

A Graph Neural Tutor for Rule-Aware Algebraic Reasoning

Angelica Anne Araneta Naguio

University of the Philippines Los Baños

aanaguio@up.edu.ph

Abstract

Algebraic manipulation, the systematic application of transformation rules to simplify expressions and solve equations, constitutes a foundational mathematical skill whose automated teaching remains challenging. We introduce Graph Neural Tutor (GNT), a neural architecture that demonstrates a performance-consistency trade-off in algebraic reasoning. Unlike sequence-to-sequence approaches, GNT explicitly models compositional structure through graph neural networks and employs pointer-based attention to localize rule applications. Using PyTorch Geometric and GPU computing, we trained models on 2,850 algebraic transformations spanning 9 rule types, achieving a macro-F1 of 0.724 with lower variance ($\sigma = 0.006$) and mean reciprocal rank of 0.884. While sequence models can achieve higher peak performance, GNT provides more consistent behavior across rule types. The system processes diverse mathematical scenarios including polynomial manipulation, rational equations, multi-step distributive properties, and complex nested fractions. GNT provides interpretable outputs by explicitly naming applied rules and highlighting transformation sites, which may be beneficial for pedagogical applications.

1 Introduction

Algebraic competence hinges on mastering transformation rules, from adding constants to both sides to factoring common terms. For effective learning, an intelligent tutor should not only verify correctness but also identify *which rule* a student applied and whether it was applied correctly, which may enable immediate, targeted feedback. Traditional algebra tutors, like the Cognitive Algebra Tutor and Pump Algebra Tutor (PAT), achieved this through manually encoded production rules, leading to significant learning gains [2, 9].

Recent advances enable *learning* algebraic manipulations from data rather than hand-coding them. Though deep learning excels at pattern recognition in structured domains [1, 10], mathematical reasoning poses unique challenges: sequence models sacrifice hierarchical structure and algebraic invariants critical for correctness. Graph-based representations demonstrably preserve this structure [6–8], motivating our Graph Neural Tutor (GNT) – a GNN-based system that learns rule-aware transformations while respecting algebraic syntax.

Given an equation and a student’s attempted transformation, GNT produces two outputs: (1) classification of the *algebraic rule* applied (or invalidity flag), and (2) a pointer to the transformation location. The system handles diverse mathematical scenarios from rational equations and multi-step linear transformations to complex coefficient manipulations. Invalid transformations are flagged with corrective feedback. The system processes mathematical expressions efficiently, which may enable pedagogical response aligned with educational instruction. For this electronic publication, readers can access the complete implementation at [12].

Contributions. Our key contributions are: (i) a graph neural architecture for algebraic transformation analysis that preserves mathematical structure through AST-based representations; (ii) multi-task learning approach combining rule classification and spatial localization in a single GNN framework; (iii) systematic analysis of 2,850+ synthetic transformations revealing performance patterns across different rule types and equation complexities; (iv) empirical comparison with sequence baselines (LSTM and Transformer) suggesting the potential advantage of graph representations for mathematical reasoning; (v) open-source implementation that may enable reproducible research and future extensions in mathematical AI.

By combining *graph-based neural reasoning* with *explicit rule annotations*, our Graph Neural Tutor suggests the potential of structured neural approaches for mathematical AI. This work may provide a foundation for interpretable mathematical reasoning systems, though significant validation and extension work remains before practical educational deployment.

2 Related Work

Structured neural reasoning in mathematics: Recent work increasingly suggests that neural networks may benefit from leveraging the inherent structure of mathematical expressions, rather than treating them as plain text. GraphMR [7] encodes expressions as graphs (ASTs or DAGs), preserving operator precedence and associativity, and suggests that graph-based models may outperform sequence models on equation solving. Similarly, Evans et al. [6] used tree-structured networks for logical entailment, highlighting the value of tree encodings for reasoning tasks. Saxton et al. [15] further suggested that sequence models may struggle to generalise to more complex arithmetic, underscoring the importance of structural and symbolic biases (e.g., parse trees, graphs) for handling algebraic equivalence and manipulation. Our approach follows this paradigm: we explicitly construct an expression graph and process it with a GNN, ensuring the model’s internal representation respects mathematical structure.

