ADVANCING MATHEMATICS RESEARCH WITH GENERATIVE AI

LISA CARBONE

ABSTRACT. The main drawback of using generative AI models for advanced mathematics is that these models are not primarily logical reasoning engines. However, Large Language Models, and their refinements, can pick up on patterns in higher mathematics that are difficult for humans to see. By putting the design of generative AI models to their advantage, mathematicians may use them as powerful interactive assistants that can carry out laborious tasks, generate and debug code, check examples, formulate conjectures and more. We discuss how generative AI models can be used to advance mathematics research. We also discuss their integration with neuro-symbolic solvers, Computer Algebra Systems and formal proof assistants such as Lean.

1. INTRODUCTION

Mathematicians have mixed views about the role of generative AI models in the mathematical landscape. While these models, such as Large Language Models (LLMs), can look convincingly like they replicate known mathematics, on careful scrutiny, it is apparent that they are just masters of the *rhetoric* of mathematics. They don't meet the standards of rigor that the field requires.

This limitation is inherent to the architecture of LLMs. They are fundamentally statistical, not logical, engines. As next-word predictors, they lack a built-in engine for formal symbolic deduction, meaning their emergent 'reasoning', called *Natural Language Reasoning*, is derived from statistical patterns in their training data.

However, this statistical foundation also provides certain benefits, as LLMs can pick up on patterns in higher mathematics that are difficult for humans to see. In particular, they have a learned geometric representation of mathematical language. By leveraging these capabilities and working with LLMs as they were designed to operate, mathematicians can use them as powerful interactive assistants.

The AI landscape is rapidly evolving beyond standard LLMs, with Large Reasoning Models (LRMs) and Large Context Models (LCMs) emerging as the next generation of this technology. While built on the foundation of LLMs, these models have distinct architectures and methods of training. In particular, LRMs, such as Gemini, use hybrid neuro-symbolic systems which combine text generation with verification (see Subsection 3.8).

In Section 2, we describe the capabilities and limitations of LLMs and the nature of their training data. In Section 3, we explore the mechanics of generative AI, from the management of context windows to the geometric representation of mathematical language within high-dimensional embedding spaces. We also discuss the evolution of architecture from standard language models to Large Reasoning Models (LRMs) that utilize neuro-symbolic verification.

In Sections 4 through 6, we examine methods for influencing the outputs of these models. We discuss how they simulate computational environments and how prompt engineering can impose logical order on probabilistic outputs (Section 5).

Date: December 17, 2025.

In Section 7, we discuss how generative AI models are useful for questions in Combinatorial Group Theory. In Section 8 we discuss the integration of LLMs with Computer Algebra Systems (CAS) and formal proof assistants such as Lean.

Finally, in Section 9, we imagine a collaborative research partnership where a mathematician engages in an iterative dialogue with AI to discover new ideas and research directions.

2. Generative AI models

2.1. **Mathematical training data for generative AI models.** Training data for generative AI models includes web documents, code with mathematical content, textbooks, online faculty-authored lecture notes and course materials, solutions to problem sets, as well as academic and scientific journal papers and content from arXiv. Google's data collection from Google Books, for example, is unparalleled in its scale.

The training data includes the standard undergraduate mathematics curriculum from US and other universities, as well as the standard coursework curriculum for PhD programs in mathematics from US and other universities.

Differences in the content and handling of training data of AI companies are primary reasons why various AI models have distinct strengths, weaknesses, and 'personalities', especially in a specialized domain like mathematics. Curation and filtering of data, such as removing low-quality content and emphasizing trusted sources, is essential. However, the training material for each model and its handling remains a closely guarded trade secret.

A key limitation is that a generative AI model's knowledge base is heavily biased towards materials that are easily digitized.

2.2. **How much do generative AI models 'know'?** Current generative AI models are trained on human data, which means they cannot generate knowledge outside of that data. While they can go beyond the knowledge of any single human, they can only come up with new ideas through extrapolation, not discovery from first principles. Their core capability is *Natural Language Reasoning* - the ability of generative AI models to process, synthesize and generate human language by identifying and replicating statistical patterns from vast amounts of data in the form of tokens.

In contrast, *Symbolic Deduction* is manipulation based on formal rules applied to abstract symbols, as in the operation of a Computer Algebra System. Current LLMs are built to imitate reasoning based on patterns, but are incapable of performing true symbolic deduction.

However, just as mathematicians are becoming familiar with the use of generative AI models, their internal architecture is already changing.

In 2024, Google's DeepMind reported silver-medal level on International Mathematical Olympiad problems using AlphaProof and AlphaGeometry. In 2025, DeepMind reported gold-medal level using Gemini 'Deep Think'. These systems did not use a separate formal proof assistant. The key innovation in their new model is the integration of Symbolic Deduction with Natural Language Reasoning, representing a first step towards an AI-based alternative to formal proof assistants. Their new models also incorporate techniques like parallel thinking, which allowed them to explore multiple solutions simultaneously.

The significant advancement in reasoning capability has been achieved by neuro-symbolic systems that use an advanced form of Chain-of-Thought and Tree-of-Thought reasoning with a feedback loop for verification.

The next frontier goes even further, with systems like Google's AlphaZero, which is not based on human-generated data. AlphaZero is given only the rules and foundations of a subject. It then learns (for example, in chess) by self-play and it generates its own training data. It is therefore likely that a future 'AlphaMath' system could be trained only on the axioms and rules of inference of ZFC set theory. Such a system would engage in genuine 'self-discovery' by attempting to prove theorems. Over time, with human feedback, it would begin to recognize patterns and strategies that lead to successful proofs.

