of input (candidate) edges generated by each graph representation. Complete graphs produce the highest number of edges, and 2NN graphs produce the lowest. (b), (d): Evaluation of each graph representation using edge recall, precision, F1, and expression coverage rate.

it suffers from extremely low precision due to an excessive number of candidate edges (over 1.7 million), as shown in Figure 5.1a, leading to increased computational cost and class imbalance during training.

Compared to the KNN-based alternatives, the LOS graph also outperforms both 2NN and 6NN. The 2NN graph achieves a recall of 78.72%, precision of 38.40%, F1 score of 51.62%, and expression coverage rate of only 57.72%. While the 6NN graph improves recall to 96.97% and expression

coverage to 89.91%, its precision drops to 17.43%, still lower than LOS, indicating a higher number of spurious edges. 6NN graph uses slightly more total edges than the LOS graph, while still missing long-range relationships essential to parsing mathematical structures. Overall, LOS achieves a more favorable balance between coverage, recall, and precision for the math domain.

The suitability of LOS graphs for math stems from the nature of mathematical notation. Math expressions often contain a wide variety of spatial relationships, such as `Above, Below, Inside, Subscript`, and `Superscript` and include many long-range dependencies, especially in constructs like fractions, square roots, and nested structures. These spatial layouts are not always well captured by fixed-neighborhood KNN graphs, even with $K = 6$ primitives. In contrast, LOS graphs preserve semantically meaningful spatial visibility relationships across variable distances, enabling more accurate parsing of structurally complex formulas with fewer candidate edges. This is particularly beneficial given the largely horizontal arrangement of math expressions, where key relationships (e.g., between base symbols and superscripts or between operators and operands) often span long horizontal or diagonal distances that are well captured by LOS.

Chemical diagrams. For chemical diagrams, the 6NN graph representation provides the most effective and efficient structure. As seen in Figure 5.1d, it achieves perfect edge recall (100%) and expression coverage rate (100%), with a substantially higher precision (41.82%) than the complete (4.24%) and LOS (16.92%) graphs. It also generates fewer input edges than both LOS and complete graphs, as seen in Figure 5.1c, making it more computationally efficient and less prone to false positives. The 2NN graph achieves a lower recall of 86.82%, a high precision of 88.02%, F1 score of 87.42%, but expression coverage rate of only 0.33%. Although the 2NN graph has the highest precision, its low recall and poor expression coverage indicates many valid edges are missing.

Chemical diagrams are inherently local in structure: edges in our parser represent bonds or annotations between adjacent atom line primitives, with no long-range spatial relationships akin to math. These connections are almost always between the nearest primitives, constrained by chemical structure rules. Using 2NN graphs results in missed edges, since an atom can have up to four bonds, and oversegmentation may split atoms into multiple line primitives, requiring a higher K to ensure all necessary edges are present. The 6NN graph compensates for these omissions by providing a broader local neighborhood, ensuring that chemically relevant connections, such as multi-bonded atoms or split primitives are preserved while still controlling the number of candidate edges.

Chosen representations. We use LOS input graph representations for math due to their ability to model long-range and semantically meaningful spatial relationships with fewer spurious edges. For chemistry, 6NN graphs offer the best balance of recall, precision, and input size by exploiting the local nature of chemical structure diagrams. These choices align with domain-specific characteristics and support the structural requirements of the parser.

5.1.3 Edge Type for Atom Number Annotations in Chemical Diagrams

Chemical diagrams often include atom number annotations that, in isolation, appear visually similar to atom symbols, particularly digits. While these annotations appear similar to digits in the image, they serve a different semantic role: indicating the atomic number of a particular atom within a molecule. In the output chemical graph format (`CDXML`), such annotations are not represented as independent nodes but are instead attributes associated with the parent atom nodes. Similarly, the output `SMILES` string does not include atom numbers, since they are not part of the molecular structure itself.

As a result, the visual parser must distinguish between digit symbols that represent atoms and those that function as atom number annotations, so that post-processing can correctly incorporate this information into the final graph representation. To support this, we introduce a new edge type to explicitly encode the annotation relationship. Atom number annotations are represented via a dedicated edge class, denoted as `ANNOTATION`, which links the digit node to the corresponding parent atom node. The edge classification task is updated to distinguish among three edge types: `CONNECTION`, `NoRelation`, and `ANNOTATION`.

5.2 Graph Attention and Task Interaction (RQ3)

This section addresses **RQ3**, which investigates whether incorporating edge-aware attention mechanisms can improve graph-based parsing performance in visual recognition tasks. In Section 5.2.2, we first describe general improvements to the visual feature extraction pipeline that enhance the quality of node and edge representations prior to graph-based reasoning. These refinements are applicable across models and are not specific to graph attention. Notably, the improvements to visual primitives also contribute toward **RQ1**, by providing more robust and consistent input representations across math and chemical domains.

We then introduce our edge-aware graph attention mechanism in Section 5.2.3, which extends a standard GATv2 architecture to jointly update node and edge features using shared attention mechanisms. Finally, in Section 5.2.4, we present a two-stage task interaction framework that leverages intermediate classification outputs to guide further refinement through a second round of graph attention, promoting information flow across tasks such as symbol classification, segmentation, and relationship detection.

5.2.1 Common Feature Extraction Pipeline

In our earlier models, LGAP and LCGP used different strategies for visual feature construction and multi-task CNN encoding, tailored to their respective target domains—math for LGAP and chemical diagrams for LCGP (see Chapter 4). In LGAP, visual features were extracted from full formula images using binary attention masks derived from connected component (CC) contours and line-of-sight (LOS) neighborhoods. These binary masks were applied to a global CNN feature map to isolate regions corresponding to query primitives or primitive pairs. However, this approach often suffered from sparsity in the masks and inefficiencies due to large image sizes relative to localized regions of interest.

In contrast, LCGP introduced several improvements by shifting to a contour-based representation in which each primitive is drawn directly onto a fixed-size, resolution-normalized canvas. This localized formulation reduces background noise, improves feature compactness, and enables independent processing of queries using a shared SE-ResNeXt CNN encoder.

Building on the improvements introduced in LCGP, we now adopt a unified visual feature extraction pipeline for both math and chemical domains. In our datasets, while chemical diagrams already include contour-based primitives, mathematical formulas (InftyMCDB-2 dataset) provide connected components (CCs) as the initial visual primitives. We extract contours from each CC in the math dataset, aligning its representation with that of the chemical domain. Then, the same contour-based, resolution-normalized primitive representations are used to construct features for both query and context primitives, and these are passed through a shared multi-task CNN encoder backbone. This unification eliminates the dependency on binary masking and large global crops used in LGAP, simplifying the pipeline while improving consistency and data efficiency. With this change, the underlying model and learning process become domain-agnostic, and the only remaining differences across tasks are the input graph representations and domain-specific postprocessing steps, such as maximum spanning tree (MST) extraction for math formulas and conversion of visual graph to chemical graph for chemical diagrams.

5.2.2 Visual Feature Improvements (RQ1)

To improve the quality of visual features used in our models, we adopted several refinements to the extraction pipeline introduced in Chapter 4. These refinements include a more robust primitive extraction method using the Line Segment Detector (LSD) for chemical diagrams, adoption of a stronger visual backbone (ResNeSt [159]), evaluation of different input feature resolutions, and tuning of spatial pooling configurations. Together, these changes provide a stronger foundation for subsequent graph-based reasoning and multi-task learning.

Visual Primitives Refinement via Line Segment Detector (LSD)

Visual parsing of chemical diagrams and math formulas from raster images in real documents presents unique challenges due to noisy rendering, over-segmented primitives, and overlapping components. Our visual parser model can handle over-segmentation by merging primitives, but cannot recover from under-segmentation. Under-segmentation occurs when distinct visual primitives are merged into a single component (e.g., oxygen atom touching a bond line as shown in Figure 5.2), forcing the model to identify them as a single entity, introducing errors. Our visual primitive extraction method, discussed in Chapter 3 uses polygon simplification, skeletal lines and geometric transformations, which is effective in simplifying contours, but prone to the under-segmentation issue. To resolve this, we introduce a refined primitive extraction method that combines Line Segment Detection (LSD) [131], geometric filtering, connected components analysis, and watershed segmentation [130].

The Line Segment Detection (LSD) algorithm. LSD is a subpixel-accurate technique that detects line segments by grouping aligned gradient responses in images. It is grounded in the Helmholtz principle [31, 32], which assumes that structures (like lines) are perceptually meaningful only if their presence is unlikely to occur by chance. LSD formalizes this through the a contrario validation framework, which quantifies how surprising a detected structure is under a background noise model, allowing the method to control false detections without extensive threshold tuning.

Watershed Segmentation. To convert line detections into segmented primitive regions, we apply the *watershed algorithm* based on the immersion simulation technique introduced by Vincent and Soille [130]. In this method, grayscale images are treated as topographic surfaces, and segmentation is simulated by immersing the surface in water: regional minima act as catchment basins (*markers*)

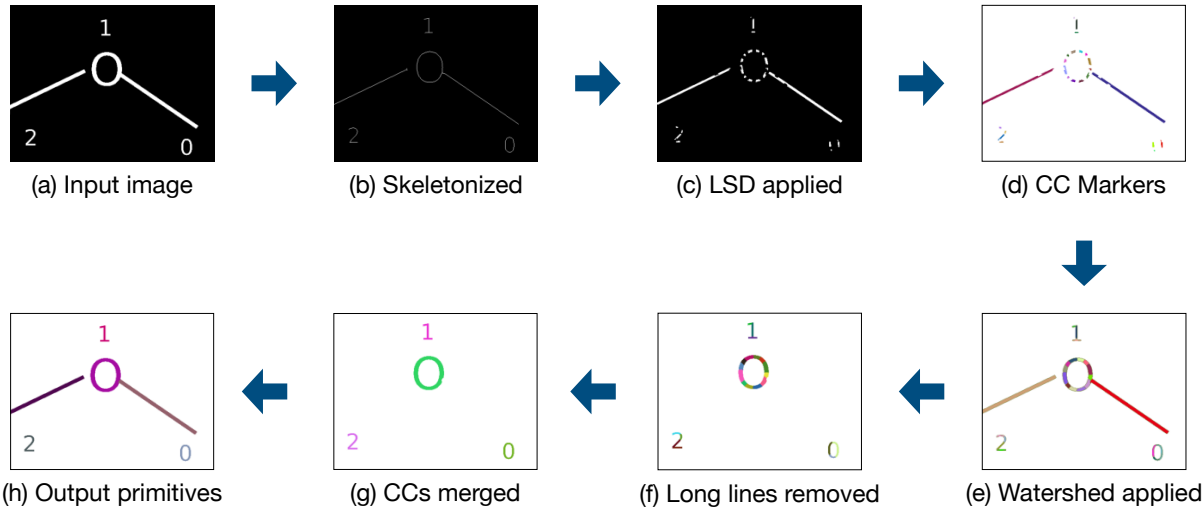


Figure 5.2: Step-by-step illustration of the LSD-based visual primitive extraction pipeline in Algorithm 1. (a) Input binary image I with 5 CCs, (b) Skeletonization produces medial axis lines $S \leftarrow \text{Skeletonize}(I)$, (c) LSD is applied on the skeleton: $L_{\text{raw}} \leftarrow \text{LSD}(S)$ (d) CC analysis generates watershed markers M, (e) Watershed algorithm is applied using these markers on the input grayscale image: $W \leftarrow \text{Watershed}(I_{\text{gray}}, M)$, (f) Regions corresponding to long lines (identified via geometric thresholding) are removed from W, (g) Over-segmented short fragments are merged using a second CC analysis to get W', (h) Long lines are reintroduced as distinct labeled regions to obtain the final set of 6 primitives, $P = W' \cup \mathcal{L}_{\text{long}}$, where different colors indicate different primitive regions.

that *flood* outward, with boundaries (watershed lines) forming where water from distinct basins meets. We use this to grow or fill regions from seed points for line segments detected by LSD.

Primitive Extraction Algorithm. The complete pipeline for the visual primitive extraction is described in Algorithm 1. We first apply skeletonization [164] to the input image, which removes pixels from object borders in iterative passes until only thin, one-pixel-wide structures remain, preserving the shape topology of the original objects. Then LSD is applied to the output medial axis structures (skseletonized image). Each detected line is shortened to break connections between closely spaced endpoints, ensuring clean separation for connected component (CC) analysis, required for watershed marker generation. These shortened lines are drawn onto a blank canvas, and connected components are computed to serve as markers for watershed segmentation. The watershed transform, based on immersion simulation [130], propagates labels from these markers across the binary image gradient surface, resulting in oversegmented primitive regions.

Algorithm 1 Primitive Extraction using LSD and Watershed Segmentation

Input: Binary input image I of size $H \times W$

Output: Segmented primitive map P

(Skeletonization) Compute skeleton image $S \leftarrow \text{Skeletonize}(I)$
(Line Detection via LSD) Apply LSD to S to obtain line segments $L_{\text{raw}} \leftarrow \text{LSD}(S)$
(Preprocessing: Line Shortening and Long Line Identification)
    (i) Initialize empty list of shortened lines $L \leftarrow \emptyset$ and set of long line midpoints $\mathcal{L}_{\text{long}} \leftarrow \emptyset$
    (ii) Compute image diagonal $D \leftarrow \sqrt{H^2 + W^2}$ and threshold $\tau \leftarrow 0.01 \cdot D$
    (iii) Shorten lines and add long line midpoints:
        for each raw line $l \in L_{\text{raw}}$ **do**
            Represent l as segment with endpoints (x_1, y_1) and (x_2, y_2)
            Shorten l at both ends by 5 pixels to get l'
            Append l' to L
            if length$(l') \geq \tau$ **then**
                Add midpoint m_l of l' to $\mathcal{L}_{\text{long}}$

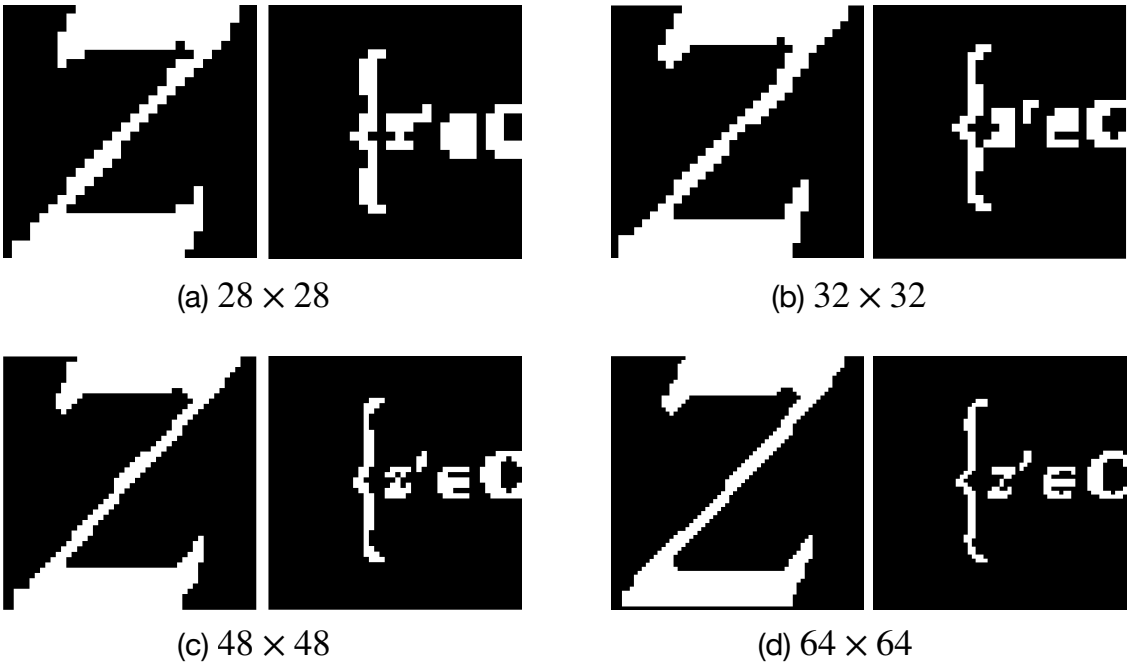
(Marker Generation) Generate markers for watershed segmentation
    (i) Initialize blank canvas C of same size as I
    (ii) Draw all shortened lines L onto C
    (iii) Compute CCs on C to obtain marker map M with unique integer labels

(Watershed Segmentation)
    (i) Convert I to 3-channel grayscale image I_{gray}
    (ii) Apply watershed segmentation: $W \leftarrow \text{Watershed}(I_{\text{gray}}, M)$

(Refine Oversegmentations and Final Map Construction)
    (i) Remove long lines from W to isolate only oversegmented fragments:
        for each $m_l \in \mathcal{L}_{\text{long}}$ **do**
            Lookup the watershed label at the midpoint: $id_l \leftarrow W(m_l)$
            Remove all pixels in W assigned to id_l
    (ii) Apply connected components to the updated W to obtain W'
    (iii) Draw long lines from $\mathcal{L}_{\text{long}}$ into W' with unique new region IDs

return Final primitives $P = W' \cup \mathcal{L}_{\text{long}}$

To refine this oversegmentation, long lines are detected using a relative length threshold based on



(a) 28×28 (b) 32×32

(c) 48×48 (d) 64×64

Figure 5.3: Examples of query (left) and context (right) features (centered around query) rendered at different input resolutions. At lower resolutions, (a) 28×28), primitives in context features become difficult to identify visually. As resolution increases up to (d) 64×64), the primitives become more distinct.

the diagonal of the image (see Algortihm 1) and removed from the initial watershed map. Connected components are then applied to merge fragmented glyphs, and the removed long lines are added back as distinct labeled regions. As illustrated in Figure 5.2, this refinement enhances parsing inputs by addressing both over- and under-segmentation. It preserves all structurally relevant primitives, including touching or overlapping strokes. This method can also be applied to typeset or handwritten mathematical formulas to obtain consistent visual primitives for parsing; however, we leave that for future work.

We examine the effect of two architectural components on parsing accuracy and robustness: the size of input feature patches extracted for each primitive, and the configuration of spatial pyramidal pooling (SPP) used within the feature extraction backbone.

Input Feature Size and Spatial Pooling

Input feature size: query and context features. We conduct a grid search on the input resolution of the cropped query and context images to evaluate how feature size impacts parsing performance. This fixed input size determines the level of spatial and contextual information preserved in the representation. Smaller input sizes (e.g., 28×28) reduce memory usage and allow efficient batch processing, but they limit the spatial resolution available to encode fine-grained details (see Figure 5.3). This issue becomes especially prominent in context features, where the canvas includes both the query and its $k = 6$ nearest neighbors. At low resolutions, the contours of neighboring primitives often overlap or become visually compressed, making their structure indistinct. This impairs the model's ability to distinguish between classes that depend on relative positioning or local patterns.

Additionally, smaller feature maps may lose surrounding context that is useful for differentiating visually similar symbols. For example, short line segments can be confused with the digit '1', and adjacent bond lines may occlude or distort atom symbols in chemical diagrams. Conversely, increasing the feature size provides more pixels per contour, reduces overlap in context patches, and allows the network to retain spatial cues that aid in classification under both clean and noisy conditions. However, larger input sizes also introduce trade-offs as they increase computational and memory overhead.

Number of spatial pyramidal pooling regions. As described in Section 4.1.3 in Chapter 4, to enhance spatial generalization and facilitate scale-invariant feature learning, we incorporate a Spatial Pyramidal Pooling (SPP) layer [44] after the final convolutional feature map. SPP aggregates features over multiple spatial subdivisions of the input, enabling the network to encode both fine-grained and global spatial structure without requiring a fixed input size for the entire network.

One of the motivations for using SPP was to address confusion between visually similar symbols that differ in orientation or spatial arrangement. For example, symbols such as `W` and `M`, or `j` and `i`, can appear visually ambiguous when their relative spatial features are not well preserved in the feature representation. By pooling features across structured horizontal and vertical regions, SPP helps retain positional information and provides a spatial signature that improves discrimination between such cases. This spatial encoding is especially beneficial in cluttered diagrams, where primitives often appear in close proximity, and subtle orientation or layout differences are important for correct classification.

In our previous models discussed in Section 4.1.3, for LGAP, we used 5 levels with 11 regions in pooling outputs: this included 1 full feature map, 2 vertical bins, 3 vertical bins, 2 horizontal bins, and 3 horizontal bins. For LCGP, feature maps were average pooled in 7 pyramidal regions (image, 3 vertical, 3 horizontal). For the new models, we further explore additional SPP configurations with different numbers of pooling regions, described in Section 5.3.3.

Split-Attention (ResNeSt) Backbone

As described in Section 4.1.3, the LCGP parser uses a shared CNN encoder to extract visual features from contour-based image patches centered on query and context primitives. In our models, we employed a SE-ResNeXt-50 encoder [49], which had previously proven effective in formula parsing tasks. This architecture builds upon a sequence of advancements: ResNet [46] introduced residual connections to improve gradient flow in deep networks; ResNeXt [138] added the concept of *cardinality* by dividing convolutional transformations into multiple parallel branches to improve representational power without significantly increasing computational cost; and SE-ResNeXt [49] extended this by adding Squeeze-and-Excitation (SE) blocks that adaptively reweight channel responses based on global context.

While SE-ResNeXt successfully models what feature channels are important, it still lacks the ability to model where in space those features occur, limiting its effectiveness in tasks involving dense local structure, such as symbol parsing or distinguishing between closely spaced primitives. This motivated our transition to a more expressive backbone, ResNeSt [159]. It adds spatial awareness to channel attention through *split-attention blocks* that extend the capabilities of SE-ResNeXt by incorporating attention across channels as well as across multiple spatial partitions within each group of convolutions. This allows the network to learn both *what* (channel attention) and *where* (spatial attention within groups) to emphasize. This makes it particularly suitable for structured visual recognition tasks such as symbol classification, segmentation, and relationship detection in both mathematical and chemical diagrams.

Adaptation for Primitive-Centric Inputs. The original ResNeSt-50 model was designed for full-scale RGB images of size 224×224, which is excessive for the small grayscale, binary contour patches used in our parsing framework (32×32 or 64×64). Moreover, aggressive downsampling and high channel count in the initial layers in the original architecture can lead to loss of spatial resolution and detail, which are critical for parsing tasks. To make ResNeSt compatible with our task, we implement several architecture-level adaptations:

- **Reduced output dimensionality:** The number of output channels is reduced from 1024 to 32 to match the capacity of prior SE-ResNeXt backbones used in formula parsing and to reduce parameter count.
- **Removed max pooling:** Max pooling operations in early layers are removed to preserve spatial resolution, preventing excessive downsampling of small input patches.
- **Reduced convolution stride:** Stride of 1 is used in the initial convolution layers (instead of 2) to maintain spatial granularity.

Efficiency and Representational Capacity. Despite its architectural enhancements, the modified ResNeSt remains lightweight. As shown in Table 5.1, our reduced ResNeSt variant contains approximately 64% of the parameters of SE-ResNeXt-50, nearly half the size, but can offer improved expressiveness through its combination of spatial and channel attention mechanisms. We also leverage pretrained weights for the ResNeSt layers, which further enhances learning efficiency and convergence stability.

Table 5.1: Model parameter counts for visual backbone encoders.

Model	**# Trainable Parameters**
SE-ResNeXt-50 (baseline)	4,052,969
ResNeSt-50 (full)	27,373,896
ResNeSt-50 (reduced)	2,623,785

5.2.3 Edge-Aware Graph Attention with Multi-Hop Message Passing

Context plays a critical role in parsing structured visual notations, as the interpretation of symbols and their relationships depends not only on individual features but also on surrounding elements. In both mathematical and chemical domains, spatial arrangements and local structural patterns are essential for accurately classifying symbols and identifying their relationships. To support structural understanding in these diagrams, it is necessary to contextualize each symbol or relationship using both its visual appearance and its position within the surrounding graph structure. In earlier approaches, node and edge features were processed independently, limiting the model's ability to share contextual information across related components.

As discussed in Chapter 2, Graph Attention Networks (GATs) [129] offer a mechanism for incorporating local structure into node embeddings by computing attention weights between a node and

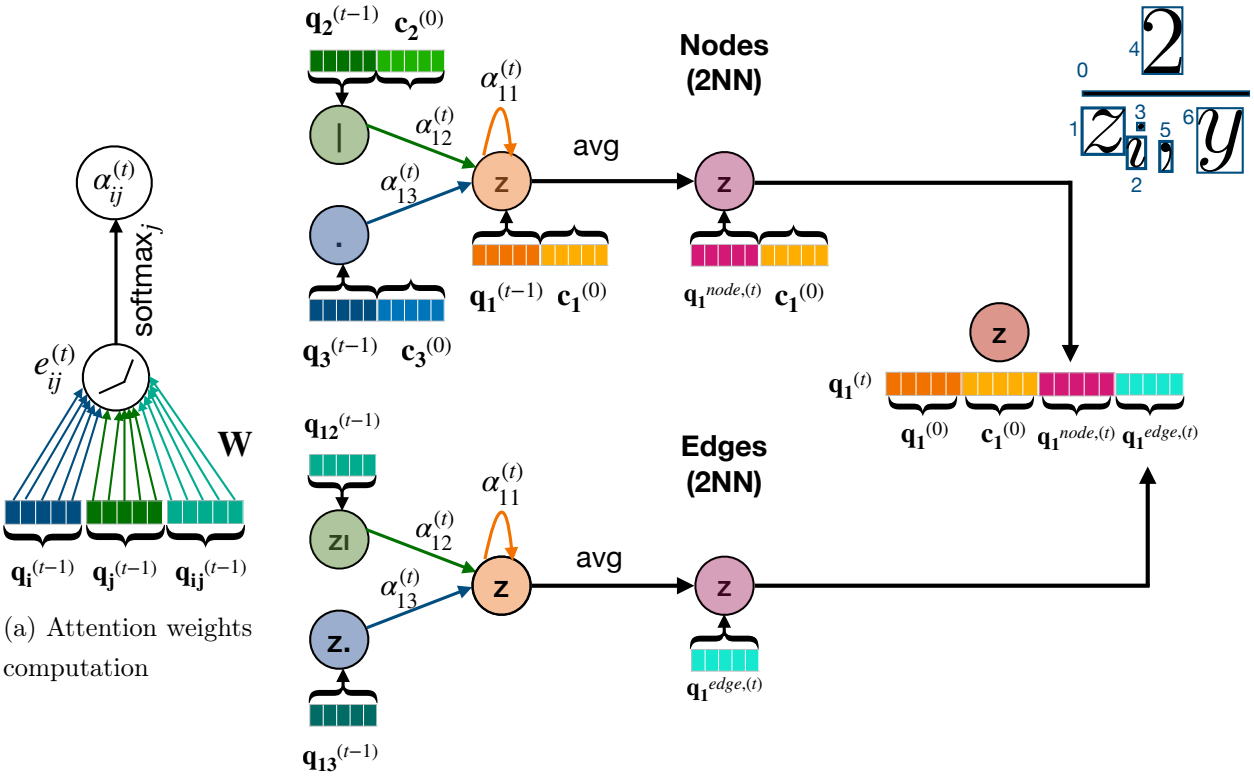
its neighbors based on their feature similarity. This enables adaptive and data-driven aggregation of information from a node's neighborhood, enhancing local contextualization. However, standard GAT architectures are limited in that they update only node representations using neighboring node features, while completely ignoring edge features. This design is suboptimal for our multi-task learning (MTL) setting, where edge-level predictions, such as spatial relationships in mathematical formulas or chemical bonds and annotations in molecular diagrams, are primary objectives. A more expressive architecture is needed that utilizes both nodes and edges as features to aggregate in the graph and allows information to flow between them during message passing.

As discussed in Section 2.3.3, recent models such as Graphormer [148], Relational Transformer (RT) [33], and the stroke-level graph model by Xie et al. [139] demonstrate how edge features can be effectively integrated into attention mechanisms. These methods encode edge information either through explicit edge embeddings or by concatenating edge and node features within the attention computation. This enables richer structural encoding by capturing not only node-level interactions but also relational attributes such as connectivity and type.

Building on these insights, we propose a modified message-passing architecture based on GATv2 [17] that incorporates edge features and supports joint node-edge interaction across multiple hops. Here, multiple hops refer to iterative message-passing steps in which each node and edge representation is updated by aggregating information not only from immediate neighbors but also from their extended neighborhood over successive layers, enabling the model to capture higher-order structural dependencies in the graph. GATv2 enhances the original GAT formulation by making attention computation dependent on both source and target node features, enabling more expressive contextualization while maintaining computational efficiency, as outlined in Section 2.3.4.

We extend this formulation to support bidirectional communication between nodes and edges. Specifically, we augment the node update steps with edge-aware attention, and introduce a parallel edge update module in which edge embeddings are refined based on the representations of their incident nodes.