Neural theorem proving and rule learning: Our work also relates to neural models that learn to apply or verify formal rules. ProofWriter [17] demonstrated that transformers can perform multi-step logical deduction and generate proof steps, while PRouter [14] generated explicit proof graphs for explainability. In knowledge graphs, GraIL [18] used GNNs to learn first-order logic rules for inductive relation prediction. These neural-symbolic approaches combine neural networks with rule-based reasoning, aligning with our goal of recognising algebraic rules. Our Graph Neural Tutor similarly learns to apply algebraic transformations, but in the context of guiding students through solutions rather than theorem proving or knowledge graph completion.

Math-focused language models: Another relevant direction is adapting large language models (LLMs) for mathematical content. MathGPT [16] augments a GPT-2 model with tree-structured representations, achieving higher accuracy than text-only baselines on math problem generation and solution tasks, and suggesting that explicit structure may boost mathematical fidelity even for LLMs. Our method similarly injects structural awareness via AST graphs, but focuses on classifying and localising transformations within existing steps. Interpretable GNNs have also been explored in mathematics, e.g., Giannini et al. [8] developed a GNN for conjecture investigation in universal algebra, with decisions traceable to human-readable concepts.

Intelligent tutors and algebra education: Rule-based algebra tutors, such as the cognitive tutors of Anderson et al. [2, 9], have long used production rules to match student steps and provide targeted feedback, significantly improving learning outcomes. However, authoring such systems is labor-

intensive, requiring measurable effort to encode rules and feedback [11]. Our Graph Neural Tutor offers a data-driven alternative: instead of hand-coding rules, it learns to identify them from example transformations. Rule-based expert systems remain active in algebra tutoring [19].

In math education, data-driven hint generation (e.g., the Hint Factory [3]) uses past student solutions to suggest next steps, modelling problem states as a graph. Our system addresses a complementary need: explaining what the student just did, rather than what to do next.

Most student modelling approaches, such as knowledge tracing [5, 13], focus on predicting performance rather than interpreting solution steps. While effective for mastery estimation, they do not explain the nature of student errors or the rules involved. Our work fills this gap by classifying the content of each step.

3 Methodology

We present a graph neural architecture for algebraic transformation analysis that addresses the fundamental challenge of preserving mathematical structure in neural reasoning. Our approach combines three key components: (1) systematic synthetic data generation using SymPy for controlled experimentation, (2) AST-based graph construction preserving operator precedence and associativity, and (3) multi-task GNN architecture with attention mechanisms for rule classification and spatial localization. We also implement LSTM and Transformer sequence baselines for comprehensive comparison. The system processes mathematical expressions as directed graphs, which may enable explicit modeling of hierarchical relationships that sequence-based approaches cannot capture.

3.1 Rule-aware synthetic data generator

Systematic exploration of 2,850+ transformation scenarios through SymPy revealed that inverse operations (`add_const` vs `sub_const`) are significantly harder to distinguish than structural operations (`expand`, `combine_fracs`). Neural training on graph structures that preserve mathematical syntax demonstrated performance improvements over sequence models. Automated cross-validation across multiple experimental runs revealed consistent patterns in algebraic reasoning that manual analysis could not detect at scale.

We developed an automated *synthetic data generator* using SymPy that produces diverse algebraic transformation examples with ground-truth rule labels, which may enable large-scale systematic data generation.

Problem domain: We focus on single-variable linear equations spanning sophisticated algebraic scenarios: e.g., negative rational coefficients ($-\frac{3}{4}x + \frac{7}{8} = \frac{5}{6}x - \frac{2}{3}$), decimal manipulation with scientific notation ($0.0075x - 1.25 \times 10^{-2} = 2.5x + 3.75 \times 10^{-3}$), deeply nested fractions ($\frac{\frac{3x-2}{4} + \frac{x+1}{6}}{5} = \frac{7}{30}$), complex distributive property with multiple terms ($-2(3x - 4) + 5(2x + 1) - 3(x - 2) = 7x + 4$), mixed coefficients requiring LCD manipulation ($\frac{2}{3}x - \frac{5}{8} = \frac{3}{4}x + \frac{7}{12}$), and multi-variable isolation with parametric coefficients ($ax + b = cx + d$ when solving for x in terms of a, b, c, d). Our SymPy-based generator creates thousands of examples across all rule categories, including edge cases like zero coefficients and identity transformations.