2.3. **Drawbacks of using LLMs in mathematics.** The primary drawback of using LLMs for mathematics is that they are probabilistic pattern-matchers, not logical reasoning engines. Their core function is to predict the next most likely word or symbol in a sequence, not to apply deterministic mathematical rules. Generative AI models are experts at reconstructing what a proof should 'look like'. This limitation leads to several key problems:

Mathematical hallucinations: AI models generate outputs that look plausible and are often formatted correctly but contain nonsensical logic, invented theorems, or critical errors. They can state these falsehoods with the same confident authority as factual information.

Lack of true reasoning: Unlike a Computer Algebra System, an LLM cannot manipulate symbolic expressions according to deterministic rules. This makes it prone to subtle but serious errors in algebra, analysis, arithmetic, and logic. It is fundamentally incapable of handling verification.

Propagation of errors: AI models learn from their training data, which includes errors and misconceptions found online. For example, it is known that ChatGPT 4.0 contains training data from retracted scientific papers [Ana25]. An LLM will inherit and reproduce these mistakes, making them unreliable for tasks that require rigor.

3. Foundations of Generative AI

3.1. **Tokens: the building blocks of language.** In AI, a *token* is the fundamental building block of language. AI models convert input materials (prompts) into tokens. A token t_i is a digital representation of a word, a symbol, punctuation, a diagram, or other input.

Let V be a finite vocabulary consisting of token representations of the training data for an LLM. An *input* to an LLM is a sequence of tokens $(t_1, t_2, \ldots, t_n)$. An *output* is a sequence of tokens $(t_1, t_2, \ldots, t_m)$, where each $t_i \in V$.

The fundamental operation of an LLM is to compute the probability of the next token, t_{k+1}, conditioned on the sequence of all preceding tokens $(t_1, t_2, \ldots, t_k)$.

All AI language models have a maximum token limit for input and output. The token input limit has an impact on the work possible in a single chat session.

3.2. **The LLM as a probabilistic knowledge graph.** Consider the following simplified view of a generative AI model as a 'knowledge graph' which represents all possible connections and probabilities within the LLM's knowledge base.

Each word is a node in the graph. Each edge is a statistical relationship or an association between words, which cluster to form concepts.

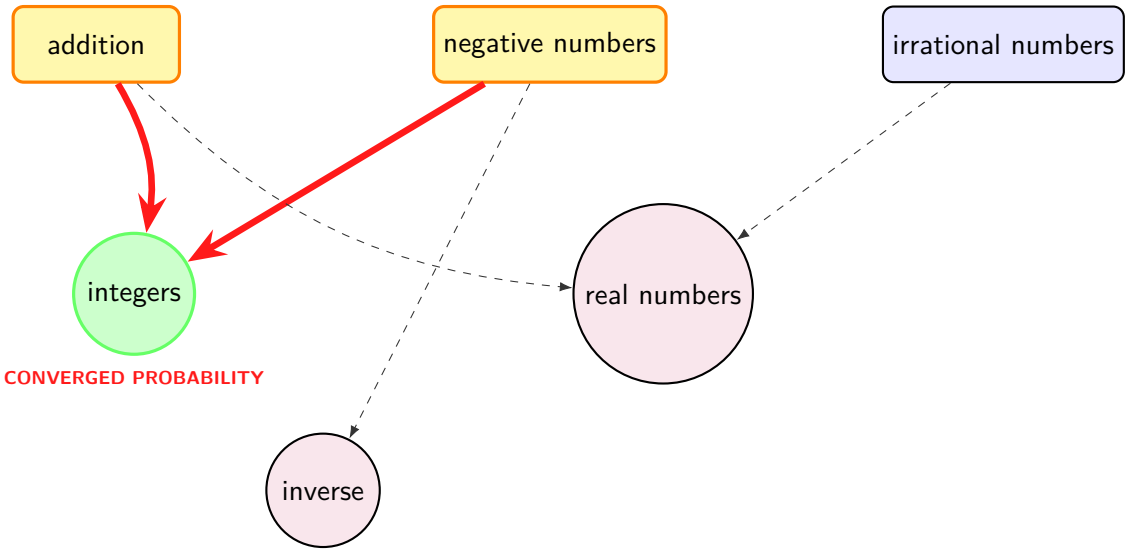


FIGURE 1. A simplified view of an LLM as a 'knowledge graph'.

Weighted edges between trillions of nodes represent how words or concepts are related. The weights are probabilities indicating the likelihood of connection.

An LLM generates text by predicting the next word, starting from the prompt, while calculating the probabilities of all possible next nodes. It then chooses one of the most likely paths through the knowledge graph.

3.3. **Generative AI models and patterns in higher mathematics.** A generative AI model can identify mathematical patterns that are not obvious to humans. It performs massive scale statistical analysis on the symbolic structure of mathematics itself. This allows it to simulate a mathematician's intuition. It can find cross-disciplinary correlations and potential analogies that humans may not think of.

The way mathematicians are trained to think assumes the progression of ideas of mathematical discovery. We have learned concepts in a chronological sequence. AI models do not 'know' that mathematical fields developed sequentially. For an AI model, mathematical concepts coexist simultaneously. They are linked only by the statistical and structural patterns embedded in the language of mathematics. Due to their inherent design, they can propose unexpected pathways, connections and points of view.

There are advantages of this capability. A generative AI model sees mathematics as a language with a complex grammar and vocabulary composed of symbols and diagrams. It learns the statistical relationships between the tokens in this language.

Trained on the entire available library of mathematics and science without disciplinary boundaries, an LLM learns the statistical relationships between tokens, allowing it to explore the entire space of mathematical connections, not just those that individual mathematicians are trained to look for.