Design. We refer to our adapted GATv2-based framework as *Edge Graph Attention Network v2 (EGATv2)*. The EGATv2 module is applied at the end of the visual feature extraction pipeline, immediately prior to task-specific classification heads. Input features for each node ($\mathbf{f}_i$) and edge ($\mathbf{f}_{ij}$) are formed by pooling the outputs of the ResNeSt encoder and concatenating query ($\mathbf{q}$) and



(a) Attention weights computation

(b) Node attention and feature aggregation

Figure 5.4: EGATv2 mechanism for nodes. (a) Edge-aware attention mechanism: attention scores $\alpha_{ij}^{(t)}$ are computed from a concatenation of node and edge query features $(\mathbf{q}_i^{(t-1)}, \mathbf{q}_j^{(t-1)}, \mathbf{q}_{ij}^{(t-1)})$, projected through a learnable matrix $\mathbf{W}$ and passed through a LeakyReLU nonlinearity. Softmax normalization is performed over the two nearest neighbors (2NN) of each node: nodes i and j and the edge ij. (b) Multi-hop node feature update for node 1 (`'z'`): aggregates attention-weighted messages from neighbors (top row): node 2 (base of `'i'`) and node 3 (dot of `'i'`), including itself (node 1), and the two corresponding edges (bottom row): z → base of `'i'` (12) and z → dot of `'i'` (13), and itself (11) update its representation based on its attention score. After aggregation, aggregated node and edge query features are combined with original query and context features via concatenation to form the updated node embedding $\mathbf{q}_1^{(t)}$ for node 1 (`'z'`).

context ($\mathbf{c}$) features:

$$\mathbf{f}_i = \mathbf{q}_i^{(0)} \oplus \mathbf{c}_i^{(0)}, \qquad \mathbf{q}_i^{(0)}, \mathbf{c}_i^{(0)} \in \mathbb{R}^F \tag{5.1}$$

$$\mathbf{f}_{ij} = \mathbf{q}_{ij}^{(0)} \oplus \mathbf{c}_{ij}^{(0)}, \qquad \mathbf{q}_{ij}^{(0)}, \mathbf{c}_{ij}^{(0)} \in \mathbb{R}^F \tag{5.2}$$

Here, F is the dimension of the feature vector. These inputs represent the initial (time step $t = 0$) features for message passing and are used as the foundation for subsequent multi-hop node and edge

updates.

Our EGATv2 module operates solely on the query features, while the context features are concatenated later during classification. This design choice is motivated by implementation efficiency, allowing the attention mechanism to operate on lower-dimensional inputs. The attention coefficients e_{ij} are computed using the concatenated node and edge query features (see Figure 5.4a):

$$e_{ij}^{(t)} = \mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{W}\left[\mathbf{q}_i^{(t-1)} \oplus \mathbf{q}_j^{(t-1)} \oplus \mathbf{q}_{ij}^{(t-1)}\right]\right), \tag{5.3}$$

Here, $\mathbf{a} \in \mathbb{R}^{F'}$ is a learnable attention vector, $\mathbf{W} \in \mathbb{R}^{F' \times 3F}$ is a learnable projection matrix, and LeakyReLU is the non-linear activation function.

The edge attention coefficients are normalized across each node's two nearest neighbors ($2\text{NN}(i)$), *including itself*, using a softmax function:

$$\alpha_{ij}^{(t)} = \frac{\exp(e_{ij}^{(t)})}{\sum_{k \in 2\text{NN}(i)} \exp(e_{ik}^{(t)})} \tag{5.4}$$

We aggregate only two nearest neighbors for each node (see Eq. 5.5), so that we capture local context from the immediate neighbors, and the multiple hops during iterative message passing captures wider context.

Aggregation via Message Passing. In the original GATv2, at message-passing step t, $\mathbf{q}_i^{node,(t)}$ represents the aggregated node features for node i, and $\mathbf{q}_{ij}^{edge,(t)}$ represents the aggregated edge features for edge ij. For our model,

$$\mathbf{q}_i^{\text{node},(t)} = \sum_{j \in 2\text{NN}(i)} \alpha_{ij}^{(t)} \mathbf{W}_n \mathbf{q}_j^{(t-1)} \tag{5.5}$$

$$\mathbf{q}_{ij}^{\text{edge},(t)} = \alpha_{ij}^{(t)} \mathbf{W}_e \mathbf{q}_{ij}^{(t-1)} \tag{5.6}$$

The attention coefficients $\alpha_{ij}^{(t)}$ determine how much influence neighboring elements should have during the message-passing step. In Eq. 5.5, each node i aggregates information from its two nearest neighbors j, with each neighbor's contribution $\mathbf{q}_j^{(t-1)}$ linearly transformed by $\mathbf{W}_n$ and weighted by its attention score $\alpha_{ij}^{(t)}$. This results in a context-aware representation $\mathbf{q}_i^{\text{node},(t)}$ that encodes node-level interactions.

Unlike standard GATv2, which typically considers all directly connected neighbors in the graph, our formulation restricts message passing to the two nearest neighbors including itself ($2\text{NN}(i)$). This choice is motivated by the domain-specific structure of our visual parsing tasks, where meaningful

local context is typically limited to a small spatial neighborhood. For example, in a mathematical expression, the symbol = (equal) typically occurs between its left and right operands, and in chemical diagrams, a bond often connects just two nearby atoms. In both cases, interactions beyond the immediate neighbors are less informative for local classification decisions.

In Eq. 5.6, the edge feature $\mathbf{q}_{ij}^{(t-1)}$ from the previous time step is transformed by a separate projection $\mathbf{W}_e$ and scaled by the same attention coefficient $\alpha_{ij}^{(t)}$. This enables the edge to update its representation $\mathbf{q}_{ij}^{\text{edge},(t)}$ based on its relevance to the surrounding structure as determined by the attention mechanism.

Although the attention coefficients are computed using both node and edge features, there is still no interaction between the updated node and edge representations during the aggregation step itself [141]. That is, the node features are updated solely based on neighboring node features, and edge features are updated solely from their own previous states. To enable better interaction, we introduce a fusion step where node and edge representations are concatenated after attention-based aggregation, allowing mutual contextualization before classification.

Bidirectional Node–Edge Feature Aggregation. The *bidirectional node-edge feature aggregation* refers to the design in which both node and edge representations are iteratively updated using information from each other during message passing. At each step, *node embeddings* are updated using both neighboring *node features* and the *features of incident edges*, thereby capturing not only symbol-level context but also the relationships those symbols participate in (see Figure 5.4b). Conversely, *edge embeddings* are updated using the embeddings of their *connected nodes*, incorporating contextual information from both endpoints into the relationship representation.

This mutual exchange of information allows the model to encode more structured representations. Rather than treating node and edge tasks independently, the mechanism enables *joint refinement* of features that accounts for both local visual patterns and the overall graph structure. This is particularly important in symbolic domains like mathematics and chemistry, where meaning arises from both the individual elements and their spatial connections.

Specifically, for each node i, we compute an edge-derived feature $\mathbf{q}_i^{\text{edge},(t)}$ by summing the attention-weighted edge features $\mathbf{q}_{ij}^{\text{edge},(t-1)}$ from its two nearest neighbors $j \in 2\text{NN}(i)$, as computed in Eq. 5.6:

$$\mathbf{q}_i^{\text{edge},(t)} = \sum_{j \in 2\text{NN}(i)} \mathbf{q}_{ij}^{\text{edge},(t-1)} \tag{5.7}$$

This aggregated edge feature $\mathbf{q}_i^{\text{edge},(t)}$ encodes structural signals from edges incident to node i, com-

plementing the node-level aggregation. The final node embedding is then formed by concatenating the node- and edge-derived components:

$$\mathbf{q}_i^{(t)} = \mathbf{q}_i^{\text{node},(t)} \oplus \mathbf{q}_i^{\text{edge},(t)} \tag{5.8}$$

This formulation allows each node to be contextualized both by its neighboring nodes and the edges it participates in, supporting richer structural awareness.

Next, we enhance the edge features by incorporating updated node information. Instead of averaging the endpoint node embeddings, we compute separate scalar attention weights for each node to determine its relative contribution to the edge representation.

For each edge ij, we compute individual attention weights $\beta_i^{(t)}$ and $\beta_j^{(t)}$ using a shared attention function over the updated node features:

$$\beta_i^{(t)} = \sigma\left(\mathbf{a}_e^\top \text{LeakyReLU}\left(\mathbf{W}_e \mathbf{q}_i^{\text{node},(t)}\right)\right) \tag{5.9}$$

$$\beta_j^{(t)} = \sigma\left(\mathbf{a}_e^\top \text{LeakyReLU}\left(\mathbf{W}_e \mathbf{q}_j^{\text{node},(t)}\right)\right) \tag{5.10}$$

Here, $\mathbf{W}_e \in \mathbb{R}^{F' \times F}$ and $\mathbf{a}_e \in \mathbb{R}^{F'}$ are learnable parameters, and $\sigma(\cdot)$ denotes the sigmoid function to ensure the attention weights lie in $[0, 1]$.

The node-derived feature for edge ij is then computed as a weighted combination of the two node embeddings:

$$\mathbf{q}_{ij}^{\text{node},(t)} = \frac{\beta_i^{(t)} \mathbf{q}_i^{\text{node},(t)} + \beta_j^{(t)} \mathbf{q}_j^{\text{node},(t)}}{\beta_i^{(t)} + \beta_j^{(t)}} \tag{5.11}$$

This formulation allows the model to learn node-specific contributions to edge representations, enabling asymmetric and context-aware information flow from nodes to edges. We then concatenate this with the updated edge feature computed from the previous step:

$$\mathbf{q}_{ij}^{(t)} = \mathbf{q}_{ij}^{\text{node},(t)} \oplus \mathbf{q}_{ij}^{\text{edge},(t)} \tag{5.12}$$

After k message-passing steps, we obtain the final representations for nodes and edges by concatenating their original features with the contextualized representations produced by the EGATv2 module. Specifically, for each node i and edge ij, we concatenate the initial query and context features with the updated aggregated features to produce vectors $\in \mathbb{R}^{4F}$,

Node:

$$\mathbf{f}_i^{(k)} = \mathbf{q}_i^{(0)} \oplus \mathbf{c}_i^{(0)} \oplus \mathbf{q}_i^{\text{node},(k)} \oplus \mathbf{q}_i^{\text{edge},(k)} \tag{5.13}$$

Edge:

$$\mathbf{f}_{ij}^{(k)} = \mathbf{q}_{ij}^{(0)} \oplus \mathbf{c}_{ij}^{(0)} \oplus \mathbf{q}_{ij}^{\text{node},(k)} \oplus \mathbf{q}_{ij}^{\text{edge},(k)} \tag{5.14}$$

Here, $\mathbf{q}_i^{(k)}$ and $\mathbf{q}_{ij}^{(k)}$ include the fused outputs from the multi-hop attention-based aggregation steps, which combine both node-derived and edge-derived information. As a result, each final feature vector contains four components: the initial query feature, the context feature, and two aggregated components.

This design enables the network to retain visual convolutional responses while incorporating structural and contextual dependencies extracted during message passing. Including the original query and context vectors in the final embeddings serves a similar role to residual connections in deep networks [46], which preserves earlier stage information and provides a direct path for gradient flow during training.

These enriched embeddings $\mathbf{f}_i^{(k)}$ and $\mathbf{f}_{ij}^{(k)}$ are subsequently passed to task-specific classification heads: a symbol classifier for node-level predictions, a segmentation classifier for identifying symbol merges, and a relationship classifier for determining spatial or chemical connections between entities. This architecture enables to incorporate both visual and structural context in graph-based parsing using a GNN approach.

5.2.4 Two-Stage Graph Attention with Cross-Task Interaction

Existing works like QD-GGA [72] and our current parser models LGAP and LCGP [114] utilize multi-task learning (MTL) to jointly optimize for symbol and relationship recognition tasks in mathematical formulas and chemical diagrams. Similarly, joint learning approaches have been applied to integrate keypoint detection and node (atom) classification tasks [79] for parsing chemical diagrams.

As discussed in Chapter 2 (Section 2.3.1), MTL enhances model performance by leveraging shared representations across related tasks, improving generalization, and reducing overfitting. Traditional MTL frameworks often predict outputs for all tasks in a single pass, limiting their ability to exploit inter-task relationships effectively. Recent works have demonstrated that iterative refinement, where outputs from earlier task predictions are utilized in subsequent steps, can enhance task performance by aligning features and promoting consistency between tasks. For instance, Xu et al. [142] employed spatial attention to refine task outputs iteratively, while Zhang et al. [167] proposed a sequential prediction method to propagate task-specific and cross-task information iteratively. MTI-Net [127]

extended this concept further by modeling task interactions across multiple scales.

Building on these approaches, we propose incorporating graph attention (EGATv2) mechanisms into our MTL framework to facilitate iterative refinement of task predictions. In this setup, initial outputs from tasks such as segmentation, symbol classification, and relationship classification serve as inputs to subsequent network iterations. These refined inputs are processed through EGATv2 attention layers to propagate enhanced task-specific and cross-task information effectively. For example, segmentation predictions from the first iteration can guide symbol classification by highlighting relevant regions, while relationship detection outputs can refine segmentation by enforcing spatial consistency between nodes and edges.

The graph attention mechanism in EGATv2, described in the previous section, is utilized in this iterative refinement process. It enables the model to dynamically adjust task predictions by identifying relevant interactions between task-specific outputs and shared representations. By leveraging these interactions, the proposed framework aims to achieve better alignment across tasks, leading to improved accuracy in symbol and relationship classification and enhanced segmentation performance.

Design. The proposed two-stage graph attention model extends the EGATv2 framework by introducing a second stage of message passing that leverages the output distributions from the first stage to facilitate cross-task interaction. In the first stage, the model is trained using visual features alone, which outputs classification distributions for symbol (cs_i), relationship (cr_{ij}), and segmentation/merge (cm_{ij}) tasks. These outputs are then used as inputs in the second stage to guide more informed message passing.

In the second stage, we use the backbone and EGATv2 weights learned in the first stage, and forward pass the classification outputs as input features alongside the original visual features.

Input Representation. For each node i, we construct a new feature vector by concatenating the symbol classification distribution from the first stage $\mathbf{cs}_i^{(0)} \in \mathbb{R}^{C_s}$ with the node query features $\mathbf{q}_i^{(0)} \in \mathbb{R}^F$ to form a feature vector $\in \mathbb{R}^{C_s+F}$:

$$\tilde{\mathbf{q}}_i^{(0)} = \mathbf{cs}_i^{(0)} \oplus \mathbf{q}_i^{(0)} \tag{5.15}$$

Here, C_s is the number of symbol classes, and F is the feature dimension of the original query features.

For each edge ij, we concatenate the relationship classification distribution $\mathbf{cr}_{ij}^{(0)} \in \mathbb{R}^{C_r}$, segmenta-

tion (merge) distribution $\mathbf{cm}_{ij}^{(0)} \in \mathbb{R}^{C_m}$, and the original edge query features $\mathbf{q}_{ij}^{(0)} \in \mathbb{R}^F$ to form a feature vector $\in \mathbb{R}^{C_r+C_m+F}$:

$$\tilde{\mathbf{q}}_{ij}^{(0)} = \mathbf{cr}_{ij}^{(0)} \oplus \mathbf{cm}_{ij}^{(0)} \oplus \mathbf{q}_{ij}^{(0)} \tag{5.16}$$

Here, C_r is the number of relationship classes and C_m is the number of segmentation classes, which is binary (merge or not merge) in our case, so $C_m = 2$.

These enhanced input vectors $\tilde{\mathbf{q}}_i^{(0)}$ and $\tilde{\mathbf{q}}_{ij}^{(0)}$ are used in place of the original query features in the second-stage EGATv2 module.

Aggregation via Message Passing. The second stage follows the same EGATv2 message-passing procedure described previously (see Equations 5.3–5.12), with the difference being the nature of the input features and new trainable weights. Instead of using purely visual query features, we now operate on enriched inputs: the node features $\tilde{\mathbf{q}}_i^{(0)} = \mathbf{cs}_i^{(0)} \oplus \mathbf{q}_i^{(0)}$, and the edge features $\tilde{\mathbf{q}}_{ij}^{(0)} = \mathbf{cr}_{ij}^{(0)} \oplus \mathbf{cm}_{ij}^{(0)} \oplus \mathbf{q}_{ij}^{(0)}$, as defined earlier, using same number of neighbors.

These enhanced representations encode both visual and task-specific class distributions from the first stage and are propagated over k message-passing steps using the same attention-based updates as before, which gives the aggregated node feature in $\mathbb{R}^{C_s+F}$:

$$\tilde{\mathbf{q}}_i^{(k)} = \mathbf{cs}_i^{(k)} \oplus \mathbf{q}_i^{(k)} \tag{5.17}$$

and aggregated edge feature in $\mathbb{R}^{C_r+C_m+F}$:

$$\tilde{\mathbf{q}}_{ij}^{(k)} = \mathbf{cr}_{ij}^{(k)} \oplus \mathbf{cm}_{ij}^{(k)} \oplus \mathbf{q}_{ij}^{(k)} \tag{5.18}$$

This formulation allows the model to learn interactions between node and edge features while conditioning on task-relevant distributions, thereby enabling cross-task interaction and learning.

Final Embedding Construction. After k message-passing steps, the final feature vectors are constructed by combining the original classification distributions, original visual features, and the aggregated outputs from the second-stage graph attention layers.

For nodes, a feature vector $\in \mathbb{R}^{2C_s+4F}$ is formed as follows:

$$\mathbf{f}_i^{(k)} = \underbrace{\mathbf{cs}_i^{(0)} \oplus \mathbf{q}_i^{(0)} \oplus \mathbf{c}_i^{(0)}}_{\text{Initial class, query, context}} \oplus \underbrace{\mathbf{cs}_i^{(k)} \oplus \mathbf{q}_i^{\text{node},(k)} \oplus \mathbf{q}_i^{\text{edge},(k)}}_{\text{Updated class, node query, edge query}} \tag{5.19}$$

And for edges, a feature vector $\in \mathbb{R}^{2C_r+2C_m+4F}$ is formed as follows:

$$\mathbf{f}_{ij}^{(k)} = \underbrace{\mathbf{cr}_{ij}^{(0)} \oplus \mathbf{cm}_{ij}^{(0)} \oplus \mathbf{q}_{ij}^{(0)} \oplus \mathbf{c}_{ij}^{(0)}}_{\text{Initial relationship, merge class, query, context}} \quad \oplus \quad \underbrace{\mathbf{cr}_{ij}^{(k)} \oplus \mathbf{cm}_{ij}^{(k)} \oplus \mathbf{q}_{ij}^{\text{node},(k)} \oplus \mathbf{q}_{ij}^{\text{edge},(k)}}_{\text{Updated relationship, merge class, node query, edge query}} \tag{5.20}$$

These final representations incorporate both raw and contextually refined information from multiple tasks and are fed into task-specific fully connected layers to yield updated predictions for symbol classification, segmentation (merge decisions), and relationship classification. This second-stage formulation enables iterative refinement, as the final predictions are based on feature representations that are not only visually grounded but also shaped by interactions across tasks—symbol, segmentation, and relationship recognition, through graph-based message passing.

5.3 Evaluation and Results

This section presents experiments evaluating the methods introduced in the preceding sections, which includes evaluating the effectiveness of different visual feature refinements like ResNeSt backbone, and the evaluation of our EGATv2 module and the two-stage graph attention model.

We begin with a description of datasets, evaluation metrics, and implementation details. Subsequent subsections report results for each method. Together, these evaluations demonstrate how different components of our framework influence the parser effectiveness and efficiency.

5.3.1 Datasets and Evaluation Metrics

Datasets. We evaluate our models on both mathematical and chemical diagram datasets, consistent with those introduced in Chapter 4.

For mathematical formulas, we use the InftyMCDB-2 (typeset) dataset, consisting of 12,551 training expressions and 6,830 test expressions, as previously described in Chapter 4. Additional benchmark results on other datasets, such as the CROHME dataset for handwritten expressions, are reported in Chapter 6.

For chemical diagrams, we evaluate on datasets derived from the data generation pipeline described in Chapter 3. Specifically, we use a curated set of 5,000 molecules from PubChem, prepared by the MolScribe team [100]. Of these, 4,590 molecules rendered with Indigo are successfully converted

into annotated label graphs using our born-digital parser. The remaining examples are excluded due to errors: 240 stemming from the born-digital parsing step, and 170 due to failures in primitive extraction, alignment, or conversion of visual graphs into SMILES strings. For testing, we use the USPTO dataset, which contains 5,719 chemical diagrams in PNG format generated from SMILES strings using the Indigo rendering toolkit. Both the training and test datasets are available online[1]. For training, we use 80-20 split on the PubChem-5k training dataset, resulting in 3,672 training and 918 validation examples.

For the chemistry dataset, we reduced the number of symbol classes to simplify the learning task and delegate visually deterministic patterns to post-processing. Multi-line bond types, such as double, triple, and hashed wedge bonds, were removed from the symbol classification label set. These bonds, typically rendered as grouped of parallel lines, were instead inferred deterministically by grouping parallel single bond lines after classification. This approach utilizes the rule-based method used in the born-digital parser (Chapter 3) and reduced the number of symbol classes from 74 to 71. We further excluded very rare visual elements–wave, circle, left parenthesis, and right parenthesis, which did not appear in the PubChem-5k training set or any of the benchmark test sets (USPTO, UOB, CLEF). The final label set used for training and evaluation included 67 symbol classes, better aligning with the distribution of relevant categories in the chemical diagrams.

Evaluation Metrics. For both mathematical formula and chemical diagram parsing, we report expression-level recognition rates for: (1) *Structure*, which evaluates whether the predicted graph has the correct topology, including symbol segmentation and edge connectivity; and (2) *Structure + Class*, which further requires that both symbol and relationship labels are correctly predicted.

We also compute F1 scores for symbol and relationship prediction using the LgEval toolkit [85]. These are reported for both detection accuracy (correct region or edge presence) and detection with classification (correct label assignment). The F1 score is calculated as the harmonic mean of precision and recall.

For chemical diagrams, we additionally report the *Exact SMILES Match* metric, which evaluates whether the predicted molecule's SMILES string, obtained by converting the predicted visual graph and canonicalizing it, matches the ground truth exactly. This is commonly used in chemical diagram recognition literature to compare and benchmark systems.

[1] `https://cs.rit.edu/~dprl/data/chem/fullset/`

Table 5.2: Computing environments used for experiments.

Name	GPU(s)	CPU	Memory	OS
Desktop 1	2 × GTX 1080 Ti (12GB)	Intel i7-9700KF (8-core, 3.6 GHz)	32 GB RAM	Ubuntu 22.04
Desktop 2	2 × GTX 1080 Ti (12GB)	Intel i7-8700K (12-core, 3.7 GHz)	32 GB RAM	Ubuntu 24.04
Server 1	4 × RTX 2080 Ti (12GB)	Intel Xeon E5-2667 v4 (32-core, 3.2 GHz)	512 GB RAM	Ubuntu 22.04
Server 2	1 × A40 (48GB)	Intel Xeon Gold 6326 (64-core, 2.9 GHz)	256 GB RAM	Ubuntu 24.04
RC Node	1 × A100 PCIe (40GB)	Intel Xeon Gold 6150 (36-core, 2.7 GHz)	376 GB RAM	RHEL 9.6

Table 5.3: Baseline configuration for math and chemistry experiments.

Control variables	Math	Chemistry
Symbol classes	207	67
Edge classes	9	2
Segmentation classes	2	2
Input primitives	CCs	Line primitives
Input graph	LOS graph	6NN graph
Loss	Cross-Entropy (CE)	Cross-Entropy (CE)

Implementation Details All models are implemented in Python using the `PyTorch` framework. For the visual parser, additional dependencies include `networkx` for graph operations, `numpy` for numerical routines, and `mr4mp` for map-reduce parallelism. Visual parsing pipelines and primitive extraction components are fully implemented in Python.

Experiments were conducted across multiple computing environments, which is summarized in Table 5.2. The main experiments were run on two desktops and two servers, with the Research Computing (RC) cluster node [93] used for a few experiments.

We use the Adam optimizer with a learning rate of 0.0005, $\beta = (0.9, 0.999)$, batch size of 64, and no weight decay. Models were trained for up to 100 epochs with early stopping based on validation loss. Specifically, training was halted if the total validation loss increased for more than five consecutive epochs, and the best epoch was selected as the one just before this increase began. Batch updates were performed using mixed node and edge queries sampled according to the strategy described in Section 6.2.

Baseline Model Configuration Table 5.3 shows the baseline configuration used for all experiments in both math and chemistry tasks. This setup defines consistent values for key variables such as loss function, input graph representation, and number of classes for each task. We use this configuration as a control to measure the effect of specific changes in the experiments that follow.

Table 5.4: Training and inference performance metrics for InftyMCDB-2 (math) and Pubchem-5k (chemistry) datasets on Server 1 using the baseline model configuration defined in Table 5.3.

Metric	Math	Chemistry
Dataset size (train/test)	12,551 / 6,830	4,590 / 5,719
Avg. input instances (nodes + edges) per formula	55.27	345.23
Avg. training time per epoch	22.4 minutes	101.6 minutes
Inference time per formula	81 ms/formula	915 ms/molecule

Additional variables, such as feature size, number of pooling regions, backbone encoder, number of hops and stages in EGATv2 are treated as independent variables and modified in isolation to evaluate their impact on performance of the parser.

Training and Inference Performance. Table 5.4 summarizes the training and inference performance for our models on the math and chemistry datasets using Server 1 (see Table 5.2 for server details). While the training time per epoch for mathematical formulas is relatively low (22.4 minutes), training on chemical diagrams takes substantially longer (101.6 minutes). Similarly, inference on math expressions is faster (81 ms/formula) compared to chemical diagrams (915 ms/molecule). These differences are primarily due to the significantly larger number of input instances (i.e., nodes and edges) in chemical graphs. On average, chemical molecules contain approximately six times as many nodes and edges (345.23 instances) as mathematical formulas (55.27 instances), increasing the computational load during both training and inference.

In the following sections, we present experiments addressing the methods introduced earlier.

5.3.2 Common Feature Extraction Pipeline

To assess the impact of adopting the contour-based primitive representation for mathematical formulas, we conduct a comparative evaluation against the previous binary-masked approach used in earlier versions of our math parsing models (LGAP), as described in Chapter 4. The earlier method used large formula-level crops with binary attention masks over CNN feature maps to isolate connected components (CCs) and their neighborhoods, which led to sparse feature maps and inefficiencies in capturing localized details. In contrast, the updated approach applies the same contour-based, resolution-normalized primitive canvas strategy originally developed for chemical diagrams (Section 5.2.1).

Table 5.5: Effect of visual input primitives on parsing performance for mathematical formulas (InftyMCCDB-2 test dataset).

Input features	Symbols		Relationships		Expressions	
	Detect.	+Class	Detect.	+Class	Structure	+Class
Global pixel map and binary masks	98.32	95.66	94.85	94.35	89.27	83.27
Drawn contour-based primitives	**98.95**	**96.83**	**95.96**	**95.13**	**90.32**	**84.55**

For this experiment, we retrain the model using the same architecture and supervision as the baseline, changing the visual input representation and the feature generation strategy. In the updated pipeline, contours are extracted from each connected component in the math dataset, drawn onto fixed-size patches, and passed through a shared CNN encoder independently. As shown in Table 5.5, this change results in consistent improvements across symbol classification, relationship recognition, and expression-level accuracy, highlighting the effectiveness of contour-based primitives as the standard visual representation.

5.3.3 Visual Feature Improvements

We evaluate the impact of improved visual primitives and feature configurations on parsing performance for chemical diagrams. The updated primitives are extracted using a combination of the Line Segment Detector and watershed segmentation, as described in Section 5.2.2, and are compared against earlier extraction methods detailed in Chapter 3.

We conducted a grid search on input feature resolution (28×28, 32×32, 48×48, and 64×64), and spatial pyramidal pooling regions (1, 7, 17, 31) regions using representative smaller subsets of math and chemical datasets. The experiments are detailed in Appendix A. For input resolution, 64×64 input feature size yielded the highest recognition for mathematical formulas while 32×32 feature size performed best for chemical diagrams.

For spatial pooling, models benefited from increased pooling regions. In particular, a 31-region configuration ($1 + 3H + 3V + 5H + 5V + 7H + 7V$) was most effective for math, while a 17-region setup ($1 + 3H + 3V + 5H + 5V$) provided the best performance for chemistry.

Based on these results, we adopt 64×64 input features with 31-region SPP for math, and 32×32 input features with 17-region SPP for chemistry, and use these as default configurations in all subsequent experiments.

As shown in Table 5.7, updates to visual primitives and tuning of input feature size and pooling regions lead to consistent performance improvements for the chemical diagrams. Both symbol and relationship detection and classification F1 scores increase across stages, and the exact SMILES match rate improves from 85.02% to 86.20%. These results demonstrate the value of new robust primitives and feature resolution optimization in enhancing chemical diagram parsing accuracy. For mathematical formulas, we retain the same configuration throughout, as the 64×64 input size with 31-region pooling was already optimal, giving an expression-level structure and classification rate of 84.55%.

5.3.4 Split-Attention (ResNeSt) Backbone

To assess the effectiveness of the split-attention ResNeSt encoder, we compare the modified ResNeSt-50 (reduced) variant against the SE-ResNeXt-50 baseline described in Section 5.2.2. Both models were trained using the same updated pipeline, which includes contour-based primitive inputs and shared CNN visual encoding. All other components (e.g., graph structure, multi-task heads, loss formulation, and pooling configurations) were held constant to isolate the effect of the encoder architecture.