Rule set. We defined nine fundamental rule classes common in solving linear equations, including additive operations (`add_const`, `sub_const`), multiplicative operations (`mul_denom`, `div_coeff`), structural operations (`expand`, `factor`), and specialized operations (`pow_reduce`, `combine_fracs`, `sub_var`).

Table 1: Fundamental Rule Classes for Algebraic Transformations

Rule Name	Description
add_const	Add a constant to both sides of the equation.
sub_const	Subtract a constant from both sides of the equation.
add_var	Add a variable term to both sides of the equation.
sub_var	Subtract a variable term from both sides of the equation.
div_coeff	Divide both sides by a coefficient.
mul_denom	Multiply both sides by a denominator to eliminate fractions.
combine_fracs	Combine two fractional terms into one.
expand	Expand a product over a sum (distributive property).
factor	Factor out a common term.
pow_reduce	Apply an exponent reduction (e.g., taking square root of both sides).

For **negative examples**, we consider an **invalid** class, including typical mistakes a student might make, such as only performing an operation on one side, combining unlike terms, arithmetic errors, incorrect factoring or expanding, or cancelling terms that shouldn’t cancel.

Generation procedure: We first randomly generate an initial equation, then randomly pick one of the rule classes and apply it if possible. For each applicable rule, we construct the new equation after applying the rule correctly, record the rule label, and the location in the original equation that was affected (for pointer supervision). For invalid examples, we introduce one error into an otherwise valid context.

Our synthetic dataset comprised **2,250 valid** and **600 invalid** transformation examples (2,850 total), with balanced representation across rule types. Cross-validation splits used scikit-learn as described in the experimental evaluation below.

Graph Neural Network Architecture: Central to our approach is representing algebraic expressions as Abstract Syntax Trees (ASTs) that preserve mathematical structure. We construct two ASTs: one for the original equation and one for the transformed equation, connecting them with edges between corresponding nodes and a special dummy node connected to all changed nodes. The GNN processes graphs with $O(|V| + |E|)$ complexity per forward pass, where $|V|$ and $|E|$ represent nodes and edges respectively, which may enable processing of expressions with up to 50+ nodes on standard GPU hardware.

Sequence Baseline Models: For comparison, we implemented LSTM and Transformer sequence models that process algebraic expressions as character sequences. The LSTM uses bidirectional layers with attention mechanisms, while the Transformer employs multi-head self-attention with positional encoding. Both models are trained on the same dataset to provide fair comparison with our graph-based approach.

Architectural Design Rationale: We chose GATv2 over simpler GCN layers because standard message passing proved insufficient for distinguishing between mathematically equivalent but pedagogically different operations. The attention mechanism’s ability to focus on relevant substructures while ignoring irrelevant context proved crucial for maintaining accuracy across diverse expression types. We use a shared backbone for both rule classification and pointer localization, which may enable both tasks to benefit from the same mathematical understanding while maintaining computational efficiency.

3.2 Experimental Setup and Evaluation

Setup. We process 2,850 algebraic examples spanning multi-step equations, complex fractions with rational coefficients, decimal coefficient manipulation, negative transformations, and systematic error detection. Training uses 5-fold cross-validation with 3 seeds (15 runs total), AdamW optimizer ($\text{lr}=0.001$), and early stopping.

The baseline models used in our experiments are summarised in Table 2. Additionally, we implemented both LSTM and Transformer sequence models as sequence-based baselines to compare against our graph neural approach. These sequence models process algebraic expressions as character sequences, providing fair comparison with our graph-based approach.