3.4. **The embedding space for an LLM.**

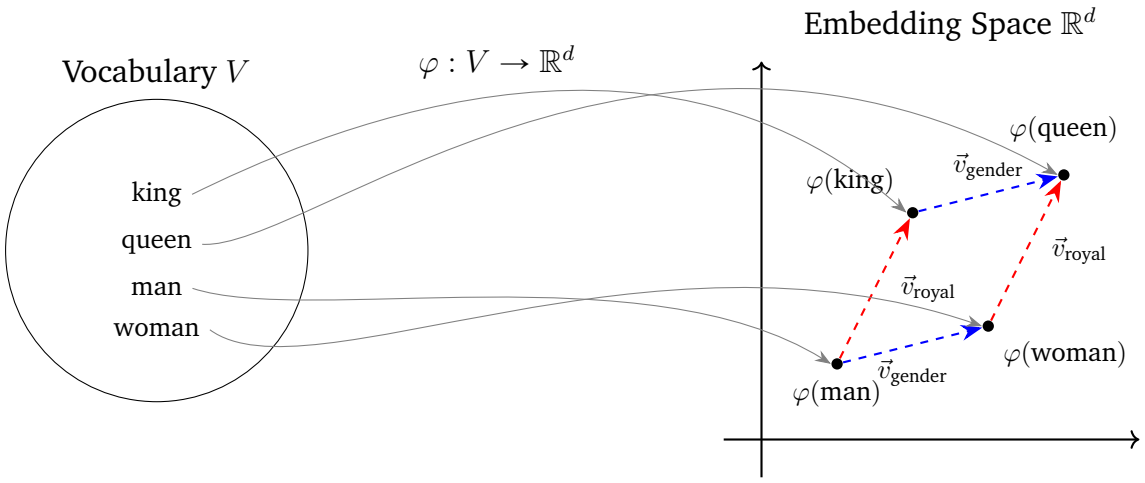


FIGURE 2. A hypothetical embedding for $d = 2$.

Generative AI models don't 'see' symbols and equations. There is a map $\varphi : V \to \mathbb{R}^d$ (for $d \leqslant 13,000$ for large models) from the vocabulary V of tokens to a high-dimensional vector space (the *embedding space*). The dimension d is a model parameter. Relationships in the vocabulary are translated into geometric relationships in $\mathbb{R}^d$ [MCCD13].

The model can detect geometric relationships in $\mathbb{R}^d$ that are difficult for humans to visualize for $d >> 0$ and that are not obvious from the symbolic definitions.

3.5. **The path of a word w through an LLM.** Let $\varphi(w)$ denote the initial embedding of the token word w in $\mathbb{R}^d$. Its 'meaning' is its relationship to all other images of tokens in the space and their locations. Furthermore, in advanced contextual models, this meaning changes with context.

The model uses linear transformations T_q, T_k and T_v to project each token embedding $\varphi(w) \in \mathbb{R}^d$ into three specialized vectors which have different purposes: a query vector q representing 'what I am looking for', a key vector k encoding 'what topic I represent' and a value vector v representing 'what information I contain'. The maps

$$T_q, \quad T_k, \quad T_v : \mathbb{R}^d \to \mathbb{R}^d,$$

project $\varphi(w)$ to three specialized vectors:

$$q = T_q\varphi(w), \qquad k = T_k\varphi(w), \qquad v = T_v\varphi(w).$$

The model determines relevancy scores by measuring the dot product between the query vector q of a token and the key vectors k_i of every token in the sequence. If q and some k_i point in similar directions, they obtain a high relevancy score.

It then uses these scores to create a new, context-rich vector as a weighted average of the value vectors. This new vector is then fed through a piecewise linear map to produce the output vector, $\zeta^{(\ell)}(w)$ for level ℓ of the model.

The process is repeated through many layers of the model; the output $\zeta^{(\ell)}(w)$ becomes the input for layer $\ell + 1$. The final contextualized representation, $\zeta^{(N)}(w)$ is the output of the final layer N and is different for every sentence it appears in.

The model generates an output sentence token-by-token in a loop. To predict the next token in the output, it uses the entire output sequence generated so far as its input. It then takes the final contextualized representation $\zeta^{(N)}(w_s)$ of the last token w_s from that sequence, passes it through one final linear map, and adds the most probable next token to the output sequence. This becomes the input for the next step.