Training Details. We used pretrained ResNeSt-50 weights from ImageNet as initialization. Since our inputs are single-channel contour patches (grayscale), the first convolutional layer's weights were adapted by averaging the pretrained weights across the three RGB channels. This simple conversion makes the model compatible with single channel inputs. We also trained the model from scratch (without pretrained weights), and found that while convergence was slower, final performance remained within 0.2–0.3% of the pretrained version across tasks, confirming the robustness of the architecture.

For the mathematical formula dataset (InftyMCDB-2), evaluation was performed on the full test set using 64×64 input feature and a 31-region SPP configuration ($1+3H+3V+5H+5V+7H+7V$). For the chemical dataset (USPTO), we used 32×32 input feature size and a 17-region SPP configuration ($1 + 3H + 3V + 5H + 5V$), consistent with the best-performing setup identified in Section 5.3.3. All models were trained with the same hyperparameters, and data splits to ensure comparability.

Discussion. The ResNeSt-based model yields consistent improvements across all evaluation metrics compared to the SE-ResNeXt baseline. Increases are observed in both local (symbol, relation)

Table 5.6: Performance comparison between SE-ResNeXt and ResNeSt backbones on the InftyMCDB-2 test set. Both models use contour-based primitive inputs and 31-region SPP.

Backbone	Symbols		Relationships		Expressions	
	Detect.	+Class	Detect.	+Class	Structure	+Class
SE-ResNeXt-50 (baseline)	98.95	96.83	95.96	95.13	90.32	84.55
ResNeSt-50 (reduced)	**99.00**	**96.90**	**96.00**	**95.20**	**90.56**	**84.73**

Table 5.7: Performance comparison on the USPTO chemical test set across different visual feature improvements. Models are evaluated using F1 scores for symbol and relationship detection and classification, expression-level rates (%) for structure and structure + class, and the percentage of exact SMILES string matches converted from the predicted visual graphs.

Model	Symbols		Relationships		Expressions		Exact SMILES
	Detect.	+Class	Detect.	+Class	Structure	+Class	(%)
Baseline (LCGP)	99.84	99.29	98.86	98.86	50.65	44.37	85.02
+ LSD Visual Primitives	99.85	99.35	98.88	98.88	51.56	45.07	85.62
+ Tuned Feat. Size and Pooling	99.86	99.43	98.89	98.89	51.98	45.55	86.20
+ ResNeSt Backbone	**99.87**	**99.50**	**98.90**	**98.90**	**52.35**	**45.98**	**86.30**

and global (structure, expression) metrics as seen in Table 5.6. These improvements extend to the chemical dataset as well, where symbol and relationship classification F1 scores improve slightly, and the exact SMILES match rate increases from 86.20% to 86.30% (Table 5.7). Notably, these gains are achieved with fewer trainable parameters and reduced computational cost, as shown in Table 5.1. The reduced ResNeSt backbone provides higher representational capacity through split-attention mechanisms, enabling the model to better distinguish fine-grained spatial patterns while preserving computational efficiency.

Accordingly, the reduced ResNeSt model with pretrained initialization is used as the default visual encoder in all subsequent experiments in Chapter 5 and Chapter 6.

5.3.5 RQ3: Graph Attention and Task Interaction

To evaluate the effect of edge-aware attention with multi-hop message passing, we conduct a systematic comparison of **EGATv2** variants against a baseline without graph attention. EGATv2 incorporates both node and edge features into the attention computation and enables joint node-edge updates across multiple message-passing iterations.

Model Variants. We consider both single-stage and two-stage configurations of EGATv2. In the single-stage setting, we vary the number of attention hops ($K = 1, 2, 3$), corresponding to the number of message-passing layers applied sequentially over the input graph. In the two-stage variant, the EGATv2 model is first trained independently to produce initial class distributions. These predicted distributions are then used as additional input features in the second stage. The second-stage EGATv2 model reuses all weights from the first stage and extends it by introducing new attention heads and linear layers to process the concatenated visual features and class distribution vectors. As such, the second stage builds upon and subsumes the first, enabling parameter sharing while incorporating task-aware layers for improved representation learning.

Implementation Details. Each EGATv2 layer comprises two linear projections: one for computing attention scores and another for updating node and edge feature embeddings. Attention scores are computed using a concatenation of feature vectors from each query and its neighbors. For a visual embedding of dimension F, the concatenated pairwise feature dimension becomes $2F$, matching the attention input dimensionality.

For math experiments, with 31 spatial pooling regions and 32 output channels from the CNN encoder, plus 10 dimensions from positional encoding bounding boxes (Query BB relative to formula window, Query BB relative to context window, and Context windoe BB relative to the formula), the visual embedding dimension is $32 \times 31 + 10 = 1002$, yielding an attention input dimension of 2004. Similarly, for chemistry with 17 pooling regions, the input to attention is $2 \times (32 \times 17 + 10) = 1108$. Attention scores weight neighboring feature embeddings during aggregation.

The downstream heads for symbol classification, segmentation, and relationship classification share a multi-layer perceptron (MLP) with a hidden layer of size 512 and ReLU activation. These heads operate on the final concatenated representation, which includes visual features, class distribution vectors (for two-stage models), and the output of the multi-hop EGATv2 layers.

All baseline and EGATv2 models use the same baseline configurations as described in Table 5.3, and also shared backbone encoder for extracting visual features: a reduced version of ResNeSt-50 with the same input feature size and SPP configuration adopted in Section 5.3.3.

As shown in Table 5.8, integrating EGATv2 increases the number of trainable parameters relative to the reduced ResNeSt-50 encoder, primarily due to expanded linear layers used in contextualized attention. These layers operate on larger input embeddings that incorporate aggregated query features. While the baseline model processes $2F$-dimensional features before classification, the

Table 5.8: Model parameter comparison of EGATv2 variants with baseline

Model	**# Trainable Parameters**
SE-ResNeXt-50	4.05M
ResNeSt-50 (reduced)	2.62M
ResNeSt-50 (reduced) + EGATv2 (1 stage)	2.97M
ResNeSt-50 (reduced) + EGATv2 (2 stages)	4.45M
CTC transformer [7]	43.80M
Vision transformer [59]	51.10M
SWIN transformer [146]	88.00M

Table 5.9: Performance of EGATv2 variants by stage and hop count across tasks on the InftyMCDB-2 dataset (6,830 formulas). We report F1% scores for symbol detection and classification, relationship detection and classification, and expression-level structure and classification rates.

Model	**Symbols**		**Relationships**		**Expressions**	
	Detect.	**+Class**	**Detect.**	**+Class**	**Structure**	**+Class**
Baseline	99.00	96.90	96.00	95.20	90.56	84.73
EGATv2, 1 stage						
1-hop	99.05	97.05	96.12	95.29	90.67	84.79
2-hop	99.23	97.46	96.92	96.17	92.14	86.94
3-hop	99.15	97.29	96.58	95.88	90.94	84.92
EGATv2, 2 stage						
1-hop	99.12	97.21	96.48	95.63	91.43	85.93
2-hop	**99.31**	**97.65**	**97.18**	**96.45**	**92.21**	**87.99**
3-hop	99.21	97.38	96.74	96.01	91.64	86.41

EGATv2 variants expand this to $4F$ in the single-stage model and to $2C_s + 2C_m + 4F$ for edges and $2C_s + 4F$ for nodes in the two-stage model. As a result, the parameter count grows from 2.62M in the baseline to 2.97M in the single-stage model (a 13.1% increase) and to 4.45M in the two-stage model (a 69.8% increase).

Despite these additions, both EGATv2 variants remain lightweight. Even the two-stage model remains closer to SE-ResNeXt-50 (4.05M parameters) and far smaller than transformer-based models like the CTC Transformer [7] (43.80M) and the Hybrid Vision Transformer [59] (51.10M).

Discussion. All EGATv2 variants outperform the baseline across symbol, relationship, and expression-level tasks in both mathematical formulas and chemical diagrams. Among single-stage models, increasing the number of hops from 1 to 2 yields consistent improvements, particularly in relationship classification (e.g., +0.88 F1% in math, +0.06 F1% in chemistry), as multi-hop message passing

Table 5.10: Performance of EGATv2 variants by stage and hop count across tasks on the USPTO test set. We report F1% scores for symbol and relationship detection and classification, expression-level structure and classification rates, and percentage of exact SMILES string matches.

Model	Symbols		Relationships		Expressions		Exact SMILES
	Detect.	+Class	Detect.	+Class	Structure	+Class	(%)
Baseline	99.87	99.50	98.90	98.90	52.35	45.98	86.30
EGATv2, 1 stage							
1-hop	99.88	99.52	98.93	98.93	52.44	46.56	86.64
2-hop	99.89	99.57	98.99	98.99	52.53	48.23	88.04
3-hop	99.88	99.53	98.94	98.94	52.42	46.83	86.83
EGATv2, 2 stage							
1-hop	99.89	99.55	98.96	98.96	52.49	47.39	87.11
2-hop	**99.90**	**99.86**	**99.02**	**99.02**	**52.54**	**52.02**	**89.04**
3-hop	99.89	99.56	98.97	98.97	52.46	47.70	87.39

facilitates richer context aggregation from neighboring nodes and edges. However, further increasing the hop count beyond 2 results in diminishing or slightly degraded performance. This decline may be attributed to the inclusion of excessive context from distant or less relevant nodes, which can dilute task-specific signal and increase learning complexity, or to underfitting due to the model's limited capacity to extract patterns from these additional contextual inputs.

Two-stage variants consistently outperform their single-stage counterparts across both domains. This confirms the benefit of leveraging intermediate classification outputs as contextual input for the second-stage attention mechanism. The two-stage, two-hop model yields the best performance overall—improving symbol classification by +0.75 F1% and expression-level structure and classification rate by +3.26% in math, and achieving a +6.04% gain in expression level structure and classification rate and +2.74% improvement in exact SMILES match in chemistry. These gains reflect the model's ability to condition message passing not only on visual features but also on early task-specific predictions.

Improvements on the chemical dataset are smaller in magnitude compared to the mathematical domain. This is partly due to the higher average number of nodes and edges per graph in chemical diagrams (see Table 5.4), which increases prediction complexity and reduces the impact of localized improvements in symbol or relationship metrics. Nonetheless, even modest gains in local F1 scores lead to noticeable increases in expression-level structure and classification rates and exact SMILES match, due to the structural propagation of correctness.

It is also notable that SMILES exact match rates remain relatively high even when expression-level

structure and classification rates are lower. This discrepancy arises because small local errors in the visual graph, such as missing or extra edges, may not affect the final molecular graph produced after postprocessing. The chemical graph reconstruction step groups bond lines (e.g., double or triple bonds) and infers connectivity based on bond group adjacency. As long as at least one edge connects the group to the correct neighbor, the resulting molecular graph remains structurally valid, leading to a correct SMILES string. Thus, expression-level errors in the visual graph can be partially absorbed during postprocessing, explaining the higher SMILES match rates despite imperfect intermediate predictions.

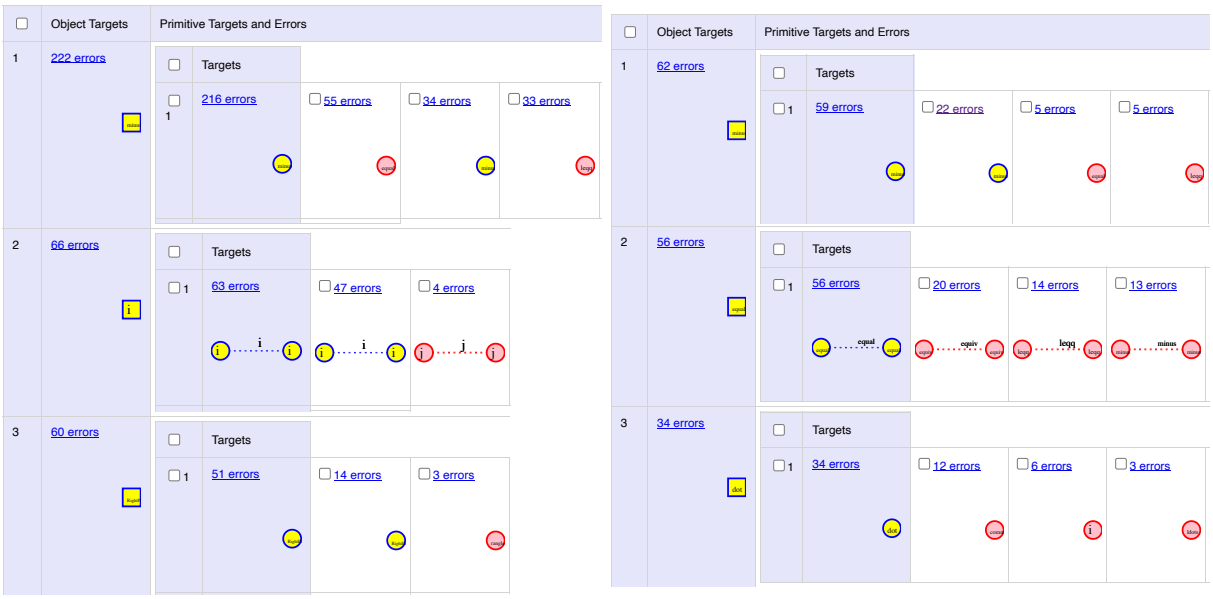
These improvements are achieved without increasing the number of parameters beyond the attention layers and maintain a lower footprint than transformer-based alternatives (see Table 5.8). The architecture supports expressive and efficient multi-task learning while preserving interpretability and generalization.

Error Analysis (Math). Figures 5.5 and 5.6 provide a detailed node- and edge-level error analysis on the InftyMCDB-2 test set, comparing the baseline model with the proposed EGATv2 (2-stage, 2-hop) model. These visualizations illustrate that EGATv2 consistently reduces the frequency of common misclassification patterns by better incorporating local context during prediction.

At the node level (Figure 5.5), the baseline model's most frequent errors involve visually similar symbols such as `minus`, `i`, and `RightParenthesis`. EGATv2 shows a lower overall frequency of these confusions, with its top errors shifting to `minus`, `equal`, and `dot`. Notably, the reduced confusion between `minus` and `equal` highlights the model's improved use of relational context: since the symbol `equal` typically consists of two horizontal lines that resemble stacked `minus` symbols, distinguishing between them without contextual cues can be difficult. EGATv2's architecture enables contextual feature propagation across neighboring primitives, which helps disambiguate such cases by leveraging surrounding symbol identities and edge types.

At the edge level (Figure 5.6), many errors in both models are a consequence of upstream symbol misclassifications. For instance, if a primitive is misclassified as `minus` instead of `equal`, the corresponding spatial relationship (e.g., `HORIZONTAL`) may also be incorrectly interpreted, leading to compounded edge-level errors. While these error chains persist in EGATv2, their overall frequency is reduced compared to the baseline, highlighting the benefits of bidirectional interaction between node and edge representations in our architecture.

Nevertheless, some residual errors in EGATv2 suggest remaining limitations in how context is ag-



(a) Baseline

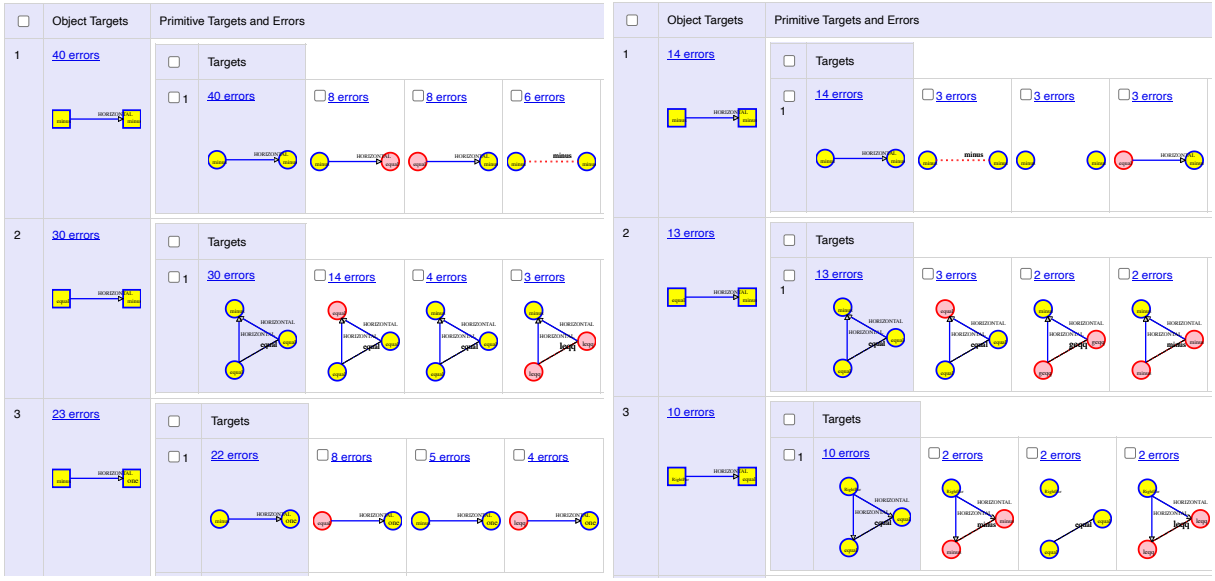
(b) EGATv2, 2-stage 2-hop

Figure 5.5: Node-level error analysis comparison between two models on the InftyMCDB-2 test set in Table 5.9: (a) baseline model, and (b) the best performing EGATv2 (2-stage, 2-hop) model (truncated at right for space). Each error table is organized by decreasing frequency of object-level node classification errors, with associated primitive-level confusions visualized to the right of each object. Red circles and lines indicate misclassifications, blue circles denote the correct class, and dotted lines illustrate merge edges between primitives belonging to the same symbol. Hyperlinked error counts allow inspection of specific formulas, and checkboxes enable selection for export. (a) The baseline model is dominated by errors involving `minus`, `i`, and `RightParenthesis` symbols, and (b) the EGATv2 model's top three errors are classifying `minus`, `equal`, and `dot`, but with lower error frequencies compared to the baseline.

gregated. While EGATv2 propagates query features across neighbors, additional gains may require mechanisms that jointly aggregates both query and context features.

Error Analysis (Chem). Figures 5.7 and 5.8 present node- and edge-level error visualizations for the USPTO test set. These reveal similar trends as observed in the math domain, with EGATv2 (2-stage, 2-hop) reducing both symbol misclassification and edge prediction errors compared to the baseline.

At the node level (Figure 5.7), the most frequent misclassifications in both models involve the `Single`



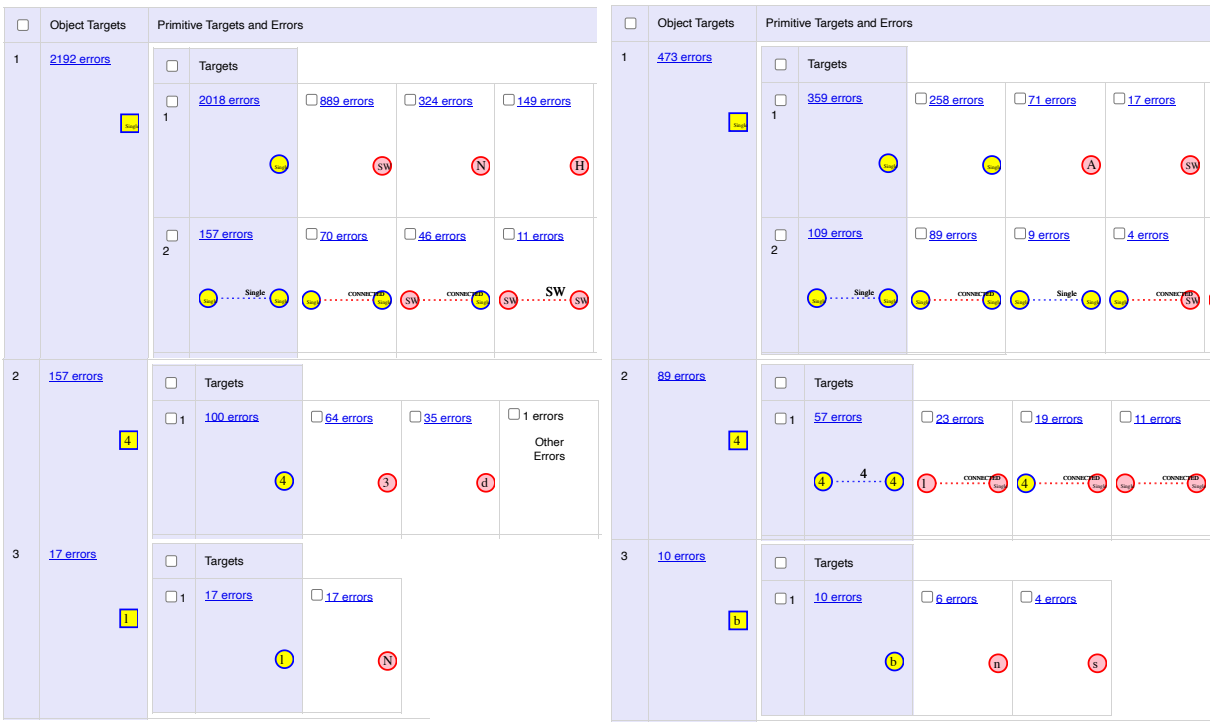
(a) Baseline (b) EGATv2, 2-stage 2-hop

Figure 5.6: Edge-level error analysis comparing the (a) baseline model with the (b) EGATv2 (2-stage, 2-hop) model on the InftyMCDB-2 test set (truncated at right for space) in Table 5.9. In both models, the top errors involve propagated symbol classification errors, particularly misclassifications of `minus` and `equal` symbols, which result in incorrect relationship detections. However, EGATv2 shows a substantially lower frequency of such propagated misclassification errors.

line primitive, which dominates the class distribution and constitutes over 50% of all primitives, as described in Chapter 6. Due to its simple appearance as a straight line, the `Single` primitive is easily confused with short strokes that appear in atom characters such as `N` and `H`, which often include similar vertical or diagonal lines.

EGATv2 reduces this confusion by over 4 times by incorporating neighborhood context during node classification. For instance, a single line surrounded by circular strokes or recognized atoms is more likely to be a character component rather than a standalone bond. This contextual differentiation improves recognition accuracy on heavily overrepresented and ambiguous primitives.

At the edge level (Figure 5.8), both models exhibit the highest frequency of errors in detecting connections between pairs of `Single` line primitives. EGATv2 reduces such missed connections, though residual errors remain. Several factors may contribute to persistent edge errors: (1) the underlying visual graph representation connects primitives based on spatial heuristics (e.g., proximity, orientation), which can omit legitimate bonds if primitives are too far apart or misaligned; (2) class



(a) Baseline

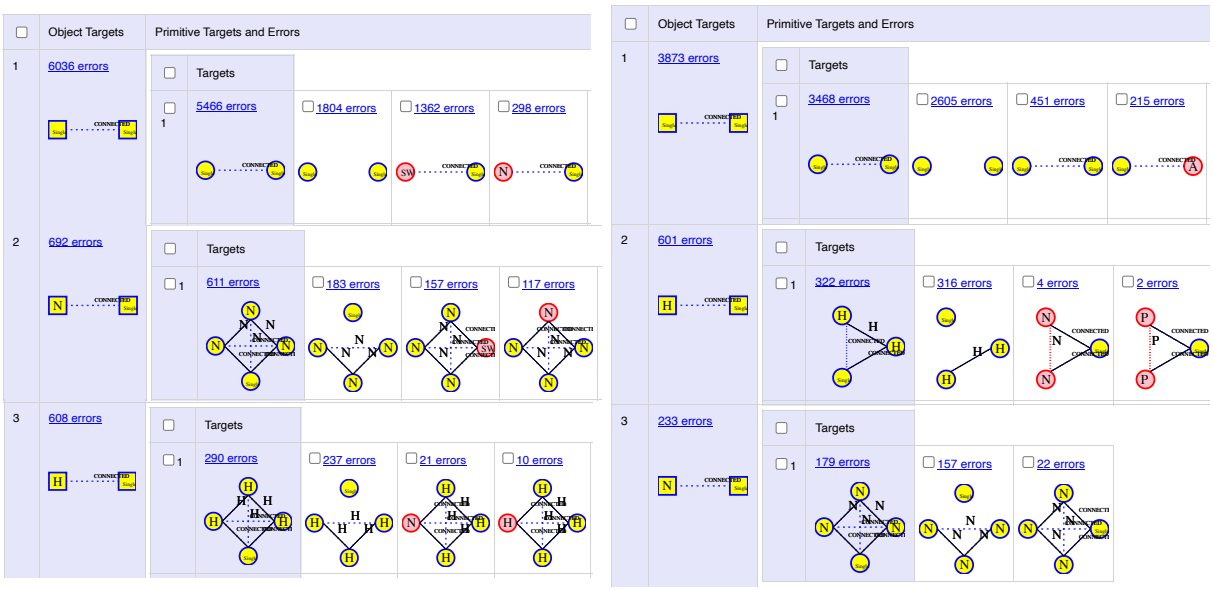
(b) EGATv2, 2-stage 2-hop

Figure 5.7: Node-level error analysis comparison between two models on the USPTO test set: (a) baseline model, and (b) the best performing EGATv2 (2-stage, 2-hop) model (truncated at right for space) in Table 5.10. `Single` line and `'4'` are the most frequent sources of error in both models; however, EGATv2 reduces `Single` line errors by nearly 7 times and `'4'` errors by almost half. The third most frequent error is `'1'` in the baseline and `b` in EGATv2.

imbalance at the edge level may skew training toward the dominant `NoRelation` or negative class.

Moreover, the current graph representation does not explicitly account for merged chemical structures, such as double or triple bonds formed by grouped parallel strokes. These are handled in post-processing, where single lines are merged based on geometric rules. A direction for future work is to design graph construction strategies that more closely align with the final semantic entities, such as atoms and bonds by representing compound primitives as single nodes and integrating edge merging into the parsing model itself.

EGATv2 demonstrates clear improvements in both math and chemistry domains by leveraging edge-aware and context-sensitive message passing, though future refinements in graph design and edge supervision could further reduce residual ambiguities. We adopt the 2-stage, 2-hop EGATv2 model



(a) Baseline

(b) EGATv2, 2-stage 2-hop

Figure 5.8: Edge-level error analysis comparing the (a) baseline model and the (b) EGATv2 (2-stage, 2-hop) model on the USPTO test set (truncated at right for space) in Table 5.10. In both models, the most frequent edge errors involve missed connections between two `Single` lines as well as `Single` line with other atoms such as `H` and `N`. EGATv2 substantially reduces the frequency of these errors, indicating improved contextual understanding of bond relationships in molecular structures.

as the default architecture for all subsequent experiments in Chapter 6, which evaluate robustness to noise and strategies for class imbalance mitigation.

5.4 Summary

This chapter addressed two central research questions: **RQ2** and **RQ3**, while also contributing partially to **RQ1** through refinements in visual primitive extraction.

For **RQ2**, we evaluated alternative input graph representations: Complete, KNN, and Line-of-Sight (LOS), to determine their suitability for math and chemical parsing tasks. We found that LOS graphs better capture the long-range, spatially meaningful relationships in mathematical formulas, due to their largely horizontal layout and symbolic alignment patterns, while 6NN graphs offered a good balance of sparsity and recall for chemical diagrams. This domain-specific graph selection enables reliable downstream parsing by ensuring essential structural relationships are preserved

during learning. This chapter also introduced enhancements to the visual input primitives, partially addressing **RQ1**. A unified contour-based primitive extraction approach was adopted for both math and chemistry, improving visual consistency and enabling a shared encoder across domains. For chemical diagrams, we further refined the visual primitive extraction from raster images using a Line Segment Detector (LSD) combined with watershed segmentation, enabling the separation of over-segmented or touching primitives. This method can be extended to other domains, including math formulas, charts, and other diagrams.

For **RQ3**, we developed and evaluated **EGATv2**, a novel edge-aware GATv2-based architecture that performs multi-hop message passing within a localized neighborhood and supports dual updates of node and edge embeddings. EGATv2 integrates both visual and class distribution features extracted from node- and edge-centric input windows. These improvements build on a unified visual feature pipeline and a compact ResNeSt-50 backbone with split attention, configured with domain-specific input sizes and pooling strategies. The two-stage, two-hop EGATv2 model consistently outperformed baselines across all tasks and datasets, demonstrating the benefits of contextual feature propagation and cross-task refinement.

Finally, detailed node- and edge-level error analyses highlighted common misclassification patterns, particularly under noisy or ambiguous visual conditions and in the presence of class imbalance. For example, confusions between `minus` and `equal` in math, or between `Single` bond segments and characters like `N` and `H` in chemistry, revealed challenges in distinguishing visually similar primitives. These findings motivate the focus of the next chapter, which addresses **RQ4** by introducing robustness enhancements through localized visual and structural noise augmentation, and by mitigating class imbalance using loss reweighting and stratified sampling strategies.

Chapter 6

Visual Noise and Loss Functions

In real-world scenarios, visual parsing systems encounter noisy and degraded inputs, especially in handwritten diagrams, and scanned documents. These documents commonly suffer from visual distortions (blur, noise, resolution degradation) and structural imperfections (fragmented symbols). Moreover, imbalanced class distributions, characterized by a prevalence of common symbols and relationships (e.g., bond lines in chemistry, frequent operators in mathematics), and a relative scarcity of rare classes, pose additional challenges by biasing the training process toward dominant classes, thus hindering generalization.