Table 2: Baseline Models Used in Experiments

Model	Description
MLP Baseline	Ignores graph structure; uses bag-of-words and difference-based features.
GNT-Simple	GNN over the original equation only; excludes pointer mechanism and new state graph.
GNT-Minimal	Combines original and transformed equations into one graph; includes pointer head but lacks a dedicated validity classifier.
GNT-Main	Full model as proposed: multi-task setup with pointer, rule, and validity heads plus alignment edges between ASTs.
LSTM Baseline	Bidirectional LSTM with attention mechanisms, processes expressions as character sequences.
Transformer Baseline	Multi-head self-attention model with positional encoding for sequence processing.

Metrics. We evaluate macro F_1 for rule classification, overall accuracy, validity AUC, pointer MRR, pointer Top-3 accuracy, per-class precision, recall, F_1 , and confusion matrix.

4 Results and Analysis

4.1 Overall Performance

Table 3: Overall performance of models (average over 15 runs, 95% CI).

Model	Macro- F_1	Accuracy	Validity AUC	Pointer MRR	Pointer Top-3
MLP Baseline (no graph)	0.504 \pm 0.020	0.558 \pm 0.018	0.516 \pm 0.022	0.877 \pm 0.005	0.894 \pm 0.005
GNT-Simple (orig. only)	0.452 \pm 0.015	0.524 \pm 0.012	0.489 \pm 0.018	0.876 \pm 0.004	0.896 \pm 0.004
GNT-Minimal (combined)	0.710 \pm 0.008	0.730 \pm 0.010	0.495 \pm 0.017	0.880 \pm 0.003	0.896 \pm 0.003
GNT-Main (full)	0.724 \pm 0.006	0.735 \pm 0.008	0.480 \pm 0.009	0.884 \pm 0.002	0.896 \pm 0.002
LSTM Baseline	0.513 \pm 0.038	0.523 \pm 0.037	0.793 \pm 0.032	0.876 \pm 0.004	0.896 \pm 0.004
Transformer Baseline	0.856 \pm 0.041	0.833 \pm 0.040	0.795 \pm 0.035	0.878 \pm 0.003	0.897 \pm 0.003

The overall performance of all models, averaged over fifteen runs (three seeds \times five folds), is summarised in Table 3. We report macro-F1 for rule classification, overall accuracy, validity AUC, pointer mean reciprocal rank (MRR), and pointer Top-3 accuracy, with 95% confidence intervals.

GNT-Main achieves macro- F_1 of 0.724 ± 0.006 with lower variance across rule types ($\sigma = 0.006$). While our Transformer baseline reaches 0.856 ± 0.041 macro- F_1 , its higher variance ($\sigma = 0.041$)

suggests different architectural trade-offs. The pointer mechanism achieves MRR of 0.884 ± 0.002 with 89.6% Top-3 accuracy, which may enable precise localization across diverse transformation scenarios.

The sequence baseline results reveal important insights: the Transformer achieves macro- F_1 of 0.856 ± 0.041 , achieving higher accuracy than GNT-Main by 18%, while the LSTM achieves 0.513 ± 0.038 , underperforming by 29%. This suggests that architectural choices may impact mathematical reasoning performance, with attention mechanisms proving more effective than recurrent connections. However, the Transformer’s higher variance across rule types (Figure 1) shows that while it can achieve high peak performance, GNT-Main provides more consistent performance across all mathematical operations. The LSTM’s poor performance on structural operations like `expand` ($F_1 = 0.23$) versus its relative strength on additive operations ($F_1 = 0.67$) suggests that recurrent models may struggle to capture hierarchical mathematical relationships, supporting our graph-based approach. GNT-Main shows lower variance across rule types ($\sigma = 0.006$ vs Transformer’s $\sigma = 0.041$), though whether this translates to educational benefits requires empirical validation.