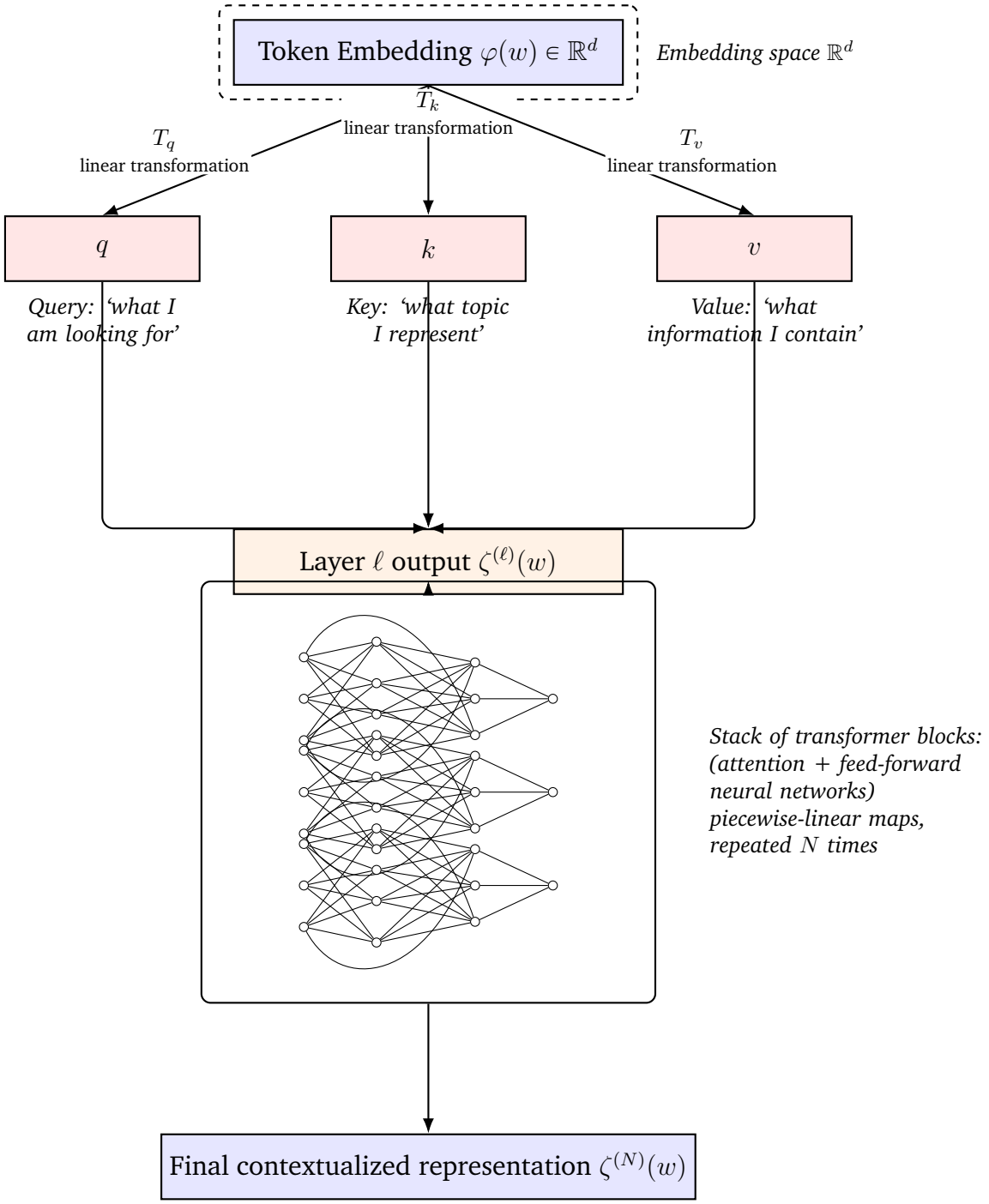


FIGURE 3. Projection of a token embedding $\varphi(w)$ in the embedding space $\mathbb{R}^d$ into specialized vectors (q, k, v) via linear transformations T_q, T_k, T_v, generation of the layer-ℓ output $\zeta^{(\ell)}(w)$, and repeated processing through a stack of transformer (neural network) blocks, including the feed-forward piecewise-linear map, producing the final contextualized representation $\zeta^{(N)}(w)$.