Motivated by these issues, this chapter presents experiments and methods designed to address research question **RQ4**, focusing on improving parser robustness and generalization under noise, and mitigating class imbalance. We introduce and evaluate methods that explicitly account for visual and structural degradations at the primitive level through targeted synthetic noise augmentation. Our noise modeling pipeline simulates realistic conditions like blur, resolution reduction, and structural noise (i.e., primitive fragmentation). To address severe class imbalance, we employ class-aware loss reweighting strategies, including weighted cross-entropy, class-balanced loss [27], and focal loss [66]. These strategies aim to enhance the parser's ability to handle varied input conditions and skewed class distributions, aligning with recommendations and future directions previously highlighted by Mahdavi [70], who emphasized the importance of loss balancing and focal loss for addressing task difficulty and sample imbalance. We also explore an alternative loss aggregation strategy, the complemented harmonic mean (CHM), which dynamically balances gradient contributions across the three tasks.

These strategies aim to enhance the robustness, stability, and generalization capabilities of our

visual parser. The remainder of this chapter details the methodologies, experimental setups, and evaluations conducted to address these research questions.

6.1 Visual Noise Augmentation

Our approach to noise augmentation is grounded in the degradation modeling framework proposed by Baird [11], which outlines a comprehensive taxonomy of document image defects and methods for simulating and validating them. Baird emphasizes that document degradations result from a sequence of physical, mechanical, scanning, and transmission-related transformations, each of which can significantly impair recognition systems such as OCR. These degradations are not uniform in scale: they occur at the page level (e.g., skew, uneven illumination), symbol level (e.g., dropout, blur, occlusion), and pixel level (e.g., sensor noise, quantization errors).

Baird classifies degradation models into two primary categories: *physics-based models* and *statistics-based models*. Physics-based models simulate the physical mechanisms of document distortion such as defocus (modeled using Gaussian blur), pixel misalignment (jitter), sensor noise (Gaussian noise), and binarization thresholding errors. Statistical models, on the other hand, approximate the effects of degradation by introducing speckle noise, dropout, and spatial distortions without modeling the underlying physical processes explicitly.

Our augmentation pipeline is designed to reflect both of Baird's modeling approaches. It departs from prior document analysis work that applies uniform, full-image noise by instead introducing degradations *locally at the visual primitive window level*. This is enabled by the feature design in our parsing framework (see Chapter 4), where each model input consists of a *query image* (for a node or edge) and a *context image* showing its $k = 6$ nearest neighbors for both math and chemical diagrams. For math formulas, the nearest neighbors within the input LOS graph are considered. Both query and context images are fixed size binary images generated by rasterizing tightly-centered primitive or primitive pair contours.

To reflect the variety of document degradations, our augmentation pipeline introduces two types of noise: *structural noise*, which simulates segmentation errors, and *visual noise*, which simulates symbol and pixel-level degradations. This design captures both geometric and appearance-based distortions commonly found in scientific documents.

6.1.1 Structural Noise

Structural noise is applied as the first step in our augmentation pipeline and simulates segmentation-level distortions by randomly oversegmenting visual primitives. This mimics common real-world artifacts such as stroke breaks, ink discontinuities, or scanner-induced gaps, which can fragment symbols in degraded or low-resolution diagrams.

During training, each primitive in a batch has a 20% probability of being oversegmented. The 20% rate is chosen arbitrarily as a balance of introducing enough structural variation to simulate real-world symbol fragmentation, while keeping most samples intact to allow learning on noise-free samples. A lower rate would not provide enough diversity to learn robust recovery from segmentation noise, while a higher rate may overwhelm the model with noisy inputs, hindering convergence. The procedure for applying structural noise is detailed in Algorithm 2. In brief, a random split line is introduced across the primitive contour, dividing it into two disconnected fragments (see Figure 6.1). A size filter ensures that neither resulting segment is too small, discarding the split if the smallest fragment comprises less than 10% of the original area, or if the operation would produce more than two disconnected parts.

If a split is successful, the original primitive node is retained, and two new nodes are added corresponding to the resulting fragments. All three nodes—the original and the two split parts are assigned the same symbol label. A new `MERGE` edge is added between the two split nodes to indicate their shared origin. Additionally, all edges that previously connected to the original node are duplicated for each of the new nodes with the same labels. This transformation increases both the node and edge count in the training batch.

This augmentation also affects the recurrent segmentation updates of the model. Since the newly introduced fragments are treated as additional input nodes, they are included in the model's segmentation and classification passes. As a result, the number of segmentation decisions increases during training, expanding the search space and encouraging the model to learn from these recurrent updates on the over-segmented primitives.

However, this structural noise augmentation is limited in scope: it only introduces topological variation by altering the segmentation structure and does not simulate appearance-level degradation. In real-world documents, visual noise such as blur, distortion, and resolution loss can significantly affect recognition. To address this, we introduce a visual noise augmentation strategy in the next section, targeting the pixel-level appearance of query and context windows.

Algorithm 2 Structural Noise: Primitive Oversegmentation

Require: Primitive contour P, split probability $p = 0.2$, minimum fragment ratio $\delta = 0.1$

for each primitive P_i in batch **do**
  if random() $< p$ **then**
    (Splitting operation)
      (i) Select a random point q on P_i
      (ii) Choose random direction $\theta \in [0, 2\pi)$
      (iii) Compute unit vector $(dx, dy) = (\cos\theta, \sin\theta)$
      (iv) Rasterize P_i into binary mask M by filling the contours
      (v) Create a thin line L (2-pixel width) through q in direction θ onto M
      (vi) Cut the mask: $M_{\text{cut}} \leftarrow M$ with L erased
      (vii) Label connected components in M_{cut} as $\{C_1, C_2, \dots\}$

    (Filtering)
      (i) Skip non-binary splits:
        if number of components $\neq 2$ **then**
          continue to next primitive
      (ii) Let $A = \text{area}(P_i) = \sum M$ be the area of the original primitive
      (iii) Compute area of each component: $A_1 = \text{area}(C_1)$, $A_2 = \text{area}(C_2)$
      (iv) Skip small components:
        if $\min(A_1, A_2) < \delta \cdot A$ **then**
          continue to next primitive

    (Graph augmentation)
      (i) Extract new contours P_i^1, P_i^2 from C_1, C_2
      (ii) Assign both P_i^1 and P_i^2 the same symbol label as P_i
      (iii) Add new nodes v_1, v_2 corresponding to P_i^1, P_i^2
      (iv) Add edge (v_1, v_2) with label `MERGE`
      (v) Add edges to neighbors of P_i:
        for each neighbor v_j of P_i **do**
          Add edges (v_1, v_j) and (v_2, v_j) with the same labels as (P_i, v_j)
return the union of original and augmented primitive nodes and edges

This augmentation increases the number of nodes and edges processed within the batch. It expands the supervision signal during training, as the model must now classify edge and node labels across

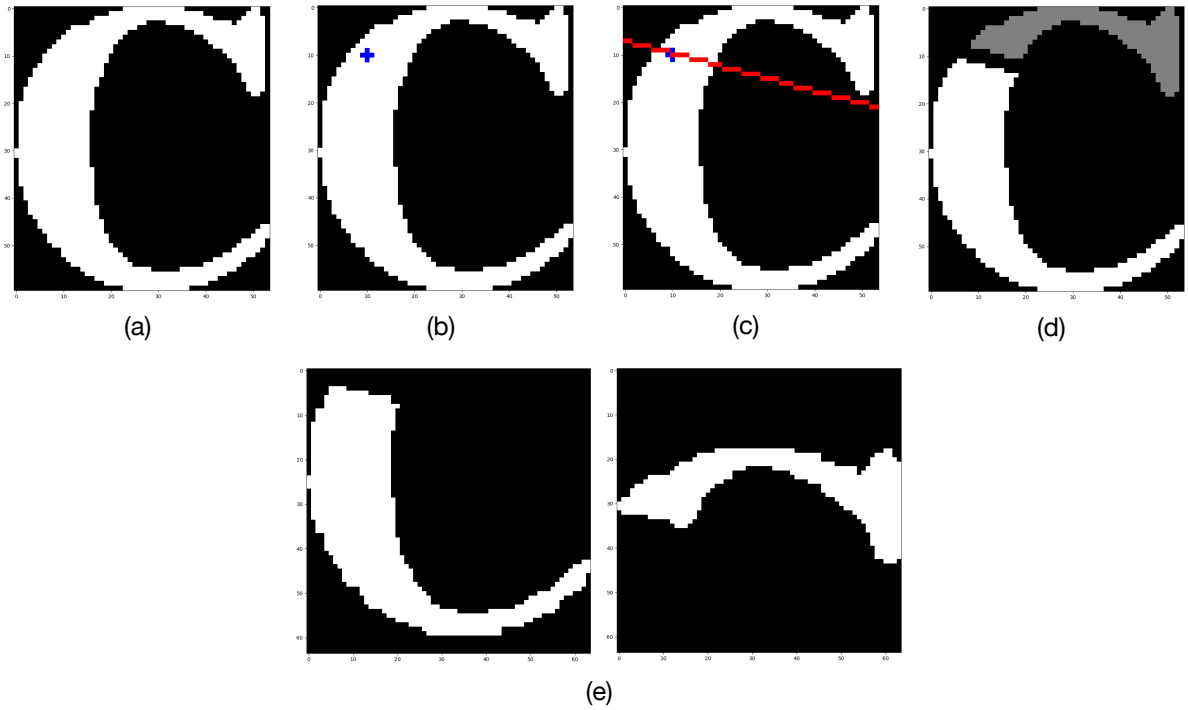


Figure 6.1: Illustration of structural noise applied to a primitive as described in Algorithm 2. (a) Original primitive (b) Random split point (in blue) selected (c) Random split direction (in red) determined (d) Primitive split along the line into two primitives (e) Resulting primitive features after applying the split, followed by rescaling and centering of contours

both original and synthetically split structures. This creates more difficult learning scenarios, for segmentation, symbol and relationship classification tasks, encouraging the model to become more robust to over-segmentations commonly found in real-world inputs.

6.1.2 Visual Noise for Primitive and Context Windows

After structural noise is applied, visual noise, such as Gaussian blur, resolution downscaling, and salt-and-pepper noise are applied independently with randomized parameters to each query and context window. This form of visual noise simulates localized distortions such as scanner blur, lossy compression, or pixel-level corruption. Each query or context window, centered on a single primitive or primitive pair, can thus experience different types and intensities of degradation, introducing greater variability during training. By enabling spatially varied noise within a single input image, this approach better reflects the heterogeneous artifacts commonly found in real-world documents.

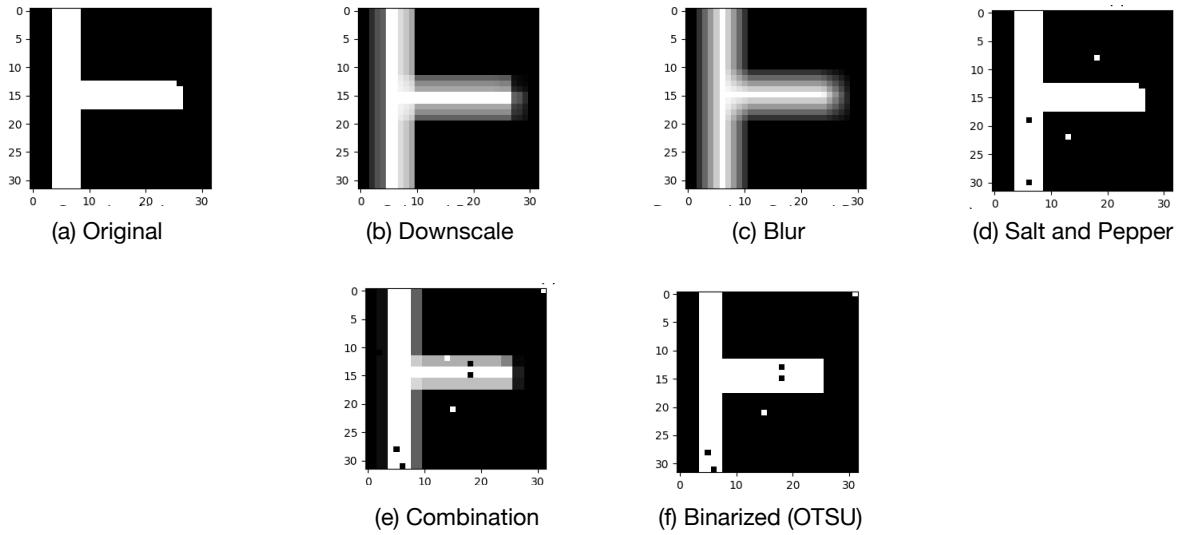


Figure 6.2: Visual noise applied to a primitive query window. (a) Original primitive query window (b) Downscaling reduces resolution to simulate low-quality scans (c) Gaussian blur simulates defocus and smudging (d) Salt-and-pepper noise introduces random pixel corruption (e) Combination of all three transformations (f) Final binarized output using Otsu's thresholding

It also allows for finer-grained control over where noise occurs. This design aligns with Baird's multi-scale degradation model [11], which emphasizes that noise often arises at the symbol or sub-symbol level.

The parameter ranges for each transformation were informed by prior work in chemical diagram recognition, including MolScribe [100], MolGrapher [79], OCMR [134], Img2Mol [25], and MolMiner [145], which incorporate various visual degradation models to increase robustness. We reviewed both the papers and their open-source implementations to determine typical parameter values, then expanded those ranges slightly to allow broader variability during training. Each visual feature window is independently subjected to the following stochastic visual transformations during training (see Figure 6.2):

1. **Resolution Degradation:** To simulate low-resolution imaging, compression artifacts, and optical blur, one of the following is randomly applied:
    - **Downscaling:** The image is resized using a randomly selected scale factor between 0.2 and 0.5 with cubic interpolation.
    - **Gaussian Blur:** Applied with a randomly selected kernel size (k) of 3×3 or 5×5 to emulate defocus or scanner blur. The standard deviation σ for Gaussian blur is not

manually specified and is inferred from the kernel size using:

$$\sigma = 0.3\left(\frac{k-1}{2} - 1\right) + 0.8$$

For example, $\sigma \approx 0.8$ for a 3×3 kernel and $\sigma \approx 1.1$ for a 5×5 kernel. Larger kernels result in stronger blurring.

2. **Salt-and-Pepper Noise:** Simulates transmission defects and binarization artifacts by randomly flipping pixel values with a corruption rate of $p = 0.01$.

Each of these transformations is applied during training with an independent probability of 0.5. For blur and downscaling, one of the two is selected at random when applied (i.e., they are mutually exclusive) since they have similar effects, while salt-and-pepper noise is applied independently. This setup allows for combinations of degradations across different windows, introducing diverse noise patterns within each training batch.

After applying the noise transformations, we perform *Otsu's binarization* [95]. This step ensures consistency with the downstream inference pipeline, which operates on binarized images of individual primitives. Otsu's method assumes a bimodal intensity histogram and determines an optimal threshold t^* that minimizes the *within-class variance* $\sigma_w^2(t)$, which reflects how well the image is separated into foreground (class 1) and background (class 2). It is defined as:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Here, $P(i)$ denotes the normalized histogram i.e., the probability of pixel intensity i, and $q_1(t) = \sum_{i=1}^{t} P(i)$ is the probability of the foreground class. The corresponding mean and variance of the foreground are:

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)}, \quad \sigma_1^2(t) = \sum_{i=1}^{t} \frac{[i - \mu_1(t)]^2 P(i)}{q_1(t)}$$

The terms for the background class (class 2), i.e., $q_2(t)$, $\mu_2(t)$, and $\sigma_2^2(t)$, are defined similarly over the range $i = t + 1$ to I, where I is the maximum intensity level (255 for 8-bit images).

The algorithm evaluates $\sigma_w^2(t)$ for all thresholds $t \in [1, I]$ and selects the threshold t^* that minimizes this value. This results in the best class separation between background and foreground in terms of intensity similarity.

Once t^* is identified, binarization is performed as:

$$\text{pixel}(x, y) = \begin{cases} 0, & \text{if intensity}(x, y) < t^* \quad \text{(background)} \\ 255, & \text{otherwise} \quad \text{(foreground)} \end{cases}$$

In summary, our noise augmentation pipeline operationalizes Baird's principles [11] by:

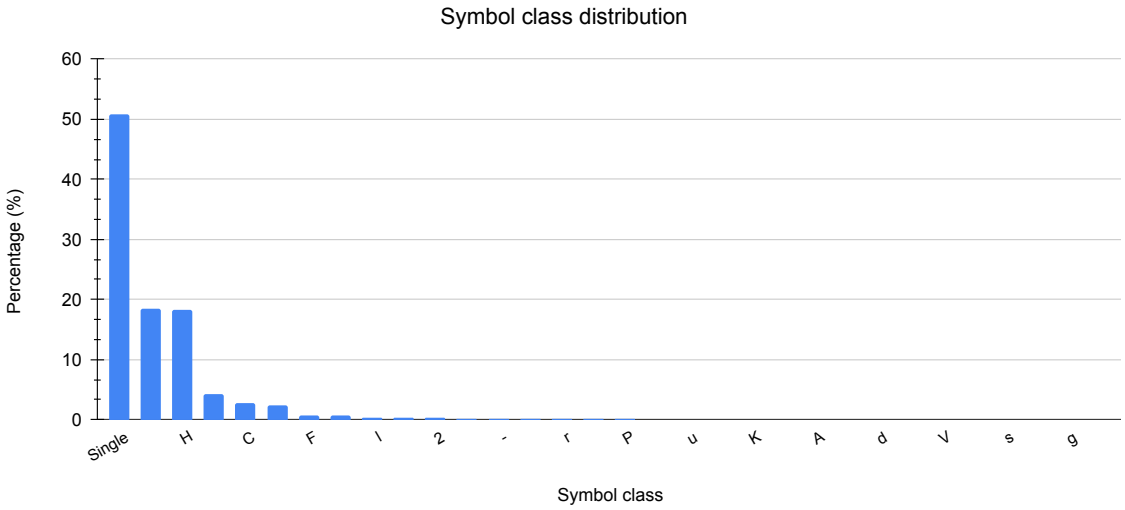
1. Simulating degradations that reflect real physical and digital processes,
2. Applying them for individual visual primitives, including both query and context windows so that each symbol candidate experiences realistic local perturbations, which is crucial for learning fine-grained distinctions in multi-task settings involving segmentation, classification, and relationship prediction,
3. Validating their effectiveness through their impact on downstream parsing performance, as discussed in Section 6.5.1.

6.2 Class Imbalance and Sampling Strategies

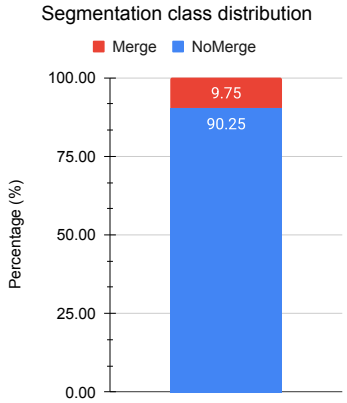
The chemical diagram and math formula parsing datasets exhibits substantial class imbalance within each of the three learning tasks—segmentation, symbol classification, and relationship classification—as well as across task types. For each task, the distribution of class labels are highly skewed: a small number of classes dominates the majority of instances, while many classes are underrepresented or rare. This intra-task imbalance affects all datasets and is particularly pronounced in the chemical domain. In addition, differences in the relative sizes of the task-specific label sets (e.g., the number of symbol nodes vs. edge pairs) introduces further imbalance at the inter-task level. Together, these issues pose challenges for training stability and generalization, especially for rare classes and multi-task interactions. These issues motivate the need for better sampling strategies, class-weighted loss functions and balanced loss aggregation to ensure balanced optimization across tasks and improved learning for underrepresented classes.

6.2.1 Imbalance in Class Distributions

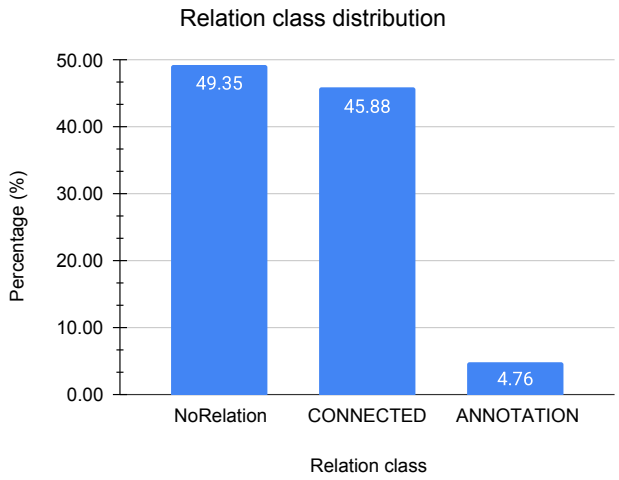
We observe significant class imbalance across both mathematical and chemical diagram datasets. In the chemical domain, particularly within the PubChem-5k training set using the $\mathbf{VGP_{6NN}}$



(a)



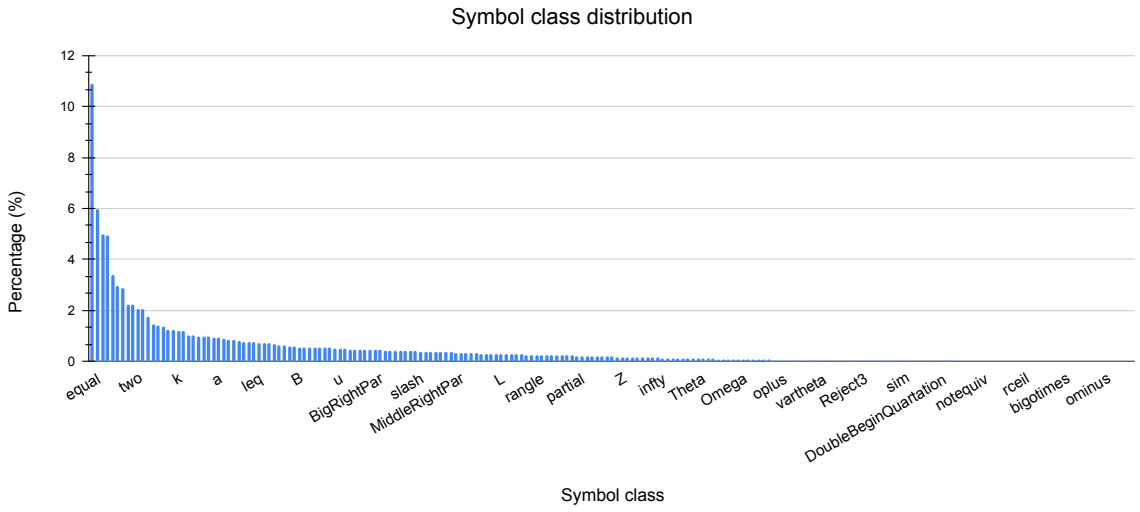
(b)



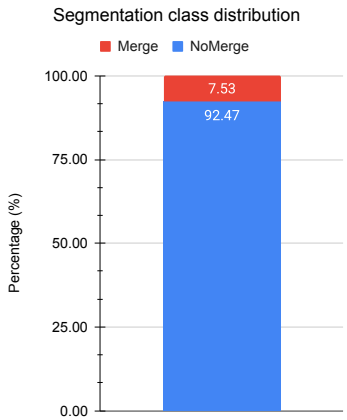
(c)

Figure 6.3: Class distributions in the chemical dataset (PubChem-5k), across symbol, segmentation, and relationship tasks: (a) symbol classes showing heavy skew; (b) segmentation labels dominated by `NoMerge`; (c) relationship edge labels dominated by `NoRelation`.

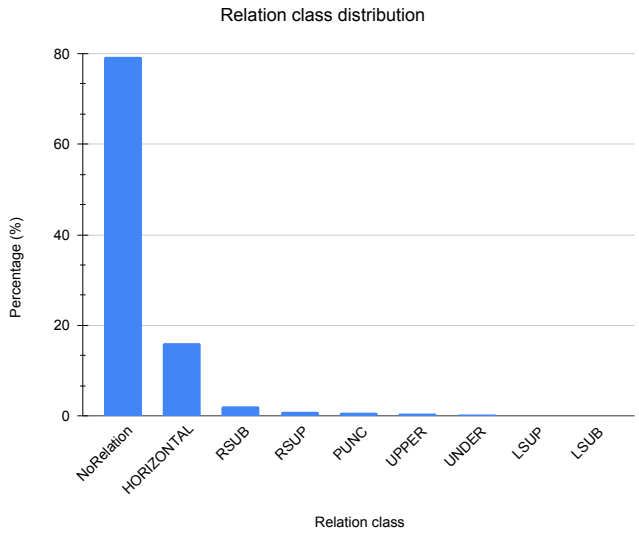
graph, symbol labels exhibit extreme skewness: over 99% of node labels belong to fewer than 10 of the 67 available classes, with a single class—`Single` bond line constituting more than half of all node instances (see Figure 6.3a). Segmentation edges show similarly severe imbalance, as negative segmentation labels (`NoMerge`) represent over 90% of all edges (see Figure 6.3b). The `NoRelation`



(a)



(b)



(c)

Figure 6.4: Class distribution in the math dataset (InftyMCDB-2), across symbol, segmentation, and relationship tasks: (a) symbol classes showing high imbalance; (b) segmentation labels dominated by `NoMerge`; (c) relationship edge labels dominated by `NoRelation`.

and `CONNECTED` labels are relatively balanced (see Figure 6.3c), as the nearest neighbor constraint filters out a large portion of `NoRelation` edges. In contrast, `ANNOTATION` edges account for less than 5% of the total.

In mathematical formulas from the InftyMCDB-2 training set (12,551 formulas), represented using the $\mathbf{VGP_{LOS}}$ graph, dominant classes such as common operators and digits account for more than half of all symbol nodes. Rare symbols, including special characters like `iota` and `bigoplus`, occur with extremely low frequency—comprising less than 0.001% of total nodes (see Figure 6.4a). Similarly, the edge labels are predominantly negative, with approximately 80% labeled as `NoRelation` and over 92% marked as `NoMerge` in segmentation tasks (see Figure 6.4c, 6.4b).

These pronounced imbalances pose significant challenges, requiring strategies to improve learning and generalization across all tasks and class distributions.

Notably, the symbol frequency distributions exhibit a Zipf-like shape [98], where a few classes dominate and many occur rarely, a phenomenon commonly observed in natural language and symbolic systems. In a Zipfian distribution, the frequency f of the r-th most common class is approximately inversely proportional to its rank:

$$f(r) \propto \frac{1}{r^s}$$

where r is the rank of a class by frequency and s is a positive constant (often close to 1 in empirical settings). This long-tailed structure amplifies the difficulty of training robust models for rare classes, especially in multi-task settings where shared representations may be dominated by frequent classes unless explicitly rebalanced.

6.2.2 Stratified Train-Validation Splits

To ensure balanced and representative model validation during training, we use a stratified splitting procedure [87] when partitioning the dataset into training and validation sets. Unlike standard random splits, which can inadvertently amplify class skew, this strategy aims to preserve the co-occurrence structure of labels across the three tasks at the level of individual formulas. It ensures that formulas in training and validation sets are similar in complexity and have similar class distributions across the three tasks, by preserving how these labels appear together within individual formulas.

For example, formulas containing many symbols with two or more oversegmented primitives consist of a higher proportion of merge labels, and formulas with complex spatial layouts or more symbols have more positive relationship labels. Preserving these co-occurrence patterns ensures that the model is evaluated on a validation set with a comparable structural and semantic composition to the training set. This reduces sampling bias and results in more reliable estimates of generalization, especially for tasks that involve interrelated decisions across multiple types of outputs.

We stratify formulas based on a three-dimensional vector of discretized statistics. Each formula f is assigned a bin index along three axes: the number of symbol nodes, the number of edges labeled `MERGE`, and the number of edges labeled `NoRelation`. For example, a formula with many symbols and mostly negative merge and relation edges would be placed in a stratum representing "high node count, high no-merge, high no-relation." These counts are discretized into $n = 4$ quantile-based bins along each axis, resulting in up to $4^3 = 64$ possible strata.

The stratum assignment is defined as a tuple

$$s(f) = (b_{\text{sym}}(f),\ b_{\text{seg=0}}(f),\ b_{\text{rel=0}}(f))$$

where each $b_i(f) \in \{0, 1, 2, 3\}$ denotes the quantile bin index for statistic i. We define the final stratum assignment $z(f)$ as a concatenated tuple of the bin indices:

$$z(f) = \text{concat}\,(b_{\text{sym}}(f),\ b_{\text{seg=0}}(f),\ b_{\text{rel=0}}(f))$$

Strata that contain only a single formula are excluded from the validation set and assigned to the training set to avoid sampling issues with stratified splitting. Additionally, sparsely populated strata, defined as those with fewer than two formulas, are excluded, and further merging is performed when the number of strata exceeds the number of available validation samples. Specifically, the smallest strata are iteratively merged until the number of strata is at most one more than the intended number of validation formulas. This ensures that each stratum is sufficiently populated for stratified sampling and maintains a representative distribution across structure size, segmentation sparsity, and relation sparsity.

We then perform a stratified shuffle split by sampling formulas proportionally from each stratum z, preserving the joint distribution of structure size and class sparsity across symbol, segmentation, and relationship tasks. For each stratum $z \in \mathcal{Z}$, if the total number of formulas is n_z, we allocate approximately $(1 - r) \cdot n_z$ formulas to the validation set and the remaining $r \cdot n_z$ formulas to the training set, where r is the training ratio (e.g., $r = 0.8$).

