Literature Review — Fine-Tuning T5 for Symbolic Math Translation

1. Task Definition and Scope

This work addresses the problem of symbolic formula synthesis from natural language, formulated as a supervised sequence-to-sequence learning task. Given a textual math prompt drawn from informal, compositional language, the goal is to generate a well-formed expression in a structured symbolic representation. The task can be defined as learning a mapping:

$$f : L_{\text{nat}} \to L_{\text{sym}},$$

where L_{nat} consists of natural language math questions (e.g., *'What is the product of 3 and 7?'*), and L_{sym} denotes a serialized prefix notation over a finite set of arithmetic functions, such as *multiply(3, 7)*.

The target space comprises function-call expressions with fixed operator arities and shallow compositional structure. Outputs are linearized as token sequences, without any enforced grammar or formal syntax constraints during decoding. The model is required to learn both operator selection and argument extraction implicitly from data, with no symbolic parsing or type checking built into the generation process.

This task sits at the intersection of semantic parsing and lightweight symbolic program synthesis, conceptually adjacent to work on natural language to code generation (e.g., Yin & Neubig, 2017; Polu et al., 2020), though with a more narrowly defined operator vocabulary and no intermediate execution layer. In contrast to numerical reasoning benchmarks such as GSM8K or Minerva, which require multi-hop logical deduction, the focus here is on direct, one-shot transduction of linguistic problems into symbolic form.

1.5 Task Motivation and Model Rationale

Translating natural language math prompts into symbolic form is a foundational task in neural-symbolic learning. It underpins capabilities in automated tutoring, symbolic solvers, data-to-code generation, and math-aware question answering. Compared to open-ended numerical reasoning, symbolic translation focuses on one-shot precision — producing expressions that can be parsed, executed, or manipulated downstream.

The appeal of this task lies in its dual structure: natural language as a high-variance, compositional input space; and symbolic language as a deterministic, formally structured output. Learning a mapping between the two tests a model's capacity for semantic parsing, operator alignment, and argument extraction under surface-level linguistic variability.

We adopt a pre-trained T5 encoder-decoder architecture for this task due to its general-purpose sequence-to-sequence formulation and strong empirical performance on both natural language understanding and structured prediction tasks. T5 treats every input and output as a text string, making it a natural fit for prefix-style symbolic languages when serialized linearly (where token order directly encodes structure and operator arity). Unlike encoder-only or decoder-only architectures, the encoder-decoder structure allows both bidirectional contextualization of input prompts while enabling controlled autoregressive generation of structured outputs.

This choice is also pragmatic: T5 models are widely available, computationally tractable for fine-tuning, and easily integrated with Hugging Face's transformers ecosystem. While the architecture imposes no hard structural priors on the output space, it enables fast convergence under standard token-level supervision, making it a clean and extensible baseline for symbolic translation tasks.

2. Model Architecture and Training

The model architecture follows a standard encoder-decoder transformer paradigm, instantiated as a pretrained T5 model (Raffel et al., 2020). Inputs are tokenized natural language math prompts, and outputs are linearized symbolic expressions in a custom prefix notation. Each training pair is cast as a sequence-to-sequence mapping, with both source and target represented as flat token streams.

During training, the model is fine-tuned using token-level cross-entropy loss, optimized with AdamW under a linear learning rate warmup schedule. Implementation is handled via PyTorch Lightning, enabling clean modularity, GPU acceleration, and automated checkpointing. All outputs are decoded auto-regressively using greedy decoding without beam search, temperature sampling, or syntactic constraints. As a result, the decoder emits raw token sequences with no structural guarantees — placing full reliance on the model's learned token transitions to recover valid symbolic structure.

This setup avoids any symbolic grammar enforcement, operator type-checking, or runtime validation. While this increases architectural flexibility and simplifies deployment, it also makes the model sensitive to distributional drift and vulnerable to malformed expressions — as later touched on.

Notably, this system is trained via direct fine-tuning on aligned example pairs and does not leverage instruction-tuning or prompt engineering. This distinguishes it from more recent work in symbolic reasoning that relies on in-context learning or chain-of-thought prompting strategies.

3. Data Preprocessing and Normalization

To minimize variability in symbolic outputs and improve comparability, a set of normalization functions is applied during preprocessing:

- **Whitespace regularization**: to enforce consistent token spacing
- **Constant stripping**: To remove auxiliary or irrelevant numeric values
- **Operator-parameter fusion**: To standardize function-call representations

These operations are implemented via utility functions (`insert_spaces()`, `remove_const()`, `fuse_operator_params()`) and are applied to both target labels and model predictions to align surface forms. Though lightweight, these transformations serve to reduce formatting variance without injecting symbolic priors or grammar awareness.

The dataset is further filtered to exclude malformed or excessively long symbolic outputs (token count > 100), as well as missing entries (dropped). While these steps improve string-level comparability, they may mask finer-grained structural errors such as arity mismatch or bracket misalignment.

The resulting dataset consists of synthetic, paired (NL, symbolic) examples. While sourced from a Kaggle dataset resembling SAT-style math questions, it bears structural resemblance to benchmark datasets like the DeepMind Mathematics Dataset and MathQA, where natural prompts are mapped to symbolic targets in constrained operator vocabularies.