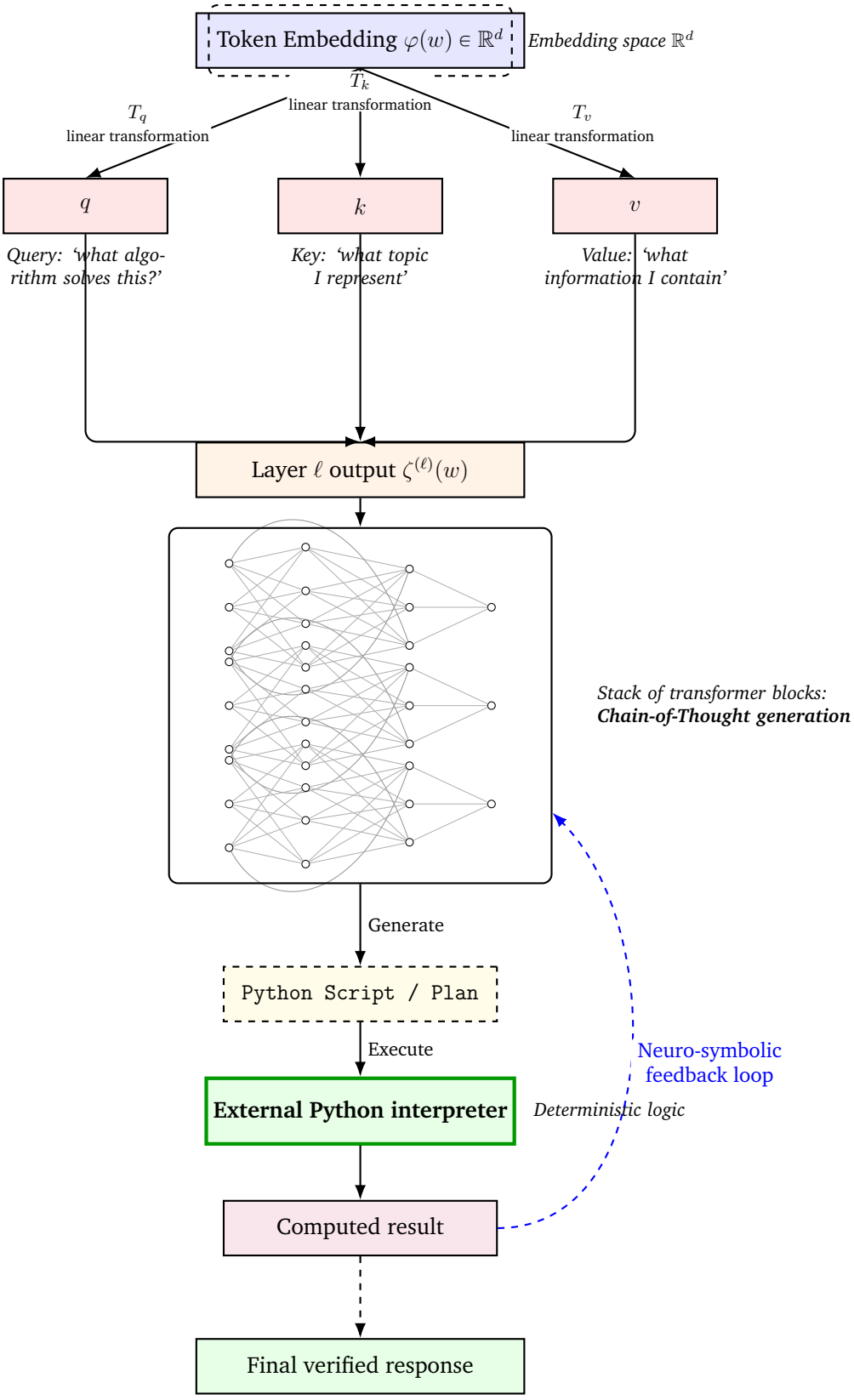


FIGURE 4. Unlike standard LLMs which predict tokens directly from the transformer stack, an LRM generates a computational plan (Python script), executes it in an external deterministic sandbox, then integrates the result back into the neural stream for the final response. Verification is built-in. The model no longer predicts mathematical answers. It predicts the algorithms needed to find the answers and it executes the algorithms.

3.6. **Structure of the point cloud of token embeddings in an LLM.** Let

$$T^{(0)} := \{\varphi(w) \mid w \text{ a token}\} \subset \mathbb{R}^d$$

denote the *finite point cloud* of embedded tokens. The *Manifold Hypothesis of Machine Learning* [TdSL00] asserts the existence of a smooth n-dimensional embedded submanifold

$$M^n \subset \mathbb{R}^d$$

such that the probability distribution supported on $T^{(0)}$ sits on a neighborhood of M.

In [RDC25], the authors find that points in $T^{(0)}$ that are close in Euclidean distance exhibit 'incompatible apparent dimensions'. That is, one point may have many nearby neighbors in $T^{(0)}$, while the other has far fewer.

This cannot occur if the point cloud is sampled from (or near) a single smooth embedded manifold $M^n \subset \mathbb{R}^d$. They conclude that there exists no low-dimensional smooth manifold M in $\mathbb{R}^d$ whose neighborhoods model the local statistical structure of the token embedding point cloud. That is, the Manifold Hypothesis is violated.

Each transformer layer in the LLM implements a map

$$f_\ell : \mathbb{R}^d \to \mathbb{R}^d$$

built from linear projections (including T_q, T_k, T_v), attention mixing, and a piecewise-linear feed-forward network.

Define the point cloud at layer ℓ to be

$$T^{(\ell)} := f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1(T^{(0)}).$$

The observed behavior is compatible with the point clouds $T^{(\ell)}$ sampling neighborhoods of a *stratified subset*

$$S = \bigsqcup_i S_i \subset \mathbb{R}^d,$$

where the strata S_i have different dimensions and meet along singular loci. The singular points are those points where the space is not smooth or locally Euclidean. Near such points, no Euclidean ball intersects S in a way resembling an open ball in $\mathbb{R}^n$ for any n.

The authors show that the maps f_ℓ preserve the types of singularities already present in $T^{(0)}$. The singularities correspond to polysemous words (words with multiple meanings). The different semantic regions of the embedded point cloud corresponding to polysemous words are 'pinched' together [JGZ20]. Near the singularities, the LLM is unstable and 'unusually sensitive', leading to unexpected associations and incoherent reasoning. This may be one of several causes of model hallucinations.

Smooth embedded manifold
(locally homeomorphic to an interval)

Stratified embedding space
(unions of pieces with singularities)

FIGURE 5. On the left, (in blue) a smooth 1-dimensional manifold embedded in $\mathbb{R}^2$, where every point has a neighborhood homeomorphic to an open interval. On the right, (in red) a stratified subset of $\mathbb{R}^2$, hypothetically resembling an LLM token embedding space. It is a union of 1-dimensional strata crossing at a singular point and an isolated 0-dimensional stratum.

3.7. **A single chat is a 'context window'.** A generative AI model's short-term memory is its context window, which consists of a single chat session. This window contains the entire conversation history, including uploaded documents and any corrections made by the user. Its capacity is measured in tokens: most standard models offer a 120K window which can hold a novel. Models like Gemini offer an unprecedented 1 million token input limit.

However, this memory is finite. Once the token limit is reached, the AI model will begin to forget the earliest messages in the conversation. Work on a difficult mathematics problem must be contained within a single chat: the AI model needs all information and previous steps to be present simultaneously to solve a complex problem.

Since a generative AI model has no inherent memory between separate sessions, new chats must be initiated with any updated information and results.

3.8. **Large Reasoning Models and Large Context Models.** LLMs are predictive and fast. They 'guess' the end result but they may not do all the intermediate work.

Large Reasoning Models (LRMs), are a class of LLMs trained to carry out tasks that require multi-step deduction, critical thinking, and structured problem-solving. LRMs deliberate, explore various solution paths, evaluate intermediate steps and revise their reasoning.

LRM models integrate frameworks that support reasoning structures, such as 'Chain-of-Thought' and 'Tree-of-Thought' structures into their systems. We give some examples in Section 5.

The training of LRMs diverges from that of LLMs. They utilize a technique called 'process supervision'. Unlike the training process of LLMs that only rewards a correct final answer, process supervision rewards the correctness of the intermediate reasoning steps.

LRM models also have self-correcting capabilities. Their basis is a neuro-symbolic system which combines text generation with verification (see Figure 4). They have a feature known as 'self-awareness'. LRMs are able to distinguish between requests that are language-based or computational. They make 'decisions' about delegating tasks to external tools such as a Python interpreter. They don't try to predict mathematical answers, they predict the algorithms needed to find the answers, then they execute the algorithms and report back with the result.

Large Context Models (LCMs) are defined by their large context windows and the volume of information that they can handle. This allows them to accept and analyze large datasets, entire codebases, or multimodal inputs, without requiring external retrieval systems. Their training data includes long-form content to improve their ability to track long-range dependencies between input tokens.