The actual assignment of formulas is performed using random sampling without replacement within each stratum, ensuring that the relative frequency of each stratum is preserved across both subsets. This procedure maintains balance in the class and structural distributions, resulting in training and validation sets that are statistically similar in composition. Such stratified sampling supports more reliable model selection and evaluation by reducing sampling variance and ensuring coverage of rare but important cases.

6.2.3 Weighted random sampling

To address class imbalance within each of the three tasks, we adopt a weighted sampling approach in which training instances are selected with probabilities proportional to their inverse class frequencies. This method prioritizes underrepresented classes during training, thereby improving model exposure to rare examples and promoting balanced learning.

Following prior work on class-balanced sampling strategies [27], we assign a weight to each node or edge instance based on the frequency of its associated class. We first compute class weights for all classes for each of the tasks based on the frequency of samples in the entire dataset. For a given task, the unnormalized class weight $\tilde{w}_i$ for class i, which has n_i labeled instances in the dataset, is computed as:

$$\tilde{w}_i = \frac{N}{Cn_i} \tag{6.1}$$

where $N = \sum_j n_j$ is the total number of labeled instances across all classes in that task, C is the number of classes for the task. For the symbol classification task, N corresponds to the number of nodes, while for segmentation and relationship classification tasks, N corresponds to the number of edges in the dataset.

To prevent distortion of the overall loss magnitude, the weights are normalized such that their mean across all classes is equal to one [27]. Let C denote the total number of classes in the task. The normalized weight w_i for a task is then given by:

$$w_i = \frac{\tilde{w}_i}{\frac{1}{C}\sum_{j=1}^{C} \tilde{w}_j} \tag{6.2}$$

Here, the normalization ensures that the average class weight is one across the entire training dataset and the total weight sum is equal to the number of classes in the dataset. This preserves the relative importance of each class while maintaining a consistent loss scale across datasets and training configurations. By preventing excessive overweighting of rare classes or underweighting of common ones, it mitigates the risk of vanishing or exploding gradients due to extreme class imbalance. As a result, class weights derived in this manner are well-suited not only for balancing training data through sampling, but also for use as α coefficients to weight the cross-entropy or focal loss, introduced in the next subsection.

This weighting scheme is applied independently to segmentation, symbol classification, and relationship classification tasks. The resulting per-instance weights are used to adjust the sampling

probabilities during mini-batch construction. Instead of uniformly sampling data points, training instances are sampled with probabilities proportional to their assigned weights, allowing underrepresented classes to appear more frequently while reducing the dominance of overrepresented ones. This sampling strategy ensures that the empirical distribution of classes seen during training more closely approximates a balanced distribution, thereby improving generalization and reducing bias toward majority classes.

Such weight-based sampling has been shown to be effective for addressing class imbalance in deep learning, particularly in convolutional networks [18]. Mathematically, let $\{w_i\}_{i=1}^{N}$ denote the normalized per-instance weights for a dataset of N samples. A training mini-batch is then constructed by drawing indices $i \in \{1, \ldots, N\}$ independently from the categorical distribution using sampling *with replacement*, defined by these weights:

$$\mathbb{P}(i) = \frac{w_i}{\sum_{j=1}^{N} w_j}$$

6.3 Loss functions

In our work, we aim to utilize loss functions to further address class imbalance, and improve task-specific learning within a multi-task framework for parsing mathematical and chemical diagrams. As described in Chapter 4, the multi-task loss is defined over a batch of node queries Q_n and edge queries Q_e as:

$$\mathcal{L}(Q_n, Q_e) = \sum_{q_n \in Q_n} \mathcal{L}_S(q_n) + \sum_{q_e \in Q_e} \left(\mathcal{L}_M(q_e) + \mathcal{L}_R(q_e)\right), \tag{6.3}$$

where $\mathcal{L}_S$, $\mathcal{L}_M$, and $\mathcal{L}_R$ denote the loss terms for (S)ymbol classification, primitive (M)erge or segmentation, and (R)elationship classification, respectively.

All losses including standard cross-entropy, weighted cross-entropy, class-balanced, focal, are used in place of $\mathcal{L}_S$, $\mathcal{L}_M$, and $\mathcal{L}_R$, depending on the experimental setting. The final loss per batch remains as given in Equation 6.3, with $\mathcal{L}_S$, $\mathcal{L}_M$, and $\mathcal{L}_R$, instantiated as any of the loss variants described below based on the configuration.

A summary of the final formulations for each loss function is provided in Table 6.1 to facilitate direct comparison of their components and weighting strategies across tasks. We describe each of them in detail below.

Table 6.1: Summary of Loss Function Variants, Formulations, and Parameter Settings

Loss Type	**Final Formulation (per query q)**	**Parameter Range**
Cross-Entropy (CE)	$\mathcal{L}^{CE}(q) = -\sum_{i=1}^{C} y_i \log p_i$	–
Weighted Cross-Entropy (WCE)	$\mathcal{L}^{WCE}(q) = -\sum_{i=1}^{C} w_i\, y_i \log p_i$	–
Class-Balanced (CB)	$\mathcal{L}^{CB}(q) = -\sum_{i=1}^{C} w_i^{CB}\, y_i \log p_i$, where $w_i^{CB} = \frac{1-\beta}{1-\beta^{n_i}}$ normalized to mean 1	$\beta \in \{0.9, 0.99, 0.999\}$
Focal Loss (FL)	$\mathcal{L}^{FL}(q) = -\sum_{i=1}^{C} \frac{\alpha_i (1-p_i)^{\gamma} y_i \log(p_i)}{\frac{1}{\lvert\mathcal{B}\rvert}\sum_{j\in\mathcal{B}} \alpha_{t_j}}$, with α_i as normalized class weights w_i	$\gamma \in \{0.5, 1, 2, 3\}$

6.3.1 Cross-Entropy Loss

The standard categorical cross-entropy loss is used as a baseline, where the loss for each task is computed using predicted softmax probabilities and one-hot encoded targets. It is defined for a single query q as:

$$\mathcal{L}^{CE}(q) = -\sum_{i=1}^{C} y_i \log p_i, \tag{6.4}$$

where C is the number of classes, y_i is the one-hot encoded target label for class i, and p_i is the predicted probability for class i, obtained after applying the softmax function to the model's logits:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)},$$

where z_i denotes the logit score for class i.

The total multi-task loss under this setting follows Equation 6.3 with each $\mathcal{L}_t$ term ($t \in \{S, M, R\}$) defined as $\mathcal{L}_t^{CE}$.

6.3.2 Weighted Cross-Entropy Loss

Weighted Cross-Entropy (WCE) loss is a widely used strategy to mitigate the effects of class imbalance in classification tasks. In standard cross-entropy, all classes contribute equally to the loss function, which can result in poor performance on underrepresented classes when class distributions are highly skewed. WCE addresses this issue by assigning a weight w_i to each class, thereby amplifying or attenuating the contribution of each class to the total loss based on its frequency.

The core idea is to give greater importance to rare classes by assigning them higher weights, and to reduce the influence of frequent classes by assigning them lower weights. This forces the model to attend more to misclassified examples from rare classes, helping to prevent the model from being biased toward majority classes.

Mathematically, the WCE loss is defined as:

$$\mathcal{L}^{WCE}(q) = -\sum_{i=1}^{C} w_i \, y_i \log p_i, \tag{6.5}$$

where, w_i are normalized class weights. The weights w_i are computed from inverse class frequencies and normalized as described in Equation 6.2 to ensure a stable loss scale.

The use of WCE is particularly useful when the class distribution exhibits a long-tail behavior, as is often the case in symbol classification, segmentation, and relationship classification in diagram parsing. Without reweighting, the model would tend to optimize primarily for the head (majority) classes, often neglecting the tail (minority) classes, leading to poor generalization on rare categories.

The multi-task loss in Equation 6.3 is instantiated with $\mathcal{L}_S$, $\mathcal{L}_M$, and $\mathcal{L}_R$ corresponding to weighted cross-entropy terms $\mathcal{L}_t^{WCE}$ for each task $t \in \{S, M, R\}$.

6.3.3 Class-Balanced Loss

Class-balanced loss [86] addresses long-tailed class distributions by reweighting the standard cross-entropy loss using the effective number of samples in each class. The effective number E_i captures the diminishing returns of additional samples as class size increases, motivated by the observation that the utility of each new sample grows logarithmically rather than linearly. Unlike inverse-frequency weighting used in weighted cross-entropy loss, which can assign disproportionately large weights to very rare classes, class-balanced loss provides a smoother and more stable adjustment. This promotes better generalization in the presence of severe class imbalance. The effective number is defined as:

$$E_i = \frac{1 - \beta^{n_i}}{1 - \beta}, \tag{6.6}$$

where n_i is the number of samples in class i, and $\beta \in [0, 1)$ is a hyperparameter that controls the degree of weighting.

Typical values for β range from 0.9 to 0.999. Smaller values (e.g., $\beta = 0.9$) provide modest class reweighting, while larger values (e.g., $\beta = 0.999$) apply stronger emphasis on rare classes. In our experiments, we explore multiple settings as shown in Table 6.1, with higher values favored due to the extreme skew observed in class distributions (e.g., Figure 6.3a).

Using this, the unnormalized class-balanced weights are:

$$\tilde{w}_i^{CB} = \frac{1}{E_i} = \frac{1-\beta}{1-\beta^{n_i}}. \tag{6.7}$$

To ensure stable training and a consistent loss scale, these weights are normalized using the same strategy defined in Equation 6.2, yielding normalized weights w_i^{CB}, such that their mean across all classes is equal to one.

The class-balanced loss for a single query, $\mathcal{L}_t^{CB}$ adopts the same formulation as Equation 6.5, with the normalized class weights w_i replaced by w_i^{CB}. The corresponding multi-task loss for a batch follows Equation 6.3, with each task-specific term $\mathcal{L}_t$ (where $t \in \{S, M, R\}$) instantiated as $\mathcal{L}_t^{CB}$ using class-balanced weights.

6.3.4 Focal Loss

Focal loss [66] was originally developed to address class imbalance in dense object detection by reducing the loss contribution from well-classified examples. It modifies the standard cross-entropy loss with a modulating factor $(1 - p_i)^\gamma$, where p_i is the predicted probability for class i, and γ is a focusing parameter that emphasizes harder, misclassified examples.

The modulating factor $(1 - p_i)^\gamma$ down-weights the loss contribution from well-classified examples (where p_i is high) and amplifies it for misclassified ones (where p_i is low). This encourages the model to focus on hard, uncertain predictions, which are typically associated with rare or ambiguous classes, thereby improving robustness in imbalanced training scenarios.

We evaluated focal loss with $\gamma \in \{0.5,\ 1.0,\ 2.0,\ 3.0\}$, covering a spectrum from mild to strong focusing on hard examples (see Table 6.1). Lower values (e.g., $\gamma = 0.5$) reduce the influence of easy examples moderately, while higher values (e.g., $\gamma = 3.0$) sharply down-weight well-classified examples and emphasize misclassified ones. These configurations allow us to explore the balance between robustness to class imbalance and gradient stability.

To address both class imbalance and overconfidence on frequent classes, we adopt the α-balanced

variant of focal loss. This combines focal weighting with normalized class weights $\alpha_i = w_i$, computed using the strategy described in Equation 6.2. The loss for a single query is defined as:

$$\mathcal{L}^{FL}(q) = -\sum_{i=1}^{C} \alpha_i (1 - p_i)^{\gamma} y_i \log(p_i), \tag{6.8}$$

where y_i is the one-hot target label and C is the number of classes.

This formulation builds on the weighted cross-entropy loss (Equation 6.5) by down-weighting confident predictions and focusing more on misclassified or ambiguous inputs. The class weights α_i act similarly to those in class-balanced loss, but the focal factor provides an additional mechanism to reduce bias from easy, high-frequency classes.

To ensure training stability and prevent gradient magnitudes from being dominated by large weights in rare classes, we normalize the focal loss using the mean of class weights in the current batch. Let $\mathcal{B}$ denote the batch of training samples, and α_{t_j} denote the weight for the ground truth class t_j of sample j. The normalized focal loss becomes:

$$\mathcal{L}^{FL}(q) = -\sum_{i=1}^{C} \frac{\alpha_i (1 - p_i)^{\gamma} y_i \log(p_i)}{\frac{1}{|\mathcal{B}|} \sum_{j \in \mathcal{B}} \alpha_{t_j}}. \tag{6.9}$$

The final multi-task focal loss is computed using the general loss aggregation in Equation 6.3, with each task-specific loss term $\mathcal{L}_t$ ($t \in \{S, M, R\}$) defined as $\mathcal{L}_t^{FL}$ using the focal loss formulation in Equation 6.9.

6.4 Loss Aggregation

While the choice of loss function directly affects learning within each individual task, multi-task performance also depends on how the losses from different tasks are aggregated during optimization. In our framework, symbol classification, segmentation (merge detection), and relationship classification are trained jointly, each with a task-specific loss term $\mathcal{L}_t$ for $t \in \{S, M, R\}$. However, naïvely summing these losses can lead to unbalanced training, especially if one task dominates the overall loss due to imbalanced gradient magnitudes.

To address this, we investigate aggregation strategies that aim to balance the gradient contributions across tasks, stabilize multi-task optimization, and encourage uniform performance improvements. Specifically, we evaluate two aggregation approaches for the primary task losses.

The first is a direct summation:

$$\mathcal{L}_{\text{primary}} = \sum_{t \in \{S,M,R\}} \mathcal{L}_t \tag{6.10}$$

where $\mathcal{L}_S$, $\mathcal{L}_M$, and $\mathcal{L}_R$ are the individual losses for symbol classification, segmentation (merge detection), and relationship classification, respectively. This approach treats all tasks equally and assumes their loss scales are comparable. While simple and widely used, this approach may bias training toward tasks with larger loss magnitudes, leading to suboptimal learning for other tasks.

Complemented Harmonic Mean Aggregation. To address disparities in loss magnitudes across tasks and promote balanced attention to difficult and easy tasks, we adopt a *complemented harmonic mean (CHM)* strategy for aggregating the three primary task losses. This method adapts the harmonic mean to emphasize high-loss tasks, counteracting the bias of the standard harmonic mean which naturally prioritizes low-loss terms.

For each task $t \in \{S, M, R\}$, we begin by normalizing the batch loss using the total loss across all three primary tasks:

$$\tilde{\mathcal{L}}_t(Q) = \frac{\mathcal{L}_t(Q)}{\sum_{t' \in \{S,M,R\}} \mathcal{L}_{t'}(Q)}. \tag{6.11}$$

This step ensures that the losses are brought to a common scale and sum to 1. Next, we compute the complemented harmonic mean over the complements of these normalized losses:

$$\mu_{\text{CHM}} = 1 - 3 \cdot \left(\sum_{t \in \{S,M,R\}} \frac{1}{1 - \tilde{\mathcal{L}}_t} \right)^{-1} \tag{6.12}$$

By applying the harmonic mean to $1 - \tilde{\mathcal{L}}_t$, we invert the standard behavior: tasks with higher normalized losses (i.e., harder tasks) now dominate the aggregation. This reversal ensures that high-loss tasks receive more attention during training, counteracting the tendency of traditional harmonic mean to emphasize already well-performing objectives. To preserve the original loss directionality and obtain a total scalar loss, we apply a final complement to the harmonic value as shown in Equation 6.12: To retain scale sensitivity to the absolute task losses, we reintroduce the sum of unnormalized losses as a multiplicative term. The final complemented harmonic mean loss is defined as:

$$\mathcal{L}_{\text{CHM}} = \left[1 - 3 \cdot \left(\sum_{t \in \{S,M,R\}} \frac{1}{1 - \tilde{\mathcal{L}}_t} \right)^{-1} \right] \cdot \sum_{t \in \{S,M,R\}} \mathcal{L}_t, \tag{6.13}$$

This formulation maintains the bounded nature of the aggregation term (between 0 and 1), while adaptively emphasizing tasks with higher normalized losses. By scaling this term with the sum

of raw task losses, we preserve meaningful gradient magnitudes for optimization. The result is a dynamic, scale-aware aggregation strategy that balances training across tasks without requiring manual weighting or hyperparameter tuning.

Unlike the arithmetic sum, which can lead to *disproportionate dominance* by tasks with large absolute losses, the complemented harmonic mean (CHM) moderates this effect by incorporating normalization. This prevents any single task from overwhelming the optimization process, while still directing focus toward underperforming objectives. CHM thus promotes more equitable convergence across tasks, especially in imbalanced or multi-objective settings.

This approach provides an implicit balancing mechanism that adaptively adjusts the relative importance of tasks based on their current loss magnitudes. CHM gives underperforming tasks sufficient attention without entirely suppressing better-performing ones, allowing training to progress more uniformly across tasks. Unlike weighted summation, CHM achieves this without requiring explicit hyperparameter tuning for task weights, making it both principled and practically convenient for multi-task learning.

Our adoption of this strategy is inspired in part by recent work on harmonic loss in single-task settings [9], where Baek et al. propose a novel loss function based on harmonic softmax Their approach promotes scale-invariant and interpretable convergence, and improves data efficiency in low-resource settings. While we do not replace cross-entropy with harmonic loss at the individual task level, we are motivated by the broader insight that harmonic formulations can improve multi-task learning through balanced aggregation. In our case, rather than redefining the loss for each task, we apply harmonic reasoning at the *inter-task aggregation* level.

6.5 Evaluation and Results

This section presents experiments evaluating the methods introduced in the preceding sections, addressing **RQ4**, which asks whether robustness and generalization in visual parsing can be improved through noise augmentation and class imbalance mitigation. We study two complementary interventions: (1) structural and visual noise augmentation during training, and (2) loss reweighting techniques for handling class imbalance across symbol, relationship, and segmentation tasks.

The datasets, evaluation metrics, and implementation setup remain consistent with those described in the previous chapter (Chapter 5). To ensure consistent comparisons, we adopt a fixed baseline configuration of control variables for all experiments, summarized in Table 6.2. This configuration

reflects the best-performing settings identified in the previous chapter, including optimal input graph representations, primitive extraction methods, input feature sizes, spatial pooling regions, backbone encoder, and EGATv2 architecture for each domain.

6.5.1 Noise Augmentation

As described in Section 6.1, we simulate real-world degradations through two types of synthetic noise during training: (1) *structural noise*, (2) *visual noise* and (3) combination of both noise types. Models are trained with and without these augmentations and evaluated on both clean and synthetic noisy test images. To evaluate generalization under degraded conditions, we apply synthetic noise to test samples using the same augmentation procedures as in training. Each primitive window (consisting of both query and context windows) is independently subjected to two types of noise: *structural* noise and *visual* noise, each applied with a probability of 0.5. This means that for each primitive, structural noise (e.g., random splits) and visual noise (e.g., blur, salt-and-pepper) may be applied individually, jointly, or not at all. As a result, the noisy test set contains a heterogeneous mixture of samples—some clean, some with isolated noise, and others with overlapping degradations. Clean test images are also retained to assess any trade-offs in performance due to augmentation.

Discussion. Models trained without augmentation performed well on clean images, achieving high F1-scores across all tasks. However, their performance deteriorated sharply under noisy conditions, with the most pronounced failures in expression-level accuracy, which dropped to zero due to cascading errors in symbol and relationship classification. Structural noise disrupted spatial graph integrity, while visual noise degraded the visual clarity of primitives, compounding the parsing

Table 6.2: Baseline configuration showing control variables for math and chemistry experiments.

Control variables	**Math**	**Chemistry**
Symbol classes	207	67
Edge classes	9	2 (Indigo) / 3 (RDKIT)
Segmentation classes	2	2
Input primitives	CCs	Line primitives
Input graph	LOS graph	6NN graph
Feature size	64×64	32×32
Pooling regions	31 (1, 3H, 3V, 5H, 5V, 7H, 7V)	17 (1, 3H, 3V, 5H, 5V)
Backbone encoder	ResNeSt	ResNeSt
EGATv2 configuration	2 stage 2 hops	2 stage 2 hops

Table 6.3: Impact of noise-augmented training on parsing performance under different test conditions on the InftyMCDB-2 dataset (6,830 formulas). F1-scores are reported for symbols and relationships, and file-level structure and expression (structure + class) accuracy rates are reported.

	Symbols		**Relationships**		**Expressions**	
	Detect.	**+Class**	**Detect.**	**+Class**	**Structure**	**+Class**
Clean test images						
No augmentation	99.31	97.65	97.18	96.45	92.21	87.99
Structure augmentation	99.35	97.61	97.12	96.42	92.40	88.10
Visual augmentation	99.28	97.78	97.25	96.58	92.50	88.30
Structure+Visual augmentation	**99.40**	**97.90**	**97.40**	**96.75**	**92.70**	**88.60**
Synthetic noisy images						
No augmentation	96.02	80.11	78.45	78.32	0.00	0.00
Structure augmentation	98.95	89.32	85.60	85.40	39.70	34.25
Visual augmentation	98.30	95.12	94.80	94.71	62.70	54.23
Structure+Visual augmentation	**99.11**	**96.80**	**96.15**	**96.08**	**78.45**	**71.36**

Table 6.4: Impact of noise-augmented training on parsing performance under different test conditions on the USPTO test set. F1-scores are reported for symbols and relationships, expression-level structure and classification rates, and the percentage of exact SMILES string matches.

	Symbols		**Relationships**		**Expressions**		**Exact SMILES**
	Detect.	**+Class**	**Detect.**	**+Class**	**Structure**	**+Class**	**(%)**
Clean test images							
No augmentation	99.90	99.86	99.02	99.02	52.54	52.02	89.04
Structure augmentation	99.91	99.83	99.00	99.00	52.61	52.09	89.05
Visual augmentation	99.88	99.87	99.05	99.05	52.65	52.20	89.12
Structure+Visual augmentation	**99.92**	**99.88**	**99.06**	**99.06**	**52.68**	**52.31**	**89.20**
Synthetic noisy images							
No augmentation	96.12	82.32	79.40	79.40	0.00	0.00	0.00
Structure augmentation	99.02	90.42	86.20	86.20	26.40	22.30	63.80
Visual augmentation	98.48	96.25	95.00	95.00	35.15	30.45	70.10
Structure+Visual augmentation	**99.18**	**97.12**	**96.28**	**96.28**	**47.20**	**44.55**	**74.95**

difficulty.

Training with structural augmentation alone mitigated these failures to some extent, improving noisy-test symbol and relationship F1-scores by over 7% in math and chemistry, and recovering partial structure and expression correctness. Visual augmentation yielded substantially greater robustness in both domains, improving symbol classification and relationship classification by over 14% under noise.

Combined structural and visual augmentation consistently provided the best generalization. In math, noisy-test expression recognition (structure + classification) improved by over 71% compared to the unaugmented model, with less than a 1% drop in clean test accuracy. In chemistry, a similar pattern was observed: expression structure and classification accuracy rose from 0% to 44.55%, and exact SMILES match increased from 0% to 74.95%.

These results suggest that the parser is highly sensitive to noise in both input modalities: graph structure and visual appearance, but can be effectively regularized through targeted augmentations. Notably, visual noise appears to have a greater impact on robustness than structural noise alone. Expression-level correctness serves as a useful summary metric, revealing the compounded effects of upstream recognition errors, while SMILES accuracy provides a downstream validation of graph fidelity in the chemical domain. Overall, joint augmentation yields the most balanced model across both domains, capable of handling clean and degraded inputs with high accuracy.

6.5.2 Class Imbalance

As described in Section 6.2, class imbalance poses a consistent challenge across all three parsing tasks: symbol classification, segmentation, and relationship detection. Skewed class distributions lead to undertraining of rare classes, degrading generalization and stability during learning. To mitigate this, we evaluate several loss functions designed to rebalance gradients by emphasizing underrepresented classes.

We compare the following loss formulations: standard cross-entropy (CE), weighted cross-entropy (WCE), class-balanced loss (CB), and focal loss (FL). Each loss is applied uniformly across the symbol, segmentation, and relationship heads, with task-level losses combined using the default summation strategy. Note that all models in this experiment are trained with the combination of both structural and visual noise augmentations, as described in the previous setup. However, evaluation is performed exclusively on clean test images to isolate the effects of class imbalance handling.

Discussion. Among the reweighting strategies, class-balanced (CB) loss consistently improved performance across tasks in both mathematical and chemical domains. Performance improved monotonically with higher values of β: the strongest gains were observed with $\beta = 0.999$, which outperformed both $\beta = 0.9$ and 0.99 in symbol, relationship, and expression-level metrics. In math, expression structure+class accuracy increased from 88.60% (CE) to 89.40% with $\beta = 0.999$,

Table 6.5: Comparison of different loss functions on parsing performance for the InftyMCDB-2 test data. F1-scores are reported for symbol and relationship prediction, and file-level structure and structure+class accuracy for expressions.

Loss Function	**Symbols**		**Relationships**		**Expressions**	
	Detect.	**+Class**	**Detect.**	**+Class**	**Structure**	**+Class**
Cross-Entropy (CE)	99.40	97.90	97.40	96.75	92.70	88.60
Class-Balanced Loss						
$\beta = 0.9$	99.43	98.00	97.50	96.90	92.85	88.90
$\beta = 0.99$	99.47	98.30	97.80	97.20	93.00	89.20
$\beta = 0.999$	99.50	98.40	97.90	97.25	93.10	89.40
Weighted CE	99.52	98.50	98.00	97.30	93.20	89.90
Focal Loss						
$\gamma = 0.5$	**99.55**	**98.94**	**98.20**	**97.85**	**94.13**	**91.75**
$\gamma = 1$	99.53	98.70	98.05	97.40	93.60	90.10
$\gamma = 2$	99.35	98.20	97.20	96.70	91.50	86.50
$\gamma = 3$	99.25	98.00	96.90	96.30	90.60	84.70

Table 6.6: Comparison of different loss functions on parsing performance for the USPTO test set. F1-scores are reported for symbol and relationship prediction, expression-level structure and classification accuracy, and exact SMILES match percentage.

Loss Function	**Symbols**		**Relationships**		**Expressions**		**Exact SMILES**
	Detect.	**+Class**	**Detect.**	**+Class**	**Structure**	**+Class**	**(%)**
Cross-Entropy (CE)	99.92	99.88	99.06	99.06	52.68	52.31	89.20
Class-Balanced Loss							
$\beta = 0.9$	99.91	99.87	99.08	99.08	52.72	52.36	89.30
$\beta = 0.99$	99.91	99.88	99.10	99.10	52.76	52.40	89.40
$\beta = 0.999$	99.92	99.89	99.11	99.11	52.78	52.45	89.50
Weighted CE	99.92	99.89	99.14	99.14	52.95	52.65	89.70
Focal Loss							
$\gamma = 0.5$	**99.92**	**99.89**	**99.20**	**99.20**	**53.45**	**53.22**	**91.99**
$\gamma = 1$	99.90	99.86	99.14	99.14	53.00	52.70	90.75
$\gamma = 2$	99.85	99.80	99.00	99.00	52.40	51.65	88.60
$\gamma = 3$	99.78	99.74	98.88	98.88	51.85	51.10	87.45

and relationship classification improved from 96.75% to 97.25%. In chemistry, similar gains were observed: structure+class accuracy rose from 52.31% to 52.45%, and exact SMILES match improved from 89.20% to 89.50%. This trend reflects the effect of *extreme class imbalance*, where stronger reweighting is needed to ensure sufficient learning for rare classes.

This pattern continues with inverse-frequency weighting, used in the *Weighted Cross-Entropy (WCE)* setting, which can be interpreted as the limiting case of class-balanced loss as $\beta \to 1$. WCE pro-

vided even better results in both domains, particularly in expression-level metrics: structure+class accuracy rose to 89.90% in math and 52.65% in chemistry, while SMILES accuracy increased to 89.70%. These results suggest that direct inverse-frequency weighting may be more effective under high imbalance, especially when class distributions are extremely skewed.

Focal loss yielded the best overall performance. With a moderate focusing parameter ($\gamma = 0.5$), the model achieved the highest F1-scores for symbol and relationship classification in both datasets. In math, this resulted in a +3.15% improvement in expression+class accuracy over CE; in chemistry, it improved structure+class accuracy to 53.22% and SMILES match to 91.99%—the highest observed. These results highlight focal loss's ability to emphasize informative, harder examples while suppressing overrepresented classes. However, higher values of γ (e.g., $\gamma = 2$ or 3) sharply reduced performance. This is likely because excessive down-weighting of easy examples can destabilize learning and lead to undertraining on frequent but important classes, especially in multi-task settings where the learning signal must remain balanced across segmentation, symbol, and relationship tasks. As γ increases, the model becomes overly focused on rare or ambiguous examples, resulting in poorer generalization and lower expression-level accuracy. For instance, at $\gamma = 3$, structure+class accuracy dropped by 4–5% and SMILES accuracy fell to 87.45% in chemistry.

Overall, *focal loss ($\gamma = 0.5$)* provided the best trade-off between rare-class emphasis and optimization stability, followed by *WCE* and *CB loss with* $\beta = 0.999$. These findings confirm that under extreme imbalance, stronger reweighting—either via high-β CB loss or inverse-frequency weighting—substantially improves parsing accuracy, particularly at the expression level, by improving learning on rare symbols, relationships, and segments. Focal loss offers the combined benefits of class-aware reweighting and difficulty-based modulation. The base term implicitly incorporates a class-balancing effect (through normalization across predictions), while the modulating factor $(1 - p_t)^\gamma$ selectively emphasizes hard, misclassified examples. This dual mechanism enables focal loss to adaptively prioritize both minority classes and ambiguous samples, making it particularly well-suited for multi-task visual parsing under heavy class imbalance.