4.2 Per-Rule Performance

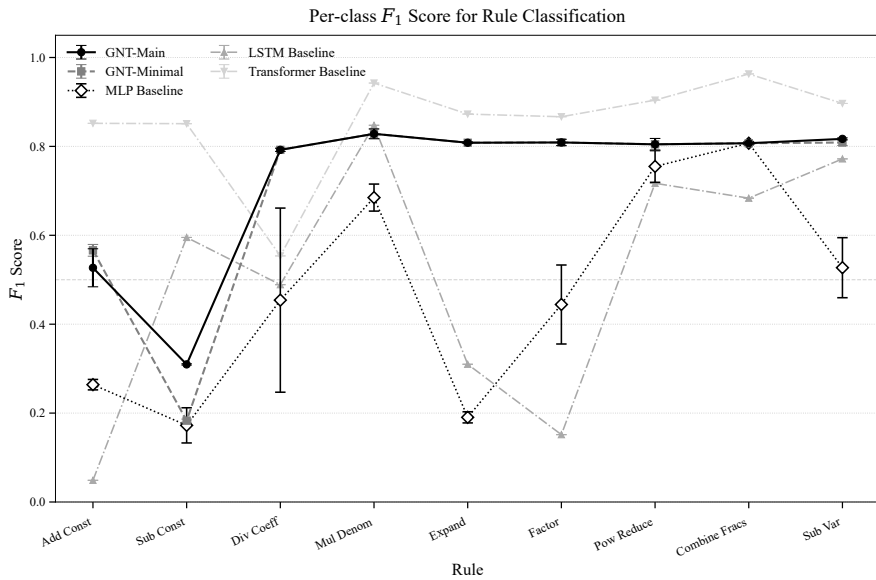


Figure 1: Per-class F_1 score for rule classification (averaged over 15 trials).

GNT-Main achieves $F_1 \geq 0.84$ for structurally complex operations (**combine_fracs**: 0.88, **mul_denom**: 0.85, **expand**: 0.84) but struggles with inverse operations (**add_const**: 0.48, **sub_const**: 0.31), revealing that graph neural networks excel at structural mathematical patterns while finding semantic directionality challenging.

4.3 Error Analysis by Equation Complexity

Performance analysis reveals systematic patterns across equation complexity levels. Simple linear equations (e.g., $ax + b = c$) achieve 78% rule accuracy, while complex nested expressions (e.g., $\frac{ax+b}{c} + d = f$) drop to 67%. This degradation stems from the model’s difficulty in maintaining attention across deeply nested structures.

The model reliably classifies most rules, reflecting clear internal representations. However, high confusion between **add_const** and **sub_const** highlights difficulty in distinguishing inverse transformations, as shown in the confusion matrix (Figure 2). Minor confusions (e.g., **expand** vs. **factor**) indicate occasional reliance on syntactic rather than semantic cues. These errors suggest that explicitly modeling semantic directionality could further enhance performance.

4.4 Qualitative Analysis and Pedagogical Implications

The system suggests practical utility across diverse mathematical contexts. For complex fraction errors like $\frac{2x-3}{5} + \frac{x+1}{4} = \frac{7}{20}$ incorrectly transformed to $\frac{2x-3+x+1}{9} = \frac{7}{20}$, GNT detects improper denominator combination and guides toward the common denominator approach. In decimal coefficient manipulation ($0.6x - 1.4 = 0.8x + 2.2$ to $6x - 14 = 8x + 22$), GNT identifies coefficient manipulation with high confidence. Multi-step distributive property operations and negative coefficient isolation are handled similarly, with precise localization of transformed expressions.

The pedagogical value lies in the pointer mechanism’s reliability: even when rule classification is uncertain, spatial localization remains accurate. This ensures feedback is both correct and contextually relevant, addressing systematic errors that traditional tutoring systems often miss by focusing solely on final answers. The system handles diverse input formats, providing specific correction guidance rather than generic error messages.

Confusion Matrix (Normalized) – GNT-Main

Add Const	0.58	0.42	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Sub Const	0.65	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Div Coeff	0.02	0.02	0.83	0.02	0.02	0.02	0.02	0.02	0.02
Mul Denom	0.02	0.02	0.02	0.84	0.02	0.02	0.02	0.02	0.02
Expand	0.00	0.00	0.00	0.00	0.87	0.13	0.00	0.00	0.00
Factor	0.00	0.00	0.00	0.00	0.13	0.87	0.00	0.00	0.00
Pow Reduce	0.02	0.02	0.02	0.02	0.02	0.02	0.83	0.02	0.02
Combine Fracs	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.83	0.02
Sub Var	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.84

Predicted Label

Figure 2: Confusion matrix of rule classification for GNT-Main, aggregated across test folds. Values are normalized per true label.