These advanced capabilities are becoming integrated into leading AI platforms. The cost, however, is additional computing power and increased time for responses.

Google's Gemini models demonstrate both LRM and LCM characteristics. These models also use internal 'thinking processes' for improved reasoning.

ChatGPT has a 'reasoning mode' in its most recent model. Reinforcement learning was used to teach the model to 'think' before generating answers, using what OpenAI refers to as 'private chain-of-thought' processes built into their system prompts. This allows the model to plan ahead and reason through tasks, performing a series of intermediate reasoning steps to assist in solving the problem.

Claude models use 'extended thinking' or the ability to 'think out loud'.

These architectural advances are complemented by the practical integration of LLMs with symbolic computation tools.

4. Computational Capabilities

4.1. **AI models simulate Computer Algebra Systems.** Advanced generative AI models simulate a scientific computing environment by delegating the computation to specialized Python libraries.

Table 1. Key Python libraries used by generative AI models for mathematics

Library	Main purpose	Key capabilities & LLM use
NumPy	Numerical Computing	Handles arrays & matrices. Used by generative AI models for linear algebra (matrix multiplication, row reduction, eigenvalues).
SymPy	Symbolic Mathematics	Performs precise algebra & calculus. Used for symbolic derivatives, integrals, and simplifying expressions.
SciPy	Scientific Computing	Provides advanced numerical routines. Used for optimization (finding minima/maxima) and solving differential equations.
Pandas	Data Analysis	Manages structured, table-like data. Used for reading and analyzing data from files such as Excel.
Matplotlib	Plotting & Visualization	Creates a wide variety of 2D graphs and charts. Used to plot functions and visualize data.

The Python code is written by the LLM using probabilistic methods. The code is then executed by a standard Python interpreter. The reliability of the final answer comes from having outsourced the computation to the verified, deterministic Python libraries. The potential unreliability comes from the LLM's probabilistic process of writing the code that uses those libraries.

For example, the LLM may hallucinate functions that don't exist in the Python libraries. In addition, the queries may give incorrect answers if the problems are inaccurately posed. The LLM could

misunderstand the prompt, have bugs in its code or fail to handle boundary cases. Debugging AI-generated Python code may be needed.

Gemini and the paid version of ChatGPT give automatic user access to the Python packages NumPy, SymPy, SciPy, Pandas, Matplotlib.

4.2. **Comparison of models.** Gemini, ChatGPT and Claude all exhibit different specializations in how they activate nodes, navigate paths and narrow down possibilities within their internal graphs of statistical associations.

For intermediate mathematical tasks, Gemini, ChatGPT, and Claude are all roughly in the same class with respect to performance. Though Gemini models performed slightly better in the standard benchmarks for competition-level mathematics, testing symbolic reasoning in algebra, calculus and number theory in 2024.

ChatGPT integrates features like WolframAlpha, which is a powerful Computer Algebra System, which can be enabled in the professional version. However, WolframAlpha is not designed for multi-step algorithms and it times-out on long tasks. Also, WolframAlpha is not optimized for integers. It makes errors with integers larger than 10^{15}.

On the other hand, Python interpreters, available in Gemini and ChatGPT (see Subsection 4.1), have 'arbitrary-precision' for integers. They automatically adjust the memory used to store an integer as its value grows. We note that Python is not enabled in WolframAlpha.

Another ChatGPT feature called SciSpace has access to hundreds of millions of peer-reviewed journal papers.

5. Prompt engineering for mathematics

5.1. **How the prompt shapes an LLM's output.** The prompt provides an initial sequence of tokens that the model uses to condition its output. The LLM generates text by sequentially examining the existing sequence of tokens, assigning a probability to every possible next token, and then choosing one to add to the sequence.

The probability of each potential next token is calculated based on the full sequence of tokens that came before it – both the original prompt and any text already generated.

The final output is the result of a chain of these probabilistic decisions. The framing of the initial prompt is critical in determining the usefulness of the output.

The design of prompts, or 'prompt engineering', significantly influences the behavior of an LLM [Ram25], [Dai25]. It disambiguates the input, leading to more relevant and useful output.

Structured prompts impose a logical framework on an LLM's probabilistic generation process. The examples of prompts in the following subsections have been shown to be useful in guiding generative AI models on how to approach complex mathematical questions [KGR+22]. These methods are also used internally in advanced LRM models like Gemini and ChatGPT (in thinking mode).

5.2. **n-shot prompts.** An *n-shot prompt* provides the LLM with n examples of the task that you would like it to perform.

Example: Zero-shot prompt

Determine the arithmetic progression corresponding to primes of the form $4n + 1$ with last digit 1, (such as $41, 61, 101, \dots$).

Example: One-shot prompt

Given that primes of the form $4n + 1$ ending in 1 (such as $41, 61, 101, \dots$) correspond to the arithmetic progression $20k + 1$, determine the arithmetic progression corresponding to primes of the form $4n + 1$ with last digit 3 (such as $13, 53, 73, 113, \dots$).

Giving the LLM an example makes it more likely to give a correct answer.

5.3. **Chain-of-Thought reasoning.** This refers to the LLM thinking step-by-step, generating intermediate reasoning before arriving at the final answer [Gad25]. If the LLM checks its work, this reduces errors.

Example prompt:

Your task is to generate a detailed description of the representation of the Weyl group of $\mathfrak{sl}_3(\mathbb{C})$ on the dual space $\mathfrak{h}^*$ of its Cartan subalgebra $\mathfrak{h}$.

Start by describing the Coxeter presentation of the group $W(A_2) \cong S_3$ which has generators s_1, s_2 and relations $s_i^2 = e$ and $(s_1 s_2)^3 = e$. Next, define the vector space $\mathfrak{h}^*$ and specify its basis of simple roots, α_1, α_2.