Error Analysis (Math). Figures 6.5 and 6.6 provide a comparison of node- and edge-level error patterns between models trained with cross-entropy (CE) loss and focal loss ($\gamma = 0.5$) on the InftyMCDB-2 test set. At the node level (Figure 6.5), the CE model shows frequent misclassifications of visually similar symbols such as `minus` and `equal, equivalence`, and `dot` and `ldots, comma, semicolon`. The focal loss model reduces the overall frequency of these errors, suggesting improved disambiguation through targeted weighting of difficult examples.

(a) CE loss

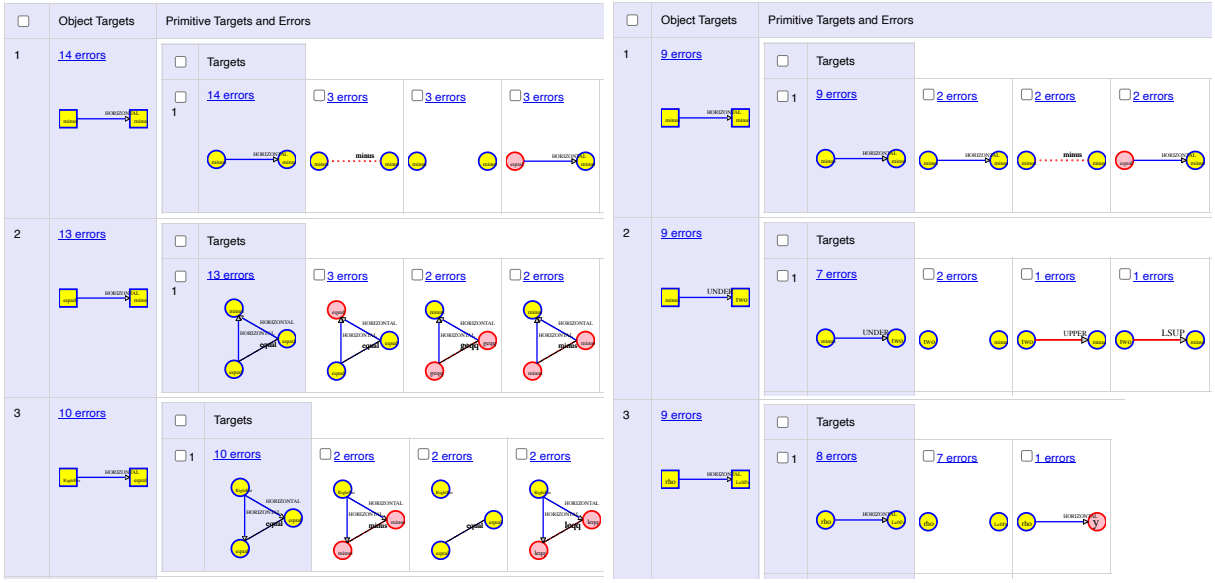
(b) Focal loss, $\gamma = 0.5$

Figure 6.5: Node-level error analysis comparing two models (see Table 6.5) on the InftyMCDB-2 test set: (a) trained with standard cross-entropy (CE) loss, and (b) trained with Focal loss, $\gamma = 0.5$ (truncated at right for space). The Focal loss variant shows reduced frequency of common misclassification errors, particularly for similar symbols such as `minus` and `equal`

This reduction in symbol-level confusion propagates to edge-level improvements as seen in Figure 6.6. Errors related to `HORIZONTAL` relationships between `minus` primitives or between `equal` components are frequent in the CE model due to the symbol-level ambiguity. The focal loss model demonstrates fewer such mistakes.

These results align with quantitative findings in Table 6.5, where focal loss achieved the highest F1-scores for both node and edge predictions. The improvement stems from focal loss's ability to suppress overrepresented classes while emphasizing uncertain, misclassified instances.

Error Analysis (Chem). Figures 6.7 and 6.8 show node- and edge-level error distributions on the USPTO test set for models trained with cross-entropy (CE) loss and focal loss ($\gamma = 0.5$). At the node level (Figure 6.7), both models show dominant misclassifications involving the `Single` line primitive and symbols such as `4, N, b`. These confusions arise in part due to the visual simplicity of `Single` lines and high class frequency in the dataset as described in Chapter 5. The model trained with focal loss exhibits a reduction in these errors, especially in the top-ranked



(a) CE loss

(b) Focal loss, $\gamma = 0.5$

Figure 6.6: Edge-level error analysis comparing two models (see Table 6.5) on the InftyMCDB-2 test set: (a) trained with standard cross-entropy (CE) loss, and (b) trained with Focal loss, $\gamma = 0.5$. The Focal loss variant reduces the frequency of edge errors linked to propagated symbol misclassifications, such as confusion between `minus` and `equal`

categories. This improvement can be attributed to the class-balancing behavior implicit in focal loss: by down-weighting the contribution of easy, high-frequency examples like correctly classified `Single` lines, and up-weighting harder, less frequent misclassifications, the model is encouraged to devote more learning capacity to discriminating between ambiguous cases. As a result, it is better able to differentiate between actual `Single` primitives and visually similar strokes belonging to other symbol categories, leading to improved node classification performance under class imbalance.

At the edge level (Figure 6.8), both models struggle with missed connections involving `Single` lines. The most frequent errors correspond to failed detection of bonds between two `Single` primitives or between a `Single` line and atom primitives like `H` or `N`. These errors are often downstream effects of incorrect node predictions, but can also result from local ambiguity in line proximity and direction. The focal loss model shows reduced error counts in the top two edge categories.

However, many of these missing edge connections are ultimately corrected during postprocessing, when bond multiplicity is inferred by merging adjacent `Single` lines into double or triple bonds. This suggests a representational limitation of the primitive-level visual graph: many true edges are not

(a) CE loss

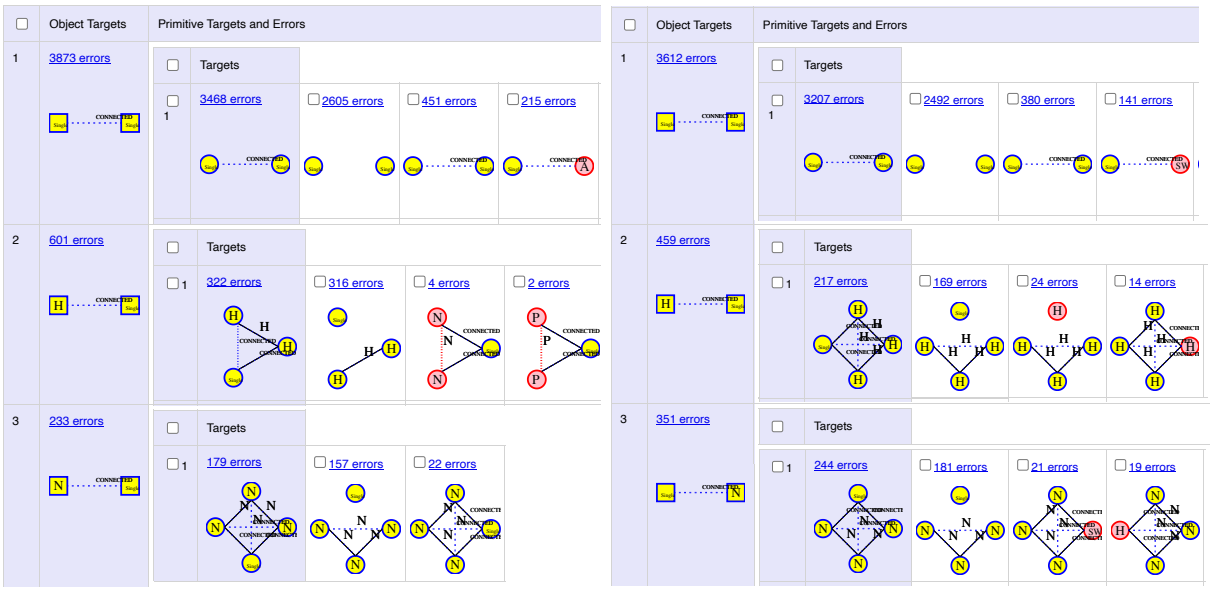
(b) Focal loss, $\gamma = 0.5$

Figure 6.7: Node-level error analysis comparison between two models on the USPTO test set: (a) trained with standard cross-entropy (CE) loss, and (b) trained with Focal loss, $\gamma = 0.5$ (see Table 6.6). `Single` line and `'4'` are the most frequent sources of error in both models; however, the Focal loss model reduces the errors slightly.

explicit in the prediction graph, but instead recovered during symbolic graph construction. Future work could explore models that operate on a more semantically-aligned graph structure, closer to the final merged molecular graph, in order to reduce dependence on postprocessing heuristics and better align the learning signal with the final output structure.

6.5.3 Loss Aggregation

We evaluate two strategies for combining task-specific losses: (1) a simple summation and (2) a complemented harmonic mean (CHM), as described in Section 6.4. While summation (Equation 6.11) offers a straightforward baseline, it may allow one task with high loss magnitude to dominate training. In contrast, CHM aggregation (Equation 6.13) normalizes and complements individual task losses to emphasize underperforming tasks and promote balanced convergence.



(a) CE loss

(b) Focal loss, $\gamma = 0.5$

Figure 6.8: Edge-level error analysis comparing models trained with (a) standard cross-entropy (CE) loss and (b) Focal loss, $\gamma = 0.5$ on the USPTO test set (truncated at right for space). In both models (see Table6.6), the most frequent edge errors involve missed connections between pairs of `Single` lines, as well as between `Single` lines and atom primitives such as `H` and `N`. The model trained with Focal loss reduces the frequency of such errors across the top 2 cases.

Table 6.7: Evaluation of loss aggregation strategies—*summation (Sum)* and *complemented harmonic mean (CHM)*—under different loss function conditions on the InftyMCDB-2 test dataset. F1-scores are reported for symbol and relationship detection and classification, and expression-level structure and structure+class accuracy.

Model	**Aggregation**	**Symbols**		**Relationships**		**Expressions**	
		Detect.	**+Class**	**Detect.**	**+Class**	**Struct.**	**+Class**
CE Loss	Sum	99.40	97.90	97.40	96.75	92.70	88.60
	CHM	99.50	98.70	98.05	97.65	93.80	90.80
Focal Loss ($\gamma = 0.5$, $\alpha = 1$)	Sum	99.51	98.80	98.00	97.60	93.30	89.00
	CHM	99.53	98.91	98.17	97.78	94.05	91.55
Focal Loss ($\gamma = 0.5$, $\alpha = w$)	Sum	99.55	98.94	98.20	97.85	94.13	91.75
	CHM	99.24	98.75	97.83	97.56	93.92	90.98

To assess the interaction between loss aggregation and class imbalance handling, we evaluate both aggregation methods under three loss function conditions: (1) standard cross-entropy (CE), (2) focal

Table 6.8: Evaluation of loss aggregation strategies—*summation* and *complemented harmonic mean (CHM)*—on the USPTO test dataset under different loss function conditions. F1-scores are reported for symbol and relationship detection and classification, expression-level structure and structure+class accuracy, and exact SMILES match. The class weights w used for α-balanced focal loss are defined in Equation 6.2.

Loss Function	**Aggregation**	**Symbols**		**Relationships**		**Expressions**		**SMILES**
		Detect.	**+Class**	**Detect.**	**+Class**	**Struct.**	**+Class**	**Exact (%)**
CE Loss	Sum	99.92	99.88	99.06	99.06	52.68	52.31	89.20
	CHM	99.92	99.89	99.13	99.13	53.15	52.80	90.10
Focal ($\gamma = 0.5$, $\alpha = 1$)	Sum	99.92	99.88	99.17	99.17	53.22	52.95	90.90
	CHM	99.92	99.89	99.19	99.19	53.43	53.19	91.97
Focal ($\gamma = 0.5$, $\alpha = w$)	Sum	99.92	99.89	99.20	99.20	53.45	53.22	91.99
	CHM	99.91	99.88	99.18	99.18	53.40	53.18	91.90

loss without class weighting ($\gamma = 0.5$, $\alpha = 1$), and (3) focal loss with normalized inverse-frequency weighting ($\gamma = 0.5$, $\alpha = w$). The class weights w are defined in Equation 6.2.

This evaluation allows us to isolate the effects of aggregation from those of loss-level reweighting. In particular, because CHM is a balancing mechanism, applying it alongside another form of loss balancing, such as α-weighted focal loss, may lead to overcorrection or over-penalization on easy or frequent classes. All experiments are conducted on both the USPTO and InftyMCDB-2 test datasets, with results summarized in Table 6.8.

Discussion. Across both datasets, complemented harmonic mean (CHM) aggregation consistently improves or maintains performance relative to simple summation, especially under standard CE loss and non-α-balanced focal loss settings.

On the USPTO dataset, CHM yields notable improvements in expression-level metrics over summation: for CE loss, it increases structure+class accuracy by +0.49% and exact SMILES match by +0.90%; for focal loss with $\gamma = 0.5, \alpha = 1$, CHM improves structure+class by +0.24% and SMILES by +1.07%. Similar trends are observed in the InftyMCDB-2 results, where CHM improves expression structure+class accuracy by +2.2% over CE with summation and by +2.55% over focal loss ($\alpha = 1$) with summation. These gains demonstrate CHM's ability to dynamically emphasize underperforming tasks, particularly when no explicit class-based loss reweighting is applied.

However, in the α-balanced focal loss condition ($\gamma = 0.5, \alpha = w$), CHM slightly underperforms

compared to summation in expression-level accuracy for both datasets. For instance, on USPTO, the expression+class accuracy drops by 0.04%, and on InftyMCDB-2, it drops by 0.77%. This suggests that combining two balancing mechanisms—CHM and focal loss with strong class weighting—may lead to over-penalization of well-classified examples, increased optimization instability, or reduced generalization.

Also, expression+class accuracy with CHM and unweighted focal loss ($\alpha = 1$) is nearly equivalent to that achieved by summation with weighted focal loss ($\alpha = w$) on both datasets. This suggests that CHM aggregation may act as a substitute for class-based loss weighting by adaptively shifting gradient focus toward underperforming tasks during optimization. In effect, it mimics the role of the α parameter in focal loss without requiring explicit class frequency statistics, thereby reducing the need for an additional parameter. This makes CHM a promising direction for achieving balanced multi-task optimization with fewer tunable components.

These results imply that CHM may be useful not only as a task-level aggregation strategy, but potentially also as an intra-task loss formulation to replace or complement CE and focal losses themselves. Future work could explore CHM-style objectives at the individual task level to unify task and class balancing under a common, adaptive formulation.

Overall, we find that CHM aggregation provides the most benefit in settings without explicit class reweighting and can serve as a lightweight, hyperparameter-free method to enhance task-level balance in multi-objective learning frameworks.

6.5.4 Benchmarks

We evaluate our best-performing model configuration—EGATv2 with two stages, two-hop message passing, contour-based visual primitives, structural and visual noise augmentation, and focal loss with $\gamma = 0.5$ and $\alpha = w$ as defined in Equation 6.2, on several benchmark datasets across math and chemistry domains.

Datasets and Metrics

For mathematical formula parsing, we use both handwritten and typeset sources, while for chemical diagram parsing, we focus on chemical structure recognition from synthetic and scanned images. The datasets and evaluation metrics are summarized below.

(Math) Handwritten Formulas: CROHME 2019. To evaluate handwritten formula parsing, we use the CROHME 2019 dataset [73, 85], which contains expressions captured as online ink data in InkML format. Each InkML file encodes (x, y) stroke sequences with associated symbol groupings, where pen-down and pen-up events define individual strokes. CROHME 2019 provides 9,993 training, 986 validation, and 1,199 test expressions spanning 101 symbol classes and 7 relationship classes: `Right`, `Sup`, `Sub`, `Above`, `Below`, `Inside`, and `NoRelation`.

We render each stroke sequence as a binary image at its original resolution to produce raster inputs compatible with our parsing model. In our pipeline, each stroke is treated as a visual primitive and processed to extract contours using an adaptive dilation-based preprocessing method. Specifically, we compute the tight bounding box of each stroke and render it onto a locally padded canvas. We then apply dilation using a square kernel whose size and number of iterations are determined dynamically. The kernel size k is computed as:

$$k = \text{clip}\left(\max\left(\frac{H}{W}, \frac{W}{H}\right) \cdot \frac{10}{N},\ 2,\ 4\right)$$

and the number of iterations is given by:

$$\text{iterations} = \text{clip}\left(k \cdot \frac{10}{N},\ 2,\ 3\right)$$

where H and W are the height and width of the image, N is the number of primitives, and 10 is a target primitive count. The function $\text{clip}(x,\ a,\ b)$ constrains x to lie within the range $[a, b]$, defined as:

$$\text{clip}(x,\ a,\ b) = \min(\max(x,\ a),\ b)$$

This adaptive dilation helps stabilize contour extraction across varying image shapes and primitive counts. This approach ensures that dilation is stronger for sparse images with fewer primitives and adapts based on shape and resolution, improving the robustness of contour extraction across varied stroke layouts.

We evaluate the models on this dataset using expression level metrics, i.e. expression structure and structure+classification rates.

(Math) Typeset Formulas: Im2Latex-100K. *Im2Latex-100K* [30] is a large-scale dataset of mathematical expressions collected from LaTeX sources in academic papers. It contains 103,556 distinct LaTeX math equations and their corresponding rendered images. The formulas are extracted from over 60,000 LaTeX documents in the 2003 KDD Cup dataset filtered to retain expressions with 40 to 1024 characters. A subset of over 100K formulas that successfully compile in vanilla LaTeX

are rendered using `pdflatex` and converted to 200 DPI PNG images. The final dataset contains 83,883 training, 9,319 validation, and 10,354 test expressions. Each rendered image is a full-page scan (1654×2339 resolution), where the formula may occupy only a small region of the page. Thus, preprocessing is required to crop and normalize the region containing the actual formula before input to the model.

We use our best model trained on the InftyMCDB-2 dataset—the sixth row in Table 6.5, corresponding to Focal loss with $\gamma = 0.5$ and $\alpha = w$, for inference on the Im2Latex-100K test set directly without re-training. This decision was made for two reasons: first, to demonstrate the generalization capacity and data efficiency of our model when trained on a smaller but curated typeset dataset; and second, time constraints and computational cost required for full retraining and hyperparameter tuning on the substantially larger Im2Latex-100K collection.

To evaluate model performance on the Im2Latex-100K dataset, we use the text and image-based metrics defined in the original paper [30]:

1. **BLEU-4 Score**: Measures the n-gram overlap (up to 4-grams) between the predicted LaTeX sequence and the ground truth after normalization [96].

2. **Edit Distance Accuracy**: Computed as

$$1 - \frac{\text{EditDist}(\hat{y}, y)}{\max(|\hat{y}|, |y|)}$$

   where $\hat{y}$ and y denote the predicted and ground truth LaTeX sequences, respectively, and $\text{EditDist}(\cdot)$ refers to the Levenshtein distance—the minimum number of single-character insertions, deletions, or substitutions required to transform one string into the other. This metric captures partial correctness by quantifying the similarity between sequences at the token level, normalized by the length of the longer string.

3. **Image Match Accuracy**: The proportion of samples where the rendered output of the predicted LaTeX sequence ($\hat{x}$) is visually identical to the rendered ground truth image (x). We also report a relaxed version of this metric where differences in whitespace columns are ignored, following [30].

Chemical Diagram Datasets. We evaluate chemical diagram parsing using two public benchmark datasets that cover both synthetic and real document raster images:

- *USPTO (Indigo)* [101]: This dataset consists of 5,719 synthetic images of chemical structures rendered from SMILES strings using the Indigo cheminformatics toolkit. The images are clean and uniformly generated, making the dataset suitable for controlled evaluation of parsing accuracy on noise-free molecular diagrams. There are 37 unique SMILES characters in this dataset.
- *CLEF-2012* [99]: This dataset contains 992 raster images of chemical compounds extracted from scanned US patent documents. It represents a more realistic benchmark with noisy backgrounds, varied rendering styles, and document-level artifacts. As such, it serves to evaluate model generalization to real-world degradation and variability. There are 71 unique SMILES characters in this dataset.

For both datasets, the evaluation metric is **Exact SMILES Match Accuracy**. This metric compares the predicted SMILES string $\hat{s}$ generated from the parsed graph against the ground truth SMILES string s. A prediction is considered correct if the canonicalized forms of $\hat{s}$ and s are identical, accounting for valid structural equivalence despite differences in notation.

Data Efficiency Metrics. To assess model efficiency, we report the amount of training data required to achieve each 1% absolute accuracy gain, measured in molecules per 1% SMILES match accuracy for chemistry, and expressions per 1% BLEU-4 score for math (Im2Latex-100K). This metric provides insight into how effectively each model learns from data and demonstrates our method's ability to generalize with fewer training examples.

Our architecture is also computationally lightweight, with approximately 4.45 million parameters, substantially smaller than transformer-based baselines such as MolScribe [100] and SwinOCSR [146] (each exceeding 88 million parameters), or HybridViT [59] with 51 million parameters, as summarized in Table 5.8. This compact design enables faster training, reduced memory consumption, and greater scalability to resource-constrained or low-data settings.

Results and Discussion

Handwritten Math Formula Parsing (CROHME 2019). Table 6.9 presents benchmark results on the CROHME 2019 test set. Our proposed model (**EGATv2**) achieves an expression recognition rate (ExpRate) of 49.54%, with structure-only recognition accuracy of 73.79%. While these results fall below those of encoder-decoder models such as USTC-iFLYTEK [73] (ExpRate

Table 6.9: Benchmark results on the CROHME 2019 test set. We report expression structure+class rate (ExpRate), which reflects correct structure as well as symbol and relation classes. The columns ≤ 1 and ≤ 2 report expression-level accuracy allowing at most 1 and 2 symbol or relation errors, respectively. The final column (Structure) measures structure-only recognition accuracy, irrespective of symbol and relation labels.

Model Category	Model	Expressions (Structure + Class)			Expressions
		ExpRate	≤ 1	≤ 2	(Structure)
Rule-based	MyScript	79.15	86.82	**89.82**	90.66
	Samsung	79.82	87.82	89.15	89.32
	TUAT	56.88	71.89	76.31	70.73
Encoder-decoder	MathType	60.13	74.40	78.57	79.15
	USTC-iFLYTEK	**80.73**	**88.99**	90.74	**91.49**
	PAL-v2	62.55	74.98	78.40	79.15
	ICAL	60.51	78	84.63	–
Graph outputs	QD-GGA	40.65	60.01	64.96	60.22
	GETD	54.13	–	–	–
	GGM-EGAT	60.72	71.14	76.73	83.74
	Ours (EGATv2)	49.54	66.17	73.92	73.79

80.73%) and MyScript [73] (79.15%), our model is competitive with prior graph-based systems, outperforming QD-GGA [72] and approaching the performance of GETD [122] and GGM-EGAT [141].

The performance gap can be reduced by specifically tuning hyperparameters for the CROHME dataset. We use the same architecture and hyperparameters optimized for the InftyMCDB-2 dataset, which consists of clean binary typeset mathematical formulas rendered from LaTeX. In contrast, CROHME images are derived from online handwritten ink traces, exhibiting greater variability in stroke thickness, placement, and spacing. These changes in the proximity and overlapping nature of handwritten strokes may require feature sizes and pooling regions to be re-tuned for the CROHME dataset.

In our error analysis, we observe that many recognition errors arise from visually similar symbol confusions such as '`l`' vs. '`1`', '`i`' vs. '`j`', or '`a`' vs. 'α'. These errors suggest that incorporating additional contextual features could improve disambiguation.

As a future direction, our visual primitive extraction pipeline could be applied on CROHME dataset to derive line-based primitives from handwritten images. This could enable the parser to learn localized representations while preserving the benefits of structure-aware graph learning.

Table 6.10: Benchmark results on the Im2LaTeX-100K dataset. We report edit distance accuracy (Edit %), BLEU-4 score, image-level exact match (Image Match), and whitespace-insensitive match (Image Match -ws). Data efficiency is measured as the number of training samples required per 1% absolute BLEU-4 score (lower is better).

Model Category	Model	Edit %	BLEU-4 %	Image Match %	Image Match (-ws) %	Data per 1% BLEU-4
Encoder-decoder	IM2TEX	–	87.73	77.46	79.88	956.15
	WYGIWYS	88.6	90.3	78.6	–	928.93
	Coarse-to-Fine	87.32	87.07	78.1	–	963.39
	DenseNet	91.57	88.25	79.1	–	950.51
	ConvMath	90.8	88.33	**83.41**	–	949.65
	EDPA	91.39	92.31	82.07	–	908.71
	DNN-RL	**96.5**	94.1	84.2	–	891.42
	MI2LS	92.3	90.3	82.3	84.8	928.93
	HybridViT	–	89.9	–	**86.5**	933.07
	MathNet	94.7	**94.5**	63.4	–	887.65
Graph-based	INFTY	53.82	66.65	15.6	26.66	1258.56
	GNN	–	90.2	81.8	–	929.96
	Im2Latex-GNN	–	90.19	81.82	–	930.07
	Ours (EGATv2)	55.23	46.05	19.7	27.49	**217.15**

Typeset Math Formula Parsing (Im2Latex-100K). Table 6.10 presents benchmark comparisons on the Im2Latex-100K dataset. While EGATv2 does not compete with the state-of-the-art (SOTA) accuracy levels, it demonstrates good data efficiency with the lowest number of training samples per 1% BLEU-4 gain. This suggests potential for scalable training under low-resource constraints.

There are several reasons for the current performance gap. First, our model was not trained on the Im2Latex-100K training set, but rather on the significantly smaller Infty-MCDB2 dataset, which differs in both resolution and symbol set. Second, images in Im2Latex-100K are rendered as full-page scans and then cropped to isolate formulas, resulting in much lower resolution formula regions, typically 8–10 times smaller than the images used in our training. Third, some math symbols appearing in Im2Latex-100K are absent from our training data, leading to out-of-vocabulary prediction errors. Lastly, during preprocessing, around 30% of LaTeX strings failed normalization, introducing syntactic variations that are penalized under string matching metrics despite being semantically equivalent.

Despite these challenges, EGATv2 achieves a competitive data efficiency score of 217.15 samples per 1% BLEU-4—substantially lower than transformer-based encoder-decoder baselines that re-

Table 6.11: Molecular Diagram Recognition Benchmarks. Percentages of predicted SMILES matching ground truth (Exact SMILES Match %) are shown. For USPTO, PNG images are rendered using Indigo. The final column reports data efficiency as the number of training molecules (in thousands, K) required to achieve 1% absolute SMILES match accuracy (Lower is better).

		Synthetic Image (USPTO)		Scanned Image (CLEF-2012)	
Model category	Model	SMILES Acc.	#Mols / 1% Acc (K)	SMILES Acc.	#Mols / 1% Acc (K)
Rule-based	MolVec 0.9.7	95.40	–	83.80	–
	OSRA 2.1	95.00	–	84.60	–
	Imago 2.0	–	–	68.20	–
Neural Network	Img2Mol	58.90	25.47	48.84	30.71
	DECIMER	69.60	–	62.70	–
Graph Outputs	OCMR	–	–	65.10	5.38
	SwinOCSR	74.00	60.81	30.00	150.00
	Image2Graph	–	–	51.70	137.33
	MolScribe	97.50	17.23	88.90	18.90
	MolGrapher	–	–	90.50	3.31
	Ours (EGATv2)	91.99	**0.04**	39.61	**0.09**

quire over 900 examples per point. This highlights the model's capacity to learn structure-aware representations with relatively limited data. With further improvements, such as incorporating domain-specific normalization, increasing training resolution, expanding the symbol set, and enhancing context-aware features in EGATv2—the model is expected to have closer metrics with state-of-the-art approaches in this benchmark.

Chemical Diagram Parsing (USPTO and CLEF-2012). Table 6.11 reports benchmark results on both synthetic (USPTO) and real scanned (CLEF-2012) chemical diagrams, measured by exact SMILES match accuracy. Our model (EGATv2) achieves strong performance, particularly in data efficiency. On the synthetic USPTO dataset, it reaches 91.99% SMILES accuracy, with only 40 molecules required per 1% accuracy gain, outperforming all prior neural and graph-based systems by a large margin. On CLEF-2012, which consists of real scanned patent diagrams, the SMILES accuracy is lower at 39.61%, but the data efficiency remains high (90 molecules per 1%).

Our model uses a compact architecture with 4.45 million trainable parameters, substantially fewer than transformer-based models like MolScribe [100], SwinOCSR [146], or MolGrapher [79], which range between 88 million and 200 million parameters. This highlights the efficiency of our graph-based parsing approach, both in terms of training data and model size.

The primary source of errors in CLEF images stems from our current limitations in handling non-structural symbols such as reaction arrows, circular annotations, or free-form text (e.g., captions, labels). Our postprocessing pipeline does not currently filter or interpret them. As a result, predicted SMILES strings may contain valid molecule graphs appended with extra tokens, which are treated as incorrect during exact match evaluation, even when the part of the molecule structure is correct.