4. Evaluation Methodology: Dual-Metric Approach

To evaluate model performance, we implement a dual-metric evaluation strategy capturing both surface-level accuracy and functional correctness (while token-level structure matters for syntactic accuracy, mathematical equivalence defines functional success):

a) Levenshtein Similarity (Normalized Edit Distance):

A normalized edit-distance score is computed between the predicted and reference formulas using difflib.SequenceMatcher. Predictions exceeding a fixed threshold (e.g., ≥ 0.91) are flagged as high-confidence structural matches. This threshold was selected heuristically to tolerate minor token drift (e.g., spacing, ordering) while penalizing deeper structural deviations.

b) Symbolic Equivalence via SymPy:

Predicted and reference expressions are parsed using `sympy.sympify()` and evaluated for numeric equality. Predictions are considered correct if they yield the same result as the ground truth when executed. This check captures semantic equivalence even when surface structure varies — for example, when `add(3, 4)` and `add(4, 3)` are both valid under commutativity.

Importantly, malformed or non-parseable expressions are treated as incorrect. No symbolic recovery or exception handling is implemented, meaning outputs that fail to compile in SymPy are scored as failures. This strict policy prioritizes syntactic precision over leniency and penalizes small errors that could otherwise obscure downstream evaluation.

5. Behavioral Characteristics of Flat Symbolic Transformers

Recent work has shown that encoder-decoder transformers, when applied to symbolic generation tasks under cross-entropy supervision, tend to converge rapidly to high-confidence output modes (Lample & Charton, 2019; Shin et al., 2021). This behavior is particularly pronounced in settings where the output space is shallow, low-entropy, and structurally uniform — such as linearized prefix languages over arithmetic operators (especially true when using greedy decoding without grammar supervision, as the model quickly overfits to dominant surface patterns).

Empirical trends observed in our system mirror this pattern. When trained on natural language math prompts with greedy decoding, the model exhibits early stabilization of outputs and minimal variance across checkpoints. This determinism is a feature of the decoding strategy but also reflects the bounded complexity of the symbolic domain.

However, reliance on flat token prediction without syntactic constraints introduces predictable vulnerabilities — operand ordering errors, malformed expression boundaries, and arity mismatches. These failure modes have been widely noted in neural program synthesis and semantic parsing without grammar supervision (Yin & Neubig, 2017; Dong & Lapata, 2016), underscoring the tradeoff between architectural flexibility and symbolic precision.

In this respect, flat symbolic generation models occupy a middle ground between grammar-constrained decoders and execution-guided symbolic systems. Their appeal lies in architectural simplicity and generality, but their limits become clear in tasks requiring structural invariance or compositional generalization.

6. Relation to Prior Work

Our work is situated within the expanding field of neural symbolic translation, where pretrained language models are adapted to generate formal representations from natural language prompts. It is most closely aligned with work on symbolic program synthesis and lightweight semantic parsing, and draws direct precedent from the application of transformer-based architectures to symbolic reasoning tasks.

Lample & Charton (2019) demonstrated that sequence-to-sequence models can learn algebraic manipulation and symbolic integration from data, while Polu & Sutskever (2020) and

Drori et al. (2022) extended these ideas to code synthesis and instruction-tuned symbolic problem solving, respectively. Our approach similarly treats symbolic language as a learnable output space, but focuses on prefix-form arithmetic expressions under constrained operator vocabularies.

From a structural perspective, this system departs from semantic parsing models targeting formal languages like SPARQL or lambda calculus (e.g., Dong & Lapata, 2016), where tree structure and grammar constraints are enforced explicitly. By contrast, our model emits unstructured flat sequences and relies purely on pattern learning and cross-entropy training to recover symbolic syntax.

Unlike models such as Minerva or Tree-of-Thought prompting (Yao et al., 2023), which emphasize multi-hop reasoning and latent memory, this system performs direct symbolic transduction with no intermediate derivational steps or interpretability scaffolds. It also excludes reward-based fine-tuning, constrained decoding, and grammar augmentation — making it representative of a clean, fully supervised baseline architecture for prefix-style symbolic generation.

7. Limitations and Future Work

The architecture examined in this study reflects a deliberate tradeoff between modeling simplicity and symbolic precision. By adopting a flat sequence-to-sequence formulation with unconstrained greedy decoding, the system avoids structural supervision and grammar enforcement, placing the burden of syntactic correctness entirely on learned token patterns. While this yields a lightweight and general-purpose symbolic transducer, it introduces several known limitations.

First, the decoder operates over a free-form output space with no structural scaffolding; as a result, the model has no innate awareness of function call boundaries, operator arity, or argument ordering beyond what is implicitly learned during training. This contributes to failure cases involving malformed expressions, operand inversions, and mismatched argument counts — particularly on inputs that diverge syntactically or semantically from training distributions. Such brittleness is well-documented in unstructured neural program synthesis and unconstrained semantic parsing (Yin & Neubig, 2017; Krishnamurthy et al., 2017).

Second, the model exhibits limited generalization on compositional or nested input constructions. Because prefix-form symbolic expressions are generated token by token in a flat sequence space (lacking structural validation or latent tree representation) the system struggles to extrapolate hierarchical form when trained solely on symbolic targets.

Several extensions could address these constraints. Constrained decoding via symbolic grammars or output templates may improve syntactic validity at generation time (Yin & Neubig, 2017). Execution-guided decoding — in which candidate outputs are validated or reranked based on symbolic correctness — has shown promise in related domains (Chen et al., 2021).

Reinforcement learning approaches, such as reward shaping based on symbolic evaluation, could offer alternatives to pure cross-entropy loss, enabling optimization over semantic fidelity rather than string accuracy. More structurally aware architectures — such as tree decoders or syntax-aware attention mechanisms — may further enhance generalization by explicitly modeling symbolic hierarchy and operator structure.

Taken together, these directions suggest a path forward for neural-symbolic systems that retain the flexibility of pretrained language models while gaining the robustness of formal symbolic reasoning. The present system offers a minimal baseline: fully supervised, unconstrained, and interpretable — a foundation from which more structured, compositional, and semantically rigorous models can be developed.