Then define the representation ρ of $W(A_2)$ using the fact that the generators act as reflections on $\mathfrak{h}^*$.

Use the Weyl reflection formula to explicitly derive the matrices for $\rho(s_1)$ and $\rho(s_2)$ in the basis for $\mathfrak{h}^*$.

With this information, the next step is to verify that this is a valid representation. Please do this by performing the necessary matrix calculations to show that these matrices satisfy the defining relations of the group $W(A_2) \cong S_3$.

An LRM may carry out internal reasoning as in Figure 6. A LLM can be explicitly prompted to use Chain-of-Thought reasoning.

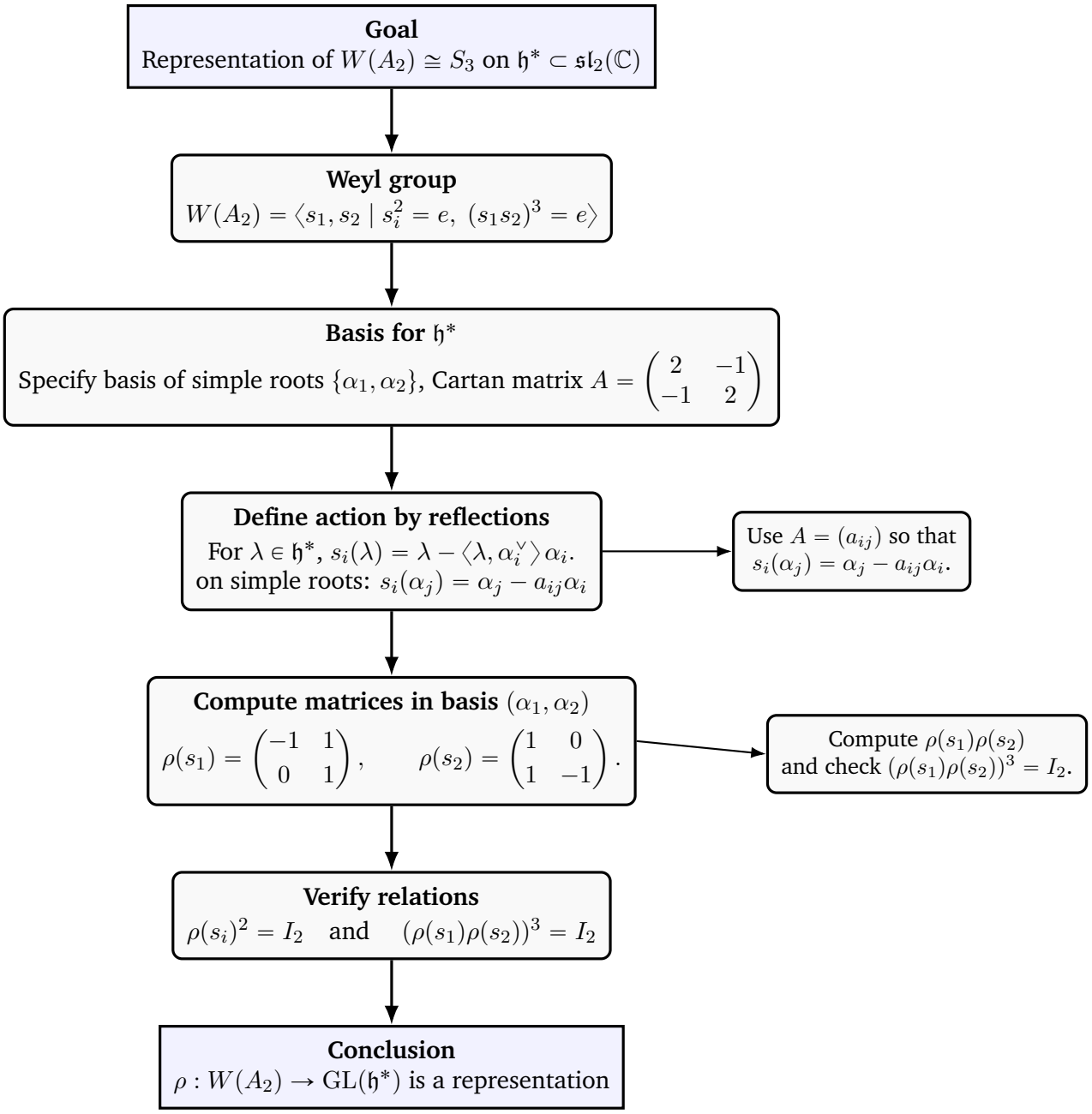


FIGURE 6. Chain-of-thought diagram.

5.4. **Tree-of-Thought reasoning.** This refers to an AI model exploring multiple different deduction paths simultaneously, such that it evaluates its own progress at each step and pursues the most promising path [Gad25].

Example prompt:

Let p be a prime number, written $p = a_n a_{n-1} \dots a_1 a_0$ in terms of its digits in base 10. Suppose that $n > 1$, that is, suppose that p has at least two digits. Your task is to determine the possible values of the last digit a_0.

An LRM may carry out internal reasoning as in Figure 7. A LLM can be explicitly prompted to use Tree-of-Thought reasoning.