Additionally, some errors on both datasets are caused by the lack of sufficient context during prediction. For instance, short horizontal or vertical lines are misclassified as `single` line symbol class when they are part of character atoms like `H, N, R`. In future work, we plan to extend EGATv2 to incorporate explicit context feature aggregation in addition to the current query windows. We also aim to improve postprocessing for real scanned diagrams by introducing symbol filtering, and caption segmentation to create accurate predicted SMILES, especially for real image inputs like CLEF.

6.6 Summary

This chapter addressed **RQ4**, which investigates whether visual and structural noise augmentation, along with class imbalance mitigation, can improve generalization and robustness in multi-task parsing of scientific diagrams.

To simulate real-world document degradations, we introduced a localized noise modeling pipeline that applies *structural noise* through stochastic primitive fragmentation and *visual noise* via blur, downscaling, and salt-and-pepper perturbations. These transformations are applied at the primitive window level and binarized using Otsu's method to align with inference-time preprocessing. The augmentation strategy draws on Baird's degradation framework and captures realistic distortions found in scanned, handwritten, or degraded digital diagrams.

To address the extreme class imbalance in both mathematical and chemical datasets, we implemented stratified train-validation splits, weighted sampling, and a range of loss reweighting strategies. These included Weighted Cross-Entropy (WCE), Class-Balanced (CB) loss with varying β, and Focal Loss (FL) with different focusing parameters γ. These methods reweighted gradients to enhance learning from underrepresented symbols, relationships, and segmentation classes.

Experiments on both the InftyMCDB-2 (math) and USPTO (chemistry) datasets demonstrated that combined structural and visual augmentation significantly improved robustness under noisy conditions, while maintaining high accuracy on clean inputs. The most consistent gains were achieved

with joint augmentation, which improved expression-level accuracy by over 70% in math and boosted exact SMILES match by nearly 75% in chemistry under noise.

Among loss functions, *focal loss with* $\gamma = 0.5$ provided the best overall performance across both domains. It combines class-balancing behavior with a modulating factor that emphasizes harder examples, enabling more effective learning on rare and ambiguous inputs. Class-Balanced loss with $\beta = 0.999$ and WCE also yielded strong improvements, particularly under extreme imbalance. These findings demonstrate that targeted augmentation and adaptive loss design are both critical for building parsers that generalize across real-world document conditions. We also showed that Complement Harmonic Mean (CHM) aggregation of the three task losses can substitute for class-based reweighting in focal loss, reducing hyperparameter dependence while maintaining balanced task optimization.

Finally, we benchmarked our best model configuration, incorporating joint augmentation and focal loss against state-of-the-art systems in both math and chemistry domains across multiple datasets. These included CROHME 2019, and Im2Latex-100K for mathematical formula parsing, and USPTO (Indigo) and CLEF-2012 for chemical diagram recognition. While absolute accuracy was lower than transformer-based models, our parser demonstrated good data efficiency. We additionally introduced a data efficiency metric—training examples per 1% accuracy gain—to highlight the model's ability to generalize from limited data. With approximately 4.45 million parameters, our approach remains significantly smaller and more efficient than typical transformer baselines (often exceeding 40 million parameters), highlighting its practical advantages in resource-limited scenarios.

Chapter 7

Conclusion

This thesis presented our work on graph-based visual parsing of scientific diagrams, focusing on mathematical formulas and chemical structures. The research addressed four core questions spanning input graph representations, input feature design, edge-aware multi-task graph attention, noise and class imbalance mitigation, and loss aggregation strategies. Each component was evaluated through a series of targeted experiments using benchmark datasets in both domains.

We began by exploring effective graph representations (complete, line-of-sight, and k-nearest neighbor) and their impact on parsing performance across domains. We then examined the benefits of our novel robust visual primitive extraction from raster images using line segment detection and watershed algorithm, followed by drawing-based contour features and spatial pyramid pooling for localized representation learning. The core model architecture was extended with a novel Edge Graph Attention Network (EGATv2) that supports message passing across node and edge queries, including multi-hop reasoning and cross-task interaction. To improve generalization and robustness, structural and visual noise augmentation methods were introduced, alongside stratified sampling and loss reweighting techniques for addressing class imbalance. Finally, the study evaluated complemented harmonic mean (CHM) loss aggregation as a means to achieve balanced optimization across tasks without requiring manual reweighting.

This chapter concludes by discussing the key findings across all research questions, highlighting contributions, and discussing their implications. It also reflects on limitations of the current work, proposes future directions, and outlines potential extensions to broader domains and applications.

7.1 Visual Primitive Features (RQ1)

RQ1: *How can primitive features be constructed and updated efficiently to support iterative refinement of segmentation and improve structural parsing?*

To address this research question, we developed segmentation-aware graph parsers that operate on visual primitives such as strokes, CCs and line segments. Our parsers are capable of iteratively refining symbol groupings and updating their associated visual features. The input refinement mechanism dynamically merges visual primitives into higher-order symbol groupings during training and inference. This iterative refinement updates the associated feature representations in a way that preserves the original structure as parsing proceeds [112].

To ensure robustness and generalization, we proposed a unified visual feature design based on primitive-centered query and context windows derived from drawn contours. This approach decouples feature construction from full-image representations, enabling consistent local context modeling across noisy and clean domains. In support of this feature design, we introduced a reliable and efficient method for extracting line-based visual primitives from raster chemical diagrams. This method proved effective at separating over-segmented components and minimizing noise due to overlapping or broken contours. While designed for chemical diagrams, it also generalizes to mathematical formulas, flowcharts, and other visually structured domains.

The hypothesis underlying RQ1 was supported. Primitive-based features with iterative refinement led to improved segmentation accuracy and more stable structural parsing across domains, particularly under conditions of noise, distortion, or ambiguous layout. The methods successfully reduced over-segmentation errors and allowed for more consistent relationship inference, without requiring full-image processing or domain-specific supervision.

Limitations and Future work. The current primitive extraction relies on hand-tuned parameters and assumes relatively clean segmentation boundaries. In highly stylized or degraded images, primitive detection and grouping may fail or require manual adjustment. Future work may involve replacing these heuristics with learning-based primitive extraction pipelines, incorporating confidence-aware refinement strategies. Also, we applied the visual primitive extraction for chemical datasets only. This can be extended to mathematical formulas: both handwritten and typeset to improve the robustness of the parser in dealing with noisy or low-quality raster images.

7.2 Input Graph Representations (RQ2)

RQ2: *What types of input graph representations are most effective for parsing mathematical formulas and chemical diagrams, and do these representations differ between the two domains?*

This research question examined how the structure of input graphs influences parsing performance across domains. In graph-based visual parsing, nodes represent visual primitives and edges capture spatial or contextual relationships. The goal was to identify graph construction strategies that enable accurate structural understanding while avoiding unnecessary noise or sparsity.

To address this, we evaluated and compared several graph types, including complete graphs, line-of-sight (LOS) graphs, and k-nearest neighbor (KNN) graphs, across both mathematical and chemical diagram datasets. For mathematical formulas, LOS graphs were found to be most effective. These graphs allowed long-range connections to symbols that are spatially aligned but visually separated, improving edge recall and enabling better structure recovery. For chemical diagrams, 6NN graphs yielded better results. Unlike LOS, which can connect distant and potentially irrelevant symbols, the 6-nearest neighbor strategy maintained local structural coherence and avoided spurious edges.

Overall, the hypothesis was supported: different domains benefit from different input graph structures. Domain-specific selection of input graphs: LOS for math and 6NN for chemistry, led to more accurate and computationally efficient parsing in both cases. Furthermore, these representations can be generated efficiently and without domain-specific supervision, making them practical for broader use.

Limitations and Future Work. Despite these improvements, fixed graph structures still present a tradeoff between computational cost and structural fidelity as we rely on heuristics for pruning of edges, which may not hold for all datasets or domains. Future work may explore dynamic or learned graph construction, where a neural module predicts which edges are necessary based on visual or semantic cues. Such adaptive graph pruning or construction could provide a better balance between coverage, precision, and runtime efficiency.

7.3 Graph Context and Task Interaction (RQ3)

RQ3: *How can graph context and class distributions be leveraged to improve contextual representation and task interaction in multi-task parsing?*

In this research question, we wanted to enhance contextual representation in graph-based parsing through the integration of class distributions and cross-task signals.

To address this, we introduced EGATv2, a two-stage, edge-aware graph attention network designed for multi-task parsing. EGATv2 allows for bidirectional message passing between node and edge representations and supports multi-hop reasoning. By using class distribution outputs to guide attention, the model achieved improved parsing performance on both math and chemistry datasets.

Limitations and Future Work. Despite these improvements, EGATv2 still faces challenges in disambiguating visually similar but semantically distinct symbols, such as minus and equals signs, or single line bonds versus lines in characters. This indicates that the current mechanism does not provide sufficient contextual depth in difficult cases.

Future work could enhance contextual representation by incorporating more global context into EGATv2's aggregation. One direction is to increase the number of neighbors used in message passing beyond the current two nearest neighbor setting during node and edge aggregation, while using dimensionality reduction techniques such as linear bottleneck layers to control model size. Another direction is to embed explicit geometric or the input context features into the attention mechanism along with the query features, allowing more informed representation of layout structure and symbol similarity. These improvements could help disambiguate tightly clustered or visually confusable elements more effectively.

7.4 Visual Noise and Loss Functions (RQ4)

RQ4: *Can visual and structural noise at the primitive level, together with class balancing strategies, improve the robustness of the visual parser?*

This research question was motivated to improve the robustness of the visual parsing framework under noisy or degraded input conditions. To address this, we implemented a localized noise augmentation strategy inspired by Baird's degradation model [11]. Visual noise—such as Gaussian blur, downsampling, and salt-and-pepper corruption—was applied at the level of extracted primitives, rather than the full image, allowing noise to be targeted and spatially localized. Structural noise was introduced by randomly splitting individual primitives into smaller fragments, creating variation in segmentation structure. We also addressed the class imbalance problem across segmentation, symbol classification, and relationship prediction tasks using weighted focal loss, and aggregating

losses using the complemented harmonic mean (CHM) to to balance gradient contributions across objectives.

These methods collectively improved model robustness, especially on noisy synthetic benchmarks. However, performance on low-resolution datasets such as Im2LaTeX remained suboptimal. Visual noise still caused symbol confusion in difficult cases, particularly between visually similar characters like minus and equals signs. This indicates that further augmentation and improved feature regularization are needed.

Limitations and Future Work. Current augmentation strategies are limited to binary structural splits and a narrow set of visual perturbations. Future work should explore more aggressive structural variations, such as multi-split distortions or more complex transformations, to better match the diversity of noise in real-world images. For visual degradation, simulating scanning artifacts, background clutter, and font irregularities could further enhance robustness.

In terms of loss functions, focal loss showed improvements when using $\gamma = 0.5$, particularly for mitigating class imbalance in segmentation and symbol classification. However, we did not explore smaller values such as $\gamma = \frac{1}{3}$, which could be beneficial in our setting. Since the individual loss values lie between 0 and 1 in the focal loss setting, using $\gamma < 1$ has the effect of amplifying their contribution, rather than suppressing them as in the standard $\gamma > 1$ case. This can stabilize training when class imbalance is severe or when gradients vanish too quickly for easy samples. Lower gamma values may allow better gradient flow and finer calibration across majority and minority classes, especially when combined with reweighting schemes.

Additionally, we propose that complemented harmonic mean (CHM), currently used for aggregating task-level losses, could be extended to operate within class-level loss aggregation. Replacing standard summation in cross-entropy or focal loss with CHM across classes would enable more balanced contributions from rare classes without requiring manual tuning or additional parameters. Further, the contraharmonic mean offers a mathematically similar alternative to CHM but avoids the need for complement operations, which could simplify implementation and improve numerical stability during training.

Finally, incorporating contrastive learning objectives could further enhance the model's ability to distinguish visually similar symbols. By encouraging representations of identical instances to be close in the feature space while maintaining separation from representations of different classes sharing some similarities, contrastive learning can promote more discriminative and structured embeddings.

This approach may help address remaining failure cases, such as confusion between visually similar symbols, by reinforcing class-specific boundaries in the learned representation space.

7.5 Other Future Work

Beyond the specific scope of mathematical and chemical diagram parsing, contributions made in this work have potential for broader impact across other domains in graphics recognition and computer vision.

The data generation pipeline based on visual primitive extraction, particularly line segments and contour-based features can be extended to other structured visual domains such as tables, flowcharts, diagrams in engineering drawings, and scientific charts. These domains often rely on compositional visual units and benefit from primitive-level modeling that avoids reliance on global appearance features. In such settings, the primitive-based augmentation and segmentation-aware parsing framework may help reduce annotation requirements while improving domain robustness.

Another natural extension of this work is to apply the methods developed here to the problem of handwritten chemical diagram recognition, similar to how CROHME has enabled progress in handwritten mathematical parsing. Existing approaches for chemical structure recognition are typically designed for clean, typeset images. However, handwritten chemical diagrams present challenges such as inconsistent stroke styles, ambiguous line connectivity, and irregular symbol placement. The primitive-based representation, coupled with edge-aware graph parsing and refinement mechanisms developed in this work, provides a foundation for addressing these challenges. A standardized benchmark for handwritten chemical diagrams, could further facilitate progress in this area.

The EGATv2 architecture, designed for edge-aware message passing and task interaction, is broadly applicable to any task where input can be represented as a graph with node and edge labels. Potential applications include road network understanding, electric circuit parsing, and medical diagram interpretation, as well as general graph-based representations in vision-language tasks, document layout analysis, and scene graph generation. The two-stage design and class-informed attention strategy can support cross-task consistency in multi-objective learning scenarios where predictions must be structurally aligned.

The complemented harmonic mean (CHM) strategy for loss aggregation, originally used for balancing multiple parsing tasks can be generalized to other optimization problems in machine learning where class imbalance or task dominance is a concern. For example, in multi-label classification,

anomaly detection, or multi-domain adaptation, CHM offers an alternative to manual reweighting or reliance on class frequency statistics. By promoting balanced gradient contributions, it provides a tunable-free mechanism for aggregating over tasks or classes, reducing the need for extensive hyperparameter tuning.

These directions open new avenues for extending this work to more diverse and complex visual reasoning tasks, while maintaining effectiveness, efficiency in terms of data and speed, and interpretability.

Chapter 8

List of Publications

Table 8.1: The 5 published papers during my Ph.D.

Title	**Venue**	**Year**
Multimodal Search in Chemical Documents and Reactions **A. K. Shah**, A. Dey, L. Luo, B. M. Amador, P. Philippy, M. Zhong, S. Ouyang, D. M. Friday, D. Bianchi, N. Jackson, R. Zanibbi, and J. Han	International ACM SIGIR Conference on Research and Development in Information (**SIGIR**)	2025
ChemScraper: Leveraging PDF Graphics Instructions for Molecular Diagram Parsing **A. K. Shah**, B. M. Amador, A. Dey, M. Creekmore, B. Ocampo, S. Denmark, and R. Zanibbi	International Journal on Document Analysis and Recognition (**IJDAR**)	2024
Line-of-Sight with Graph Attention Parser (LGAP) for Math Formulas **A. K. Shah** and R. Zanibbi	International Conference on Document Analysis and Recognition (**ICDAR**)	2023
Searching the ACL Anthology with Math Formulas and Text B. M. Amador, M. Langsenkamp, A. Dey, **A. K. Shah**, and R. Zanibbi	International ACM SIGIR Conference on Research and Development in Information Retrieval (**SIGIR**)	2023
A Math Formula Extraction and Evaluation Framework for PDF Documents **A. K. Shah**, A. Dey, and R. Zanibbi	International Conference on Document Analysis and Recognition (**ICDAR**)	2021

Bibliography

[1] Bristol-Myers Squibb - molecular translation competition, 2021. Kaggle.

[2] Nadeem Akhtar and U. Ragavendran. Interpretation of intelligence in CNN-pooling processes: a methodological survey. Neural Computing and Applications, 32(3):879–898, February 2020.

[3] M. Alkalai, J. B. Baker, V. Sorge, and X. Lin. Improving Formula Analysis with Line and Mathematics Identification. In 2013 12th International Conference on Document Analysis and Recognition, pages 334–338, August 2013. ISSN: 2379-2140.

[4] F. Alvaro, J. S´nchez, and J. Benedi. Recognition of Printed Mathematical Expressions Using Two-Dimensional Stochastic Context-Free Grammars. In 2011 International Conference on Document Analysis and Recognition, pages 1225–1229, September 2011. ISSN: 2379-2140.

[5] Bryan Amador, Matt Langsenkamp, Abhisek Dey, Ayush Kumar Shah, and Richard Zanibbi. Searching the acl anthology with math formulas and text. In Proc. ACM SIGIR, page to appear, 2023.

[6] Robert H. Anderson. Syntax-directed recognition of hand-printed two-dimensional mathematics. In Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium, pages 436–459, New York, NY, USA, August 1967. Association for Computing Machinery.

[7] Dan Anitei, Daniel Parres, Joan Andreu Sánchez, and José Miguel Benedí. Improving Efficiency and Performance Through CTC-Based Transformers for Mathematical Expression Recognition. In Elisa H. Barney Smith, Marcus Liwicki, and Liangrui Peng, editors, Document Analysis and Recognition - ICDAR 2024, pages 3–20, Cham, 2024. Springer Nature Switzerland.

[8] Dan Anitei, Joan Andreu Sánchez, José Miguel Benedí, and Ernesto Noya. The IBEM dataset: A large printed scientific image dataset for indexing and searching mathematical expressions. Pattern Recognition Letters, 172:29–36, August 2023.

[9] David D. Baek, Ziming Liu, Riya Tyagi, and Max Tegmark. Harmonic Loss Trains Interpretable AI Models, February 2025.

[10] Gaetan Bahl, Mehdi Bahri, and Florent Lafarge. Single-Shot End-to-end Road Graph Extraction. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pages 1402–1411, June 2022.

[11] Henry S. Baird. The State of the Art of Document Image Degradation Modelling. In Bidyut B. Chaudhuri, editor, Digital Document Processing: Major Directions and Recent Advances, pages 261–279. Springer, London, 2007.

[12] Josef B. Baker, Alan P. Sexton, and Volker Sorge. A Linear Grammar Approach to Mathematical Formula Recognition from PDF. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors, Intelligent Computer Mathematics, Lecture Notes in Computer Science, pages 201–216, Berlin, Heidelberg, 2009. Springer.

[13] Josef B. Baker, Alan P. Sexton, and Volker Sorge. A linear grammar approach to mathematical formula recognition from PDF. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors, 16th Symposium on Intelligent Computer Mathematics, volume 5625 of LNCS, pages 201–216. 2009.

[14] Jonathan Baxter. A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling. Machine Learning, 28(1):7–39, July 1997.

[15] Dorothea Blostein and Ann Grbavec. RECOGNITION OF MATHEMATICAL NOTATION. In Handbook of Character Recognition and Document Image Analysis, pages 557–582. WORLD SCIENTIFIC, May 1997.

[16] Henning Otto Brinkhaus, Kohulan Rajan, Achim Zielesny, and Christoph Steinbeck. RanDepict: Random chemical structure depiction generator. Journal of Cheminformatics, 14(1):31, June 2022.

[17] Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks? In International Conference on Learning Representations, October 2021.

[18] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw., 106(C):249–259, October 2018.

[19] Syed Saqib Bukhari, Zaryab Iftikhar, and Andreas Dengel. Chemical structure recognition (CSR) system: Automatic analysis of 2D chemical structures in document images. In International Conference on Document Analysis and Recognition (ICDAR), pages 1262–1267, 2019.

[20] Daniel Campos and Heng Ji. IMG2SMI: Translating molecular structure images to Simplified Molecular-Input Line-entry System. 2021. arXiv.

[21] Rich Caruana. Multitask Learning. Machine Learning, 28(1):41–75, July 1997.

[22] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. International Journal on Document Analysis and Recognition, 3(1):3–15, August 2000.

[23] Mingjun Chen, Hao Wu, Qikai Chang, Hanbo Cheng, Jiefeng Ma, Pengfei Hu, Zhenrong Zhang, Chenyu Liu, Changpeng Pi, Jinshui Hu, Baocai Yin, Bing Yin, Cong Liu, and Jun Du. ICDAR 2024 Competition on Recognition of Chemical Structures. In Elisa H. Barney Smith, Marcus Liwicki, and Liangrui Peng, editors, Document Analysis and Recognition - ICDAR 2024, pages 397–409, Cham, 2024. Springer Nature Switzerland.

[24] Yufan Chen, Ching Ting Leung, Yong Huang, Jianwei Sun, Hao Chen, and Hanyu Gao. MolNexTR: A Generalized Deep Learning Model for Molecular Image Recognition, August 2024.

[25] Djork-Arné Clevert, Tuan Le, Robin Winter, and Floriane Montanari. Img2Mol – accurate SMILES recognition from molecular graphical depictions. Chemical Science, 12(42):14174–14181, November 2021.

[26] Paolo Comelli, Paolo Ferragina, Mario Notturno Granieri, and Flavio Stabile. Optical Recognition. 44(4):627–631, 1995.

[27] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-Balanced Loss Based on Effective Number of Samples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 9268–9277, 2019.

[28] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Computational Geometry. In Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars, edi-

tors, Computational Geometry: Algorithms and Applications, pages 1–17. Springer, Berlin, Heidelberg, 2008.

[29] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention. In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research, pages 980–989. PMLR, 2017.

[30] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, pages 980–989, Sydney, NSW, Australia, August 2017. JMLR.org.

[31] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. Meaningful Alignments. International Journal of Computer Vision, 40(1):7–23, October 2000.

[32] Agnés Desolneux, Lionel Moisan, and Jean-Michel Morel. From Gestalt Theory to Image Analysis: A Probabilistic Approach, volume 34 of Interdisciplinary Applied Mathematics. Springer, New York, NY, 2008.

[33] Cameron Diao and Ricky Loynd. Relational Attention: Generalizing Transformers for Graph-Structured Tasks. In The Eleventh International Conference on Learning Representations, September 2022.

[34] Yancarlos Diaz, Gavin Nishizawa, Behrooz Mansouri, Kenny Davila, and Richard Zanibbi. The mathdeck formula editor: Interactive formula entry combining latex , structure editing, and search. In CHI Extended Abstracts, pages 192:1–192:5. ACM, 2021.

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In International Conference on Learning Representations, October 2020.

[36] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In Chengqing Zong and Michael Strube, editors, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 845–850, Beijing, China, July 2015. Association for Computational Linguistics.

[37] Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs, January 2021.

[38] Jack Edmonds. Optimum branchings. Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics, 71B(4):233, October 1967.

[39] Y. Eto and M. Suzuki. Mathematical formula recognition using virtual link network. In Proceedings of Sixth International Conference on Document Analysis and Recognition, pages 762–767, September 2001.

[40] Y. Eto and M. Suzuki. Mathematical formula recognition using virtual link network. In International Conference on Document Analysis and Recognition (ICDAR), pages 762–767, 2001.

[41] Igor V. Filippov and Marc C. Nicklaus. Optical structure recognition software to recover chemical information: OSRA, an open source solution. Journal of Chemical Information and Modeling, 49(3):740–743, 2009.

[42] Liyu Gong and Qiang Cheng. Exploiting Edge Features in Graph Neural Networks. arXiv:1809.02709 [cs, stat], January 2019.

[43] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing XU, and Yunhe Wang. Transformer in transformer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems (NeurIPS), pages 15908–15919, 2021.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, Computer Vision – ECCV 2014, Lecture Notes in Computer Science, pages 346–361, Cham, 2014. Springer International Publishing.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[47] Stephen Heller, Alan McNaught, Stephen Stein, Dmitrii Tchekhovskoi, and Igor Pletnev. InChI - the worldwide chemical structure identifier standard. Journal of Cheminformatics, 5(1):7, 2013.

[48] Stephen R. Heller, Alan McNaught, Igor Pletnev, Stephen Stein, and Dmitrii Tchekhovskoi. InChI, the IUPAC International Chemical Identifier. Journal of Cheminformatics, 7(1):23, 2015.

[49] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7132–7141, June 2018.

[50] L. Hu and R. Zanibbi. MST-based Visual Parsing of Online Handwritten Mathematical Expressions. In 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pages 337–342, October 2016. ISSN: 2167-6445.

[51] Lei Hu. Features and Algorithms for Visual Parsing of Handwritten Mathematical Expressions. Theses, May 2016.

[52] Lei Hu and Richard Zanibbi. Line-of-Sight Stroke Graphs and Parzen Shape Context Features for Handwritten Math Formula Representation and Symbol Segmentation. In 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pages 180–186, October 2016. ISSN: 2167-6445.

[53] P. Ibison, M. Jacquot, F. Kam, A. G. Neville, R. W. Simpson, C. Tonnelier, T. Venczel, and A. P. Johnson. Chemical Literature Data Extraction: The CLiDE project. Journal of Chemical Information and Computer Sciences, 33(3):338–344, 1993.

[54] Abin Jose, Ricard Durall Lopez, Iris Heisterklaus, and Mathias Wien. Pyramid Pooling of Convolutional Feature Maps for Image Retrieval. In 2018 25th IEEE International Conference on Image Processing (ICIP), pages 480–484, October 2018. ISSN: 2381-8549.

[55] Ivan Khokhlov, Lev Krasnov, Maxim V. Fedorov, and Sergey Sosnin. Image2SMILES: Transformer-Based Molecular Optical Recognition Engine. Chemistry–Methods, 2(1):e202100069, 2022.

[56] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, page 14, 2017.

[57] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing Embedded Strings (SELFIES): A 100% robust molecular string representation. Machine Learning: Science and Technology, 1(4):045024, 2020.

[58] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 2, pages 2169–2178, June 2006. ISSN: 1063-6919.

[59] Anh Duy Le, Van Linh Pham, Vinh Loi Ly, Nam Quan Nguyen, Huu Thang Nguyen, and Tuan Anh Tran. A Hybrid Vision Transformer Approach for Mathematical Expression Recognition. In 2022 International Conference on Digital Image Computing: Techniques and Applications (DICTA), pages 1–7, November 2022.

[60] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. 2005.

[61] Youngwan Lee, Jonghee Kim, Jeffrey Willette, and Sung Ju Hwang. Mpvit: Multi-path vision transformer for dense prediction. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 7277–7286, 2022.

[62] Bohan Li, Ye Yuan, Dingkang Liang, Xiao Liu, Zhilong Ji, Jinfeng Bai, Wenyu Liu, and Xiang Bai. When Counting Meets HMER: Counting-Aware Network for Handwritten Mathematical Expression Recognition. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, Computer Vision – ECCV 2022, Lecture Notes in Computer Science, pages 197–214, Cham, 2022. Springer Nature Switzerland.

[63] Weijun Li, Yuxiao Gao, Ang Li, Xinyong Zhang, Jianlai Gu, and Jintong Liu. Sparse Subgraph Prediction Based on Adaptive Attention. Applied Sciences, 13(14):8166, January 2023.

[64] Yanchi Li, Guanyu Chen, and Xiang Li. Automated Recognition of Chemical Molecule Images Based on an Improved TNT Model. Applied Sciences, 12(2):680, 2022.

[65] Fan Lin and Jianhua Li. MPOCSR: Optical chemical structure recognition based on multi-path Vision Transformer. Complex & Intelligent Systems, 10(6):7553–7563, December 2024.

[66] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2):318–327, February 2020.

[67] X. Lin, L. Gao, Z. Tang, X. Lin, and X. Hu. Mathematical Formula Identification in PDF Documents. In 2011 International Conference on Document Analysis and Recognition, pages 1419–1423, September 2011. ISSN: 2379-2140.

[68] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 11966–11976, 2022.

[69] Jun Long, Quan Hong, and Liu Yang. An Encoder-Decoder Method with Position-Aware for Printed Mathematical Expression Recognition. In Gernot A. Fink, Rajiv Jain, Koichi Kise, and Richard Zanibbi, editors, Document Analysis and Recognition - ICDAR 2023, pages 167–181, Cham, 2023. Springer Nature Switzerland.

[70] Mahshad Mahdavi. Query-Driven Global Graph Attention Model for Visual Parsing: Recognizing Handwritten and Typeset Math Formulas. Theses, August 2020.

[71] Mahshad Mahdavi, Michael Condon, Kenny Davila, and Richard Zanibbi. LPGA: Line-of-Sight Parsing with Graph-Based Attention for Math Formula Recognition. In 2019 International Conference on Document Analysis and Recognition (ICDAR), pages 647–654, Sydney, Australia, September 2019. IEEE.

[72] Mahshad Mahdavi, Leilei Sun, and Richard Zanibbi. Visual Parsing with Query-Driven Global Graph Attention (QD-GGA): Preliminary Results for Handwritten Math Formula Recognition. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pages 2429–2438, Seattle, WA, USA, June 2020. IEEE.

[73] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchere, Christian Viard-Gaudin, and Utpal Garain. ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection. In 2019 International Conference on Document Analysis and Recognition (ICDAR), pages 1533–1538, Sydney, Australia, September 2019. IEEE.

[74] Behrooz Mansouri, Vít Novotný, Anurag Agarwal, Douglas W. Oard, and Richard Zanibbi. Overview of arqmath-3 (2022): third CLEF lab on answer retrieval for questions on math (working notes version). In CLEF (Working Notes), volume 3180 of CEUR Workshop Proceedings, pages 1–27. CEUR-WS.org, 2022.