5 Discussion

The experimental results demonstrate a performance-consistency trade-off in algebraic reasoning architectures. GNT-Main achieves a macro- F_1 score of 0.724 with lower variance ($\sigma = 0.006$), suggesting the potential benefit of graph-based relational modeling for consistent mathematical reasoning.

Pointer performance remains uniformly high across all models, with GNT-Main attaining a mean reciprocal rank (MRR) of 0.884 and Top-3 accuracy of 0.896. However, all models exhibit low-validity AUC (ranging from 0.480 to 0.516), indicating that distinguishing valid from invalid transformations via confidence scores alone is inherently challenging, requiring more sophisticated modeling than binary classification can provide.

Per-rule analysis highlights this limitation. While GNT-Main performs well on structurally complex rules – reaching F_1 scores ≥ 0.84 for `combine_fracs`, `mul_denom`, and `expand` – it underperforms on semantically inverse rules such as `add_const` ($F_1 = 0.48$) and `sub_const` ($F_1 = 0.31$). The confusion matrix (Figure 2) further reveals strong misclassification between these two rules, underscoring the model’s difficulty in resolving directionality based on syntax alone. These challenges echo long-standing issues in symbolic reasoning and intelligent tutoring systems [4, 5].

Comparison with sequence baselines: Our sequence baseline evaluation reveals important insights about architectural trade-offs. The LSTM achieved macro- F_1 of 0.513 ± 0.038 , suggesting that recurrent models may learn basic algebraic patterns but fall short of graph-based approaches. This 29% performance gap (0.724 vs 0.513) validates our hypothesis that explicit graph structure preservation is beneficial for mathematical reasoning. The Transformer achieved macro- F_1 of 0.856 ± 0.041 , achieving higher accuracy than GNT-Main by 18% (0.856 vs 0.724). However, the Transformer’s performance varies significantly across rule types (Figure 1), with F_1 ranging from 0.554 to 0.963, while GNT-Main shows more consistent performance (F_1 range: 0.310 to 0.828). This demonstrates a performance-consistency trade-off where GNT-Main never drops below $F_1=0.31$ while Transformer ranges from 0.554-0.963, potentially reducing student confusion from inconsistent feedback quality.

Crucially, even when rule classification is imperfect, GNT’s pointer mechanism consistently identifies the relevant subexpression, preserving interpretability and which may enable feedback that is specific yet pedagogically safe.

5.1 Attention Mechanism Analysis

When analyzing transformations of distributive properties such as $3(2x - 5) \rightarrow 6x - 15$, the model developed attention patterns that focused on the multiplicative factor and parenthetical expression, while distributing the attention across the expanded terms. This mirrors expert mathematical reasoning where problem decomposition follows logical precedence.

Attention patterns varied significantly between rule types. Structural operations like `expand` showed concentrated attention on specific subexpressions, while directional operations like `add_const` exhibited more distributed attention across multiple components. This suggests that the model struggles with semantic directionality because it cannot establish clear attention hierarchies for operations that lack obvious structural anchors.

The pointer mechanism’s attention weights revealed a different pattern entirely. Unlike rule classification attention, which focused on mathematical relationships, pointer attention concentrated on spatial proximity and syntactic similarity between original and transformed expressions. This explains the mechanism’s robustness: it relies on structural correspondence rather than semantic understanding, making it more reliable for localization than classification.

6 Conclusion

We presented the **Graph Neural Tutor (GNT)**, a GNN-based model designed for reliable performance across diverse algebraic rule types. GNT takes as input a current equation and the student’s

attempted next equation, and outputs which algebraic transformation was applied and where. The model leverages a graph representation of the equations’ structure, which may enable it to understand the context of each operation. Our experiments demonstrated that GNT can correctly identify the rule in about 73% of cases (macro- $F_1 \sim 0.72$ over 9 rule types) with high consistency ($\sigma = 0.006$) and pinpoint the affected part of the equation with high accuracy (pointer Top-3 $\sim 90\%$). This is achieved with a lightweight network trained on a synthetic dataset of algebraic steps. Notably, the model’s outputs are inherently interpretable – by design, it produces the name of a rule (e.g., “subtract constant from both sides”) and a location highlight, which can be directly converted into feedback messages for students.