[75] MathML. Mathml — Wikipedia, the free encyclopedia, 2024. Online; accessed: 2024-10-15.

[76] Nicholas E. Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, 1999.

[77] Łukasz Maziarka, Tomasz Danel, Sławomir Mucha, Krzysztof Rataj, Jacek Tabor, and Stanisław Jastrzębski. Molecule Attention Transformer, February 2020.

[78] Joe R. McDaniel and Jason R. Balmuth. Kekule: OCR-Optical Chemical (structure) Recognition. Journal of Chemical Information and Computer Sciences, 32(4):373–378, 1992.

[79] Lucas Morin, Martin Danelljan, Maria Isabel Agea, Ahmed Nassar, Valery Weber, Ingmar Meijer, Peter Staar, and Fisher Yu. Molgrapher: Graph-based visual recognition of chemical structures. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 19552–19561, October 2023.

[80] Harold Mouchere, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. Crohme2011: Competition on recognition of online handwritten mathematical expressions. In 2011 International Conference on Document Analysis and Recognition, pages 1497–1500, 2011.

[81] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011-2014. Int. J. Document Anal. Recognit., 19(2):173–189, 2016.

[82] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain. ICFHR 2014 Competition on Recognition of On-Line Handwritten Mathematical Expressions (CROHME 2014). In 2014 14th International Conference on Frontiers in Handwriting Recognition, pages 791–796, September 2014. ISSN: 2167-6445.

[83] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain. ICFHR2016 CROHME: Competition on Recognition of Online Handwritten Mathematical Expressions. In 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pages 607–612, October 2016. ISSN: 2167-6445.

[84] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim. ICDAR 2013 CROHME: Third International Competition on Recognition of Online Handwritten Mathematical Expressions. In 2013 12th International Conference on Document Analysis and Recognition, pages 1428–1432, August 2013. ISSN: 2379-2140.

[85] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. International Journal on Document Analysis and Recognition (IJDAR), 19(2):173–189, June 2016.

[86] Nikhil Nasalwai, Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Addressing the Class Imbalance Problem in Medical Image Segmentation via Accelerated Tversky Loss Function. In Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty, editors, Advances in Knowledge Discovery and Data Mining, pages 390–402, Cham, 2021. Springer International Publishing.

[87] Jerzy Neyman. On the Two Different Aspects of the Representative Method: The Method of Stratified Sampling and the Method of Purposive Selection. Journal of the Royal Statistical Society, 97(4):558, 1934.

[88] An Nguyen, Yu-Chieh Huang, Pierre Tremouilhac, Nicole Jung, and Stefan Bräse. CHEMSCANNER: Extraction and re-use(ability) of chemical information from common scientific documents containing ChemDraw files. Journal of Cheminformatics, 11:77, 2019.

[89] Cuong Tuan Nguyen, Thanh-Nghia Truong, Hung Tuan Nguyen, and Masaki Nakagawa. Global Context for Improving Recognition of Online Handwritten Mathematical Expressions. In Josep Lladós, Daniel Lopresti, and Seiichi Uchida, editors, Document Analysis and Recognition – ICDAR 2021, Lecture Notes in Computer Science, pages 617–631, Cham, 2021. Springer International Publishing.

[90] Gavin Nishizawa, Jennifer Liu, Yancarlos Diaz, Abishai Dmello, Wei Zhong, and Richard Zanibbi. Mathseer: A math-aware search interface with intuitive formula editing, reuse, and lookup. In ECIR (2), volume 12036 of Lecture Notes in Computer Science, pages 470–475. Springer, 2020.

[91] Noel O'Boyle and Andrew Dalke. DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures. ChemRxiv, pages 1–9, 2018.

[92] Noel M. O'Boyle, Michael Banck, Craig A. James, Chris Morley, Tim Vandermeersch, and Geoffrey R. Hutchison. Open Babel: An open chemical toolbox. Journal of Cheminformatics, 3(1):33, 2011.

[93] Rochester Institute of Technology. Research computing services, 2025.

[94] Martijn Oldenhof, Adam Arany, Yves Moreau, and Jaak Simm. ChemGrapher: Optical Graph Recognition of Chemical Compounds by Deep Learning. Journal of Chemical Information and Modeling, 60(10):4506–4517, October 2020. Publisher: American Chemical Society.

[95] Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1):62–66, January 1979.

[96] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, pages 311–318, USA, July 2002. Association for Computational Linguistics.

[97] Shuai Peng, Liangcai Gao, Ke Yuan, and Zhi Tang. Image to LaTeX with Graph Neural Network for Mathematical Formula Recognition. In Josep Lladós, Daniel Lopresti, and Seiichi Uchida, editors, Document Analysis and Recognition – ICDAR 2021, Lecture Notes in Computer Science, pages 648–663, Cham, 2021. Springer International Publishing.

[98] Steven T. Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. Psychonomic bulletin & review, 21(5):1112–1130, October 2014.

[99] Florina Piroi, Mihai Lupu, Allan Hanbury, Alan P. Sexton, Walid Magdy, and Igor V. Filippov. CLEF-IP 2012: Retrieval experiments in the intellectual property domain. In Pamela Forner, Jussi Karlgren, and Christa Womser-Hacker, editors, CLEF 2012 Evaluation Labs and Workshop, CEUR Workshop Proceedings (CEUR-WS.org), 2012.

[100] Yujie Qian, Jiang Guo, Zhengkai Tu, Zhening Li, Connor W. Coley, and Regina Barzilay. MolScribe: Robust molecular structure recognition with image-to-graph generation. Journal of Chemical Information and Modeling, 63(7):1925–1934, 2023.

[101] Kohulan Rajan, Henning Otto Brinkhaus, Achim Zielesny, and Christoph Steinbeck. A review of optical chemical structure recognition tools. Journal of Cheminformatics, 12(1):60, 2020.

[102] Kohulan Rajan, Henning Otto Brinkhaus, Achim Zielesny, and Christoph Steinbeck. Advancements in hand-drawn chemical structure recognition through an enhanced DECIMER architecture. Journal of Cheminformatics, 16(1):78, July 2024.

[103] Kohulan Rajan, Achim Zielesny, and Christoph Steinbeck. DECIMER: towards deep learning for chemical image recognition. Journal of Cheminformatics, 12(1):1–9, 2020.

[104] Louis C. Ray and Russell A. Kirsch. Finding chemical records by digital computers. Science, 126(3278):814–819, 1957.

[105] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, Medical Image Computing and Computer-Assisted Intervention (MICCAI), pages 234–241, 2015.

[106] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks, June 2017.

[107] Noureddin M. Sadawi, Alan P. Sexton, and Volker Sorge. Performance of MolRec at TREC 2011 overview and analysis of results. In Ellen M. Voorhees and Lori P. Buckland, editors, Text REtrieval Conference (TREC), volume 500-296 of NIST Special Publication, 2011.

[108] Noureddin M. Sadawi, Alan P. Sexton, and Volker Sorge. Molrec at CLEF 2012 - overview and analysis of results. In Pamela Forner, Jussi Karlgren, and Christa Womser-Hacker, editors, CLEF 2012 Evaluation Labs and Workshop, volume 1178 of CEUR Workshop Proceedings (CEUR-WS.org), 2012.

[109] Chris Sasarak, Kevin Hart, Richard Pospesel, David Stalnaker, Lei Hu, Robert LiVolsi, Siyu Zhu, and Richard Zanibbi. min: A multimodal web interface for math search. In Proc. Human-Centered Information Retrieval (HCIR), Cambridge, MA, USA, 2012.

[110] Felix M. Schmitt-Koopmann, Elaine M. Huang, Hans-Peter Hutter, Thilo Stadelmann, and Alireza Darvishy. MathNet: A Data-Centric Approach for Printed Mathematical Expression Recognition. IEEE Access, 12:76963–76974, 2024.

[111] Klaus U. Schulz and Stoyan Mihov. Fast string correction with Levenshtein automata. International Journal on Document Analysis and Recognition (IJDAR), 5(1):67–85, 2002.

[112] Ayush Kumar Shah, Bryan Amador, Abhisek Dey, Ming Creekmore, Blake Ocampo, Scott Denmark, and Richard Zanibbi. Chemscraper: leveraging pdf graphics instructions for molecular diagram parsing. International Journal on Document Analysis and Recognition (IJDAR), 2024.

[113] Ayush Kumar Shah, Abhisek Dey, and Richard Zanibbi. A math formula extraction and evaluation framework for pdf documents. In International Conference on Document Analysis and Recognition (ICDAR), pages 19–34, 2021.

[114] Ayush Kumar Shah and Richard Zanibbi. Line-of-sight with Graph Attention Parser (LGAP) for math formulas. In International Conference on Document Analysis and Recognition (ICDAR), page 401–419, 2023.

[115] Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph Transformers: A Survey, July 2024.

[116] Volker Sorge, Akashdeep Bansal, Neha M Jadhav, Himanshu Garg, Ayushi Verma, and M Balakrishnan. Towards generating web-accessible STEM documents from PDF. In Proceedings of the 17th International Web for All Conference, W4A '20, pages 1–5, New York, NY, USA, April 2020. Association for Computing Machinery.

[117] Joshua Staker, Kyle Marshall, Robert Abel, and Carolyn M. McQuaw. Molecular structure extraction from documents using deep learning. Journal of Chemical Information and Modeling, 59(3):1017–1029, 2019. PMID: 30758950.

[118] M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. In Eighth International Conference on Document Analysis and Recognition (ICDAR'05), pages 675–679 Vol. 2, 2005.

[119] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INFTY: an integrated OCR system for mathematical documents. In Proceedings of the 2003 ACM symposium on Document engineering, DocEng '03, pages 95–104, New York, NY, USA, November 2003. Association for Computing Machinery.

[120] Masakazu Suzuki, Seiichi Uchida, and Akihiro Nomura. A ground-truthed mathematical character and symbol image database. In ICDAR, pages 675–679. IEEE Computer Society, 2005.

[121] Masakazu Suzuki and Katsuhito Yamaguchi. Recognition of E-Born PDF Including Mathematical Formulas. In Klaus Miesenberger, Christian Bühler, and Petr Penaz, editors, Computers Helping People with Special Needs, Lecture Notes in Computer Science, pages 35–42, Cham, 2016. Springer International Publishing.

[122] Jia-Man Tang, Hong-Yu Guo, Jin-Wen Wu, Fei Yin, and Lin-Lin Huang. Offline handwritten mathematical expression recognition with graph encoder and transformer decoder. Pattern Recognition, 148:110155, April 2024.

[123] Jia-Man Tang, Jin-Wen Wu, Fei Yin, and Lin-Lin Huang. Offline Handwritten Mathematical Expression Recognition via Graph Reasoning Network. In Christian Wallraven, Qingshan Liu, and Hajime Nagahara, editors, Pattern Recognition, Lecture Notes in Computer Science, pages 17–31, Cham, 2022. Springer International Publishing.

[124] Peng Tang, Siu Cheung Hui, and Chi-Wing Fu. Online chemical symbol recognition for handwritten chemical expression recognition. In 2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS), pages 535–540, 2013.

[125] Seiichi Toyota, Seiichi Uchida, and Masakazu Suzuki. Structural Analysis of Mathematical Formulae with Verification Based on Formula Description Grammar. In Horst Bunke and A. Lawrence Spitz, editors, Document Analysis Systems VII, Lecture Notes in Computer Science, pages 153–163, Berlin, Heidelberg, 2006. Springer.

[126] Thanh-Nghia Truong, Cuong Tuan Nguyen, Richard Zanibbi, Harold Mouchère, and Masaki Nakagawa. A survey on handwritten mathematical expression recognition: The rise of encoder-decoder and GNN models. Pattern Recognition, 153:110531, September 2024.

[127] Simon Vandenhende, Stamatios Georgoulis, and Luc Van Gool. MTI-Net: Multi-scale Task Interaction Networks for Multi-task Learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, Computer Vision – ECCV 2020, pages 527–543, Cham, 2020. Springer International Publishing.

[128] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS), pages 5998–6008, 2017.

[129] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In International Conference on Learning Representations, February 2018.

[130] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6):583–598, June 1991.

[131] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: A Line Segment Detector. Image Processing On Line, 2:35–55, March 2012.

[132] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling cross stage partial network. In Conference on Computer Vision and Pattern Recognition (CVPR), pages 13024–13033, 2021.

[133] Jiaming Wang, Jun Du, Jianshu Zhang, Bin Wang, and Bo Ren. Stroke constrained attention network for online handwritten mathematical expression recognition. Pattern Recognition, 119:108047, November 2021.

[134] Yan Wang, Ruochi Zhang, Shengde Zhang, Liming Guo, Qiong Zhou, Bowen Zhao, Xiaotong Mo, Qian Yang, Yajuan Huang, Kewei Li, Yusi Fan, Lan Huang, and Fengfeng Zhou. OCMR: A comprehensive framework for optical chemical molecular recognition. Comput. Biol. Med., 163(C), 2023.

[135] David Weininger. SMILES, a chemical language and information system: Introduction to methodology and encoding rules. Journal of Chemical Information and Computer Sciences, 28(1):31–36, 1988.

[136] Changjie Wu, Jun Du, Yunqing Li, Jianshu Zhang, Chen Yang, Bo Ren, and Yiqing Hu. TDv2: A Novel Tree-Structured Decoder for Offline Mathematical Expression Recognition. Proceedings of the AAAI Conference on Artificial Intelligence, 36(3):2694–2702, June 2022.

[137] Jin-Wen Wu, Fei Yin, Yan-Ming Zhang, Xu-Yao Zhang, and Cheng-Lin Liu. Graph-to-Graph: Towards Accurate and Interpretable Online Handwritten Mathematical Expression Recognition. Association for the Advancement of Artificial Intelligence, page 9, 2021.

[138] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5987–5995, Honolulu, HI, July 2017. IEEE.

[139] Yejing Xie and Harold Mouchère. Stroke-Level Graph Labeling with Edge-Weighted Graph Attention Network for Handwritten Mathematical Expression Recognition. In Elisa H. Barney Smith, Marcus Liwicki, and Liangrui Peng, editors, Document Analysis and Recognition - ICDAR 2024, pages 38–55, Cham, 2024. Springer Nature Switzerland.

[140] Yejing Xie, Harold Mouchère, Foteini Simistira Liwicki, Sumit Rakesh, Rajkumar Saini, Masaki Nakagawa, Cuong Tuan Nguyen, and Thanh-Nghia Truong. Icdar 2023 crohme: Competition on recognition of handwritten mathematical expressions. In Document Analysis and Recognition - ICDAR 2023: 17th International Conference, San José, CA, USA, August 21–26, 2023, Proceedings, Part II, page 553–565, Berlin, Heidelberg, 2023. Springer-Verlag.

[141] Yejing Xie, Richard Zanibbi, and Harold Mouchère. Local and Global Graph Modeling with Edge-weighted Graph Attention Network for Handwritten Mathematical Expression Recognition, October 2024.

[142] Dan Xu, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 675–684, 2018.

[143] Guikun Xu, Yongquan Jiang, PengChuan Lei, Yan Yang, and Jim Chen. GTMGC: Using Graph Transformer to Predict Molecule's Ground-State Conformation. In The Twelfth International Conference on Learning Representations, October 2023.

[144] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: neural image caption generation with visual attention. In Proceedings of the 32nd International Conference on

International Conference on Machine Learning - Volume 37, ICML'15, pages 2048–2057, Lille, France, July 2015. JMLR.org.

[145] Youjun Xu, Jinchuan Xiao, Chia-Han Chou, Jianhang Zhang, Jintao Zhu, Qiwan Hu, Hemin Li, Ningsheng Han, Bingyu Liu, Shuaipeng Zhang, Jinyu Han, Zhen Zhang, Shuhao Zhang, Weilin Zhang, Luhua Lai, and Jianfeng Pei. MolMiner: You Only Look Once for Chemical Structure Recognition. Journal of Chemical Information and Modeling, 62(22):5321–5328, November 2022.

[146] Zhanpeng Xu, Jianhua Li, Zhaopeng Yang, Shiliang Li, and Honglin Li. SwinOCSR: End-to-end optical chemical structure recognition using a Swin transformer. Journal of Cheminformatics, 14(1):41, 2022.

[147] Yongxin Yang and Timothy M. Hospedales. Trace norm regularised deep multi-task learning. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. OpenReview.net, 2017.

[148] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.

[149] Sanghyun Yoo, Ohyun Kwon, and Hoshik Lee. Image-to-graph transformers for chemical structure recognition. In International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3393–3397, 2022.

[150] Abdou Youssef. Part-of-math tagging and applications. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, Intelligent Computer Mathematics, pages 356–374, Cham, 2017. Springer International Publishing.

[151] D. Yu, X. Li, C. Zhang, T. Liu, J. Han, J. Liu, and E. Ding. Towards Accurate Scene Text Recognition With Semantic Reasoning Networks. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 12110–12119, June 2020. ISSN: 2575-7075.

[152] R. Zanibbi, D. Blostein, and J. R. Cordy. Recognizing mathematical expressions using tree transformation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(11):1455–1467, November 2002. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[153] Richard Zanibbi, Akiko Aizawa, Michael Kohlhase, Iadh Ounis, Goran Topic, and Kenny Davila. NTCIR-12 mathir task overview. In NTCIR. National Institute of Informatics (NII), 2016.

[154] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. International Journal on Document Analysis and Recognition (IJDAR), 15(4):331–357, December 2012.

[155] Richard Zanibbi, Harold Mouchère, and Christian Viard-Gaudin. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In Document Recognition and Retrieval XX, volume 8658, page 865817. International Society for Optics and Photonics, February 2013.

[156] Richard Zanibbi and Awelemdy Orakwue. Math Search for the Masses: Multimodal Search Interfaces and Appearance-Based Retrieval. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, Intelligent Computer Mathematics, Lecture Notes in Computer Science, pages 18–36, Cham, 2015. Springer International Publishing.

[157] Richard Zanibbi, Amit Pillay, Harold Mouchere, Christian Viard-Gaudin, and Dorothea Blostein. Stroke-based performance metrics for handwritten mathematical expressions. In International Conference on Document Analysis and Recognition (ICDAR), page 334–338, USA, 2011.

[158] Richard Zanibbi and Li Yu. Math spotting: Retrieving math in technical documents using handwritten query images. In ICDAR, pages 446–451. IEEE Computer Society, 2011.

[159] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. ResNeSt: Split-Attention Networks. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pages 2735–2745, June 2022.

[160] J. Zhang, J. Du, and L. Dai. Track, Attend, and Parse (TAP): An End-to-End Framework for Online Handwritten Mathematical Expression Recognition. IEEE Transactions on Multimedia, 21(1):221–233, January 2019. Conference Name: IEEE Transactions on Multimedia.

[161] Jianshu Zhang, Jun Du, Yongxin Yang, Yi-Zhe Song, and Lirong Dai. SRD: A Tree Structure Based Decoder for Online Handwritten Mathematical Expression Recognition. IEEE Transactions on Multimedia, 23:2471–2480, 2021.

[162] Jianshu Zhang, Jun Du, Shiliang Zhang, Dan Liu, Yulong Hu, Jinshui Hu, Si Wei, and Lirong Dai. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. Pattern Recognition, 71:196–206, November 2017.

[163] Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. Graph-to-tree learning for solving math word problems. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 3928–3937, Online, July 2020. Association for Computational Linguistics.

[164] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. Commun. ACM, 27(3):236–239, March 1984.

[165] X. Zhang, L. Gao, K. Yuan, R. Liu, Z. Jiang, and Z. Tang. A Symbol Dominance Based Formulae Recognition Approach for PDF Documents. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), volume 01, pages 1144–1149, November 2017. ISSN: 2379-2140.

[166] Yang Zhang, Guangshun Shi, and Jufeng Yang. Hmm-based online recognition of handwritten chemical symbols. In 2009 10th International Conference on Document Analysis and Recognition, pages 1255–1259, 2009.

[167] Zhenyu Zhang, Zhen Cui, Chunyan Xu, Zequn Jie, Xiang Li, and Jian Yang. Joint task-recursive learning for semantic segmentation and depth estimation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, Computer Vision – ECCV 2018, pages 238–255, Cham, 2018. Springer International Publishing.

[168] Zhenyu Zhang, Zhen Cui, Chunyan Xu, Yan Yan, Nicu Sebe, and Jian Yang. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4101–4110, 2019.

[169] Dmytro Zhelezniakov, Viktor Zaytsev, and Olga Radyvonenko. Online Handwritten Mathematical Expression Recognition and Applications: A Survey. IEEE Access, 9:38352–38373, 2021.

[170] Jianhua Zhu, Liangcai Gao, and Wenqi Zhao. ICAL: Implicit Character-Aided Learning for Enhanced Handwritten Mathematical Expression Recognition. In Elisa H. Barney Smith, Marcus Liwicki, and Liangrui Peng, editors, Document Analysis and Recognition - ICDAR 2024, pages 21–37, Cham, 2024. Springer Nature Switzerland.

Appendices

Appendix A

Feature Resolution and Pooling Configuration Search

This appendix presents a grid search evaluating the impact of input feature size and spatial pyramidal pooling (SPP) configurations on the parsing performance of mathematical formulas and chemical diagrams. These experiments were conducted on representative subsets of each domain to inform design choices for the visual encoding backbone.

Subset Datasets. To perform the grid search efficiently while ensuring diversity, we used curated subsets of both chemical and mathematical datasets.

For the **chemical domain**, all samples were drawn from the PubChem-5k dataset. The small training and validation set consists of 216 diagrams sampled from the Indigo-rendered PubChem corpus. These were split into an 80% training set (115 diagrams) and 20% validation set (27 diagrams). For evaluation, we used a held-out test set of 231 diagrams, also sampled from Indigo (150) and RDKit (81) renderers to reflect style variability. All diagrams used in these subsets were verified for correctness using our born-digital parser pipeline.

For the **mathematical domain**, we sampled from the INFTY-MCDB training set. Specifically, 10% of the full INFTY training set was used for training (1255 formulas), with an 80-20 split into 1004 training and 251 validation examples. A separate 15% held-out portion (1708 formulas) was reserved for development testing.

These subsets enabled fast, reproducible grid search while maintaining a representative distribution of layout structures and symbol classes. All subsequent performance evaluations in this appendix are based on these small-scale but diverse subsets.

We define two sets of **independent variables** for this search: *(i)* the resolution of input visual primitives, and *(ii)* the number and arrangement of pooling regions used in spatial pyramidal pooling.

Input Feature Sizes. We explore four fixed-size square resolutions for cropped primitive inputs: 28×28, 32×32, 48×48, and 64×64. The smallest size, 28×28, was inspired by classical digit recognition tasks such as MNIST [60], where symbols are small and relatively isolated. While efficient, this resolution risks omitting key contextual details for visually similar but structurally distinct symbols. The 32×32 resolution, previously used in our chemical parsing models, offers slightly better spatial coverage while maintaining low memory usage. Increasing the patch size further to 48×48 helps evaluate whether intermediate scaling improves performance in cluttered settings. Finally, 64×64 patches provide a large context window, better preserving structural boundaries and layout relationships, particularly useful for dense math formula regions. However, this comes at the cost of increased memory and inference time.

Spatial Pyramidal Pooling Regions. To assess the effect of spatial layout encoding, we vary the number of pooled regions in the spatial pyramid used after the CNN encoder as shown in Table A.1. The simplest configuration is a single-region global average pooling (GAP), which discards all spatial detail and produces a compact, low-dimensional representation. To capture coarse layout information, we introduce pooling over horizontal and vertical thirds ($3H + 3V$), resulting in a total of 7 pooled regions including the global one. A more fine-grained configuration adds pooling over horizontal and vertical fifths ($5H + 5V$), yielding 17 total regions. The densest setup includes horizontal and vertical sevenths ($7H + 7V$), forming a 31-region pooling scheme. As pooling granularity increases, so does the dimensionality of the pooled feature vector, which can improve task performance through better spatial discrimination but also leads to more learnable parameters and higher computational cost.

All other model components, including CNN backbone, loss function (Cross-Entropy), graph construction (LOS for math, 6NN for chemistry) were held constant across runs. These correspond to the standard settings described in Table 5.3 in Chapter 5.

Table A.1: Spatial pyramidal pooling configurations used in grid search.

Name	**Region Count**	**Configuration**
GAP (Global Average Pooling)	1	1
Coarse SPP	7	$1 + 3H + 3V$
Mid SPP	17	$1 + 3H + 3V + 5H + 5V$
Dense SPP	31	$1 + 3H + 3V + 5H + 5V + 7H + 7V$

Table A.2: Effect of input feature size on math and chemical subset performance.

Domain	**Input Size**	**Sym Det.**	**+Class**	**Rel Det.**	**+Class**	**Struct.**	**+Class**
Math	28x28	97.20	95.30	94.00	93.20	85.10	81.20
Math	32x32	97.60	95.80	94.50	93.70	87.30	83.40
Math	48x48	98.40	96.90	95.60	94.80	89.50	86.20
Math	64x64	**98.95**	**96.83**	**95.96**	**95.13**	**90.32**	**86.55**
Chem	28x28	96.10	92.30	90.50	88.40	72.50	68.00
Chem	32x32	**98.30**	**95.10**	**94.40**	**93.20**	**80.20**	**76.50**
Chem	48x48	98.20	94.90	94.10	92.80	80.00	76.00
Chem	64x64	98.10	94.70	93.90	92.50	79.50	75.40

A.1 Effect of Input Feature Size

Table A.2 summarizes the effect of input feature size on parsing performance for both math and chemical domains. In the case of mathematical formulas, performance improved consistently with increasing input size, with highest metrics at 64×64 for all tasks, including symbol classification, relationship classification, and expression-level accuracy. For chemical diagrams, performance also improved with larger sizes up to 32×32, beyond which metrics declined.

These results can be explained by differences in visual complexity and symbol diversity between the two domains. For mathematical formulas, the symbol set is large (207 classes), and symbols vary significantly in shape and spatial arrangement. Many expressions are deeply nested, and fine-grained visual features such as stroke curvature or subscript positioning are essential for accurate classification. A higher input resolution of 64×64 preserves these details, allowing the model to distinguish between visually similar but semantically distinct symbols, and to better resolve multi-level layout structures.

In contrast, chemical diagrams primarily consist of a smaller symbol set (67 classes), where most nodes represent line segments (bonds) or standard atom characters. These primitives are simpler in shape and generally more uniform in structure. The spatial context, rather than the internal visual

Table A.3: Effect of SPP pooling region configurations on parsing performance.

Domain	SPP Config	Sym Det.	+Class	Rel Det.	+Class	Struct.	+Class
Math	1	98.00	96.10	95.00	94.00	88.0	82.3
Math	7	98.21	96.42	95.13	94.32	88.7	83.4
Math	17	98.64	96.71	95.71	95.02	89.9	84.2
Math	31	**98.95**	**96.83**	**95.96**	**95.13**	**90.32**	**84.55**
Chem	1	96.50	93.20	91.80	90.00	76.1	71.9
Chem	7	97.22	94.00	92.50	91.30	78.5	74.1
Chem	17	98.00	94.80	93.80	92.70	79.7	75.3
Chem	31	**98.30**	**95.10**	**94.40**	**93.20**	**80.20**	**76.50**

details of each primitive, is often sufficient for classification. As a result, 32×32 provides a strong balance between resolution and efficiency. Increasing the input size beyond this adds unnecessary whitespace or background artifacts around small line-based primitives, which can dilute informative features, reduce effective contrast, and even introduce noise due to segmentation jitter, leading to slight drops in performance at 48×48 and 64×64.

A.2 Effect of Spatial Pyramidal Pooling Regions

The effect of pooling granularity was tested using a fixed input size (64×64 for math, 32×32 for chemistry). Table A.3 presents task-wise performance for different pooling configurations. We observed that progressively deeper SPP configurations led to consistent improvements in layout-sensitive tasks such as relationship classification and expression parsing. However, the optimal depth varied by domain.

The results in Table A.3 demonstrate how varying the number of spatial pyramidal pooling (SPP) regions impacts parsing performance across mathematical and chemical domains. SPP enhances the visual encoding of each primitive by capturing query and context features at multiple scales and orientations, which is critical in tasks that depend on the relative layout and local interactions of symbols or segments.

For **mathematical formulas**, larger SPP configurations, particularly the 31-region setup, yielded the best performance across all tasks. This is due to the highly structured and often nested nature of mathematical notation, where the position of symbols in 2D space (e.g., superscripts, fractions, radicals) directly determines their semantic role. Fine-grained spatial encoding provided by deeper pyramids helps disambiguate between similar primitives placed in different spatial configurations.

Additionally, the large symbol vocabulary (207 classes) increases the need for rich contextual cues to accurately interpret the role of each symbol, especially when dealing with subtle variations in arrangement or scale.

For **chemical diagrams**, the 17-region SPP configuration was found to be sufficient, with minimal gains observed when increasing to 31 regions. Chemical structures are predominantly composed of atoms and bonds with limited variability in spatial layout. Most relationships are local, and the diagram structure is relatively sparse compared to densely written math formulas. Thus, excessive pooling regions may introduce redundancy without offering meaningful gains, while also increasing the model's computational load. Furthermore, the over-segmentation of visual primitives (e.g., bond lines) in chemistry often makes the visual signal more fragmented; thus, simpler pooling helps avoid overfitting to small irrelevant variations.