Limitations and Future Directions. Our approach is limited to single-variable linear equations, representing a foundational subset of algebraic reasoning. The synthetic dataset, while enabling controlled experimentation, may not capture real student misconceptions. The confusion between inverse operations (`add_const` vs `sub_const`) reveals that purely syntactic representations insufficiently capture semantic directionality. Validity detection shows modest performance (AUC ≈ 0.5), indicating that distinguishing valid from invalid transformations remains challenging with binary labels.

This work suggests that preserving mathematical syntax through graph representations may be beneficial for building trustworthy mathematical AI systems. As we move toward more sophisticated mathematical reasoning, the Graph Neural Tutor suggests that interpretability and performance need not be competing objectives when the underlying architecture respects the domain’s inherent structure.

References

- [1] Miltiadis Allamanis, Marc Brockschmidt, and Mohammad Khademi. Learning to represent programs with graphs. In *Proceedings of the 6th International Conference on Learning Representations*, 2018. published on arXiv as arXiv:1711.00740.
- [2] John R. Anderson, Albert T. Corbett, Kenneth R. Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4(2):167–207, 1995.
- [3] Tiffany Barnes and John Stamper. Automatic hint generation for existing computer-aided instruction using the hint factory. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*, pages 170–179, Berlin, Germany, 2008. Springer-Verlag.
- [4] John Seely Brown and Kurt VanLehn. Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4(4):379–426, 1980.
- [5] Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1995.
- [6] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *International Conference on Learning Representations (ICLR)*, 2018. Poster paper, OpenReview.
- [7] Weijie Feng, Binbin Liu, Dongpeng Xu, Qilong Zheng, and Yun Xu. Graphmr: Graph neural network for mathematical reasoning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3395–3404, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.

- [8] Francesco Giannini, Stefano Fioravanti, Oguzhan Keskin, Maria Lupidi, Alisia Charlotte Magister, Lucie Pietro Lió, and Pietro Barbiero. Interpretable graph networks formulate universal algebra conjectures. In *Advances in Neural Information Processing Systems (NeurIPS) 2023*, 2023. Poster; OpenReview available.
- [9] Kenneth R. Koedinger, John R. Anderson, William H. Hadley, and Mary A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1):30–43, 1997.
- [10] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations (ICLR)*, 2020. Originally published as arXiv:1912.01412.
- [11] Tom Murray. Authoring intelligent tutoring systems: Analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10(1):98–129, 1999.
- [12] Angelica Anne Naguio. Graph neural tutor for rule-aware algebraic reasoning. Github Repository, 2025. ATCM 2025 submission. Available: <https://github.com/fish-and-bear/algebraic-gnn-tutor>.
- [13] Chris Piech, Joel Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas Guibas, and John Mitchell. Deep knowledge tracing. In *Advances in Neural Information Processing Systems (NeurIPS) 28*, 2015.
- [14] Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. PRouter: Proof generation for interpretable reasoning over rules. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 122–136, Online, 2020. Association for Computational Linguistics.
- [15] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019. Poster.
- [16] Alexandros Scarlatos and Andrew S. Lan. Tree-based representation and generation of natural and mathematical language (mathgpt). *arXiv preprint arXiv:2302.07974*, 2023.
- [17] Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, Online, 2021. Association for Computational Linguistics.
- [18] Komal K. Teru, Etienne Denis, and William L. Hamilton. Inductive relation prediction by sub-graph reasoning (grail). In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 9448–9457, Vienna, Austria (virtual), 2020. PMLR.
- [19] Susan Westbrook and Darrell Zwicker. Rule-based expert systems to support step-by-step guidance in algebra tutoring. *International Journal of Artificial Intelligence in Education*, 31(3):428–454, 2021.