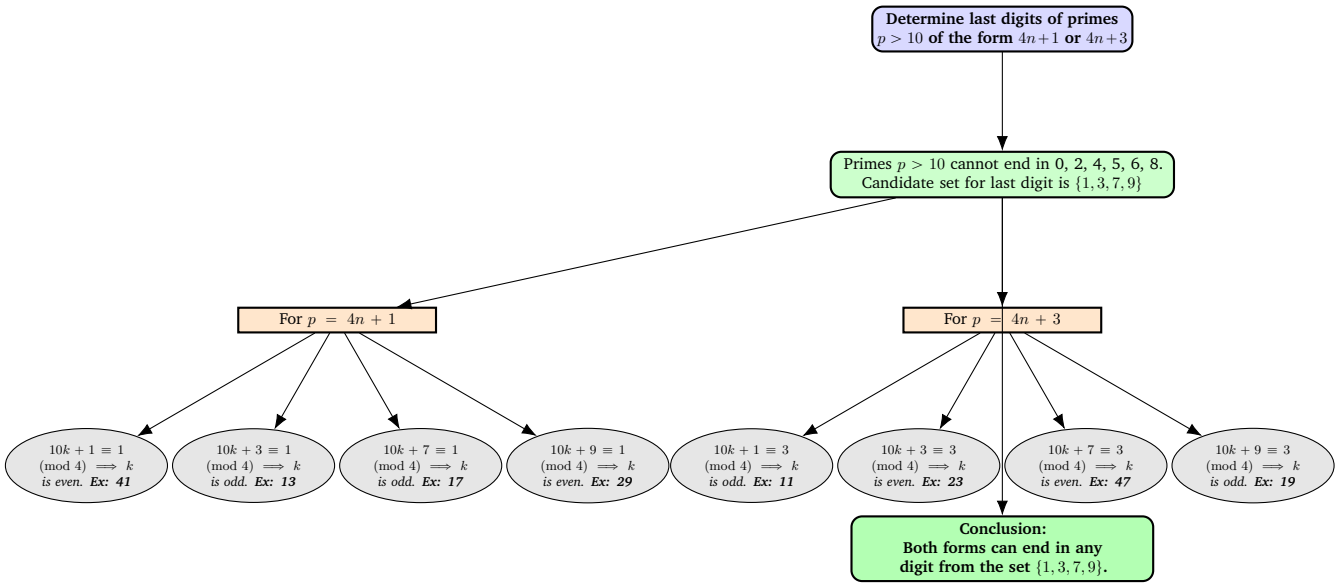


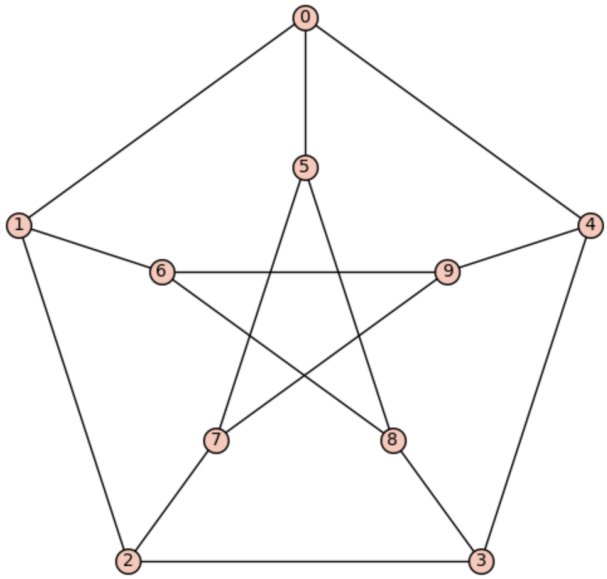
FIGURE 7. A Tree-of-Thought diagram for determining the possible last digits of primes $p > 10$.

5.5. **Using LLMs to generate Python code to run in SageMath.** Generative AI models can generate code for Computer Algebra Systems such as GAP, Magma, and SageMath with varying degrees of reliability. Proficiency is particularly high with SageMath, given its Python-based syntax and integration with GAP for computations in group theory, ring theory and field theory.

Example prompt:

Generate Python code to find the minimum number of colors needed to color the Petersen Graph such that no two adjacent vertices share the same color (the 'chromatic number'). Then create the Petersen Graph using the built-in SageMath generator. Find its chromatic number. Display the graph.

```
Working with the Petersen Graph.
The minimum number of colors needed is: 3
Displaying the graph...
```



The emergence of LRMs and LCMs (Section 3.8) has altered the practice of prompt engineering.

For LRMs, the need for prompt engineering techniques designed to force reasoning, such as explicit Chain-of-Thought prompting, is reduced. The reasoning process is handled internally. Simpler, goal-oriented prompts that clearly define the problem may be more effective.

With LCMs, the focus is shifting to 'context engineering'. The challenge becomes optimizing the content and structure of the information provided in the prompt. This maximizes performance and prevents recall accuracy from decreasing as the context window fills.

5.6. **Targeted prompts.** You can ask a generative AI model to:

- Give an explicit reference (with page numbers) on a topic known to be in some book or research paper.
- Apply the statement of a theorem to a particular example.
- Generate LaTeX code from an uploaded pdf file of a math paper.
- Translate a mathematics paper from another language and give the output in a LaTeX file.
- Explain a section of a physics paper in mathematical language.
- Generate a Tikz diagram or table from a description in natural language.
- Generate a bibliography on a specific topic.
- Rewrite the LaTeX code for an entire math paper in different notation.
- Find the typos in an uploaded pdf file.
- Analyze if the flow of ideas in a paper is appropriately sequential.
- Generate LaTex, Lean, Python, Mathematica, Maple and other forms of code.

5.7. **General rules for writing prompts for advanced math questions.**

- Specify the task in short sentences.
- Use 'your task is to...' or 'your goal is to...'.
- Specify the context of the task.
- Upload all necessary background information.
- Be explicit and detailed.
- Include all relevant keywords.
- Guide the reasoning process.
- Specify the output format.
- Verify the output.
- Never ask for a complex proof in one shot.
- Break a difficult task up into smaller tasks.
- Take the model's output and ask for it to be modified with specific constraints.
- Constrain the method the model is allowed to use.
- Find errors in the output and ask the model to self-correct.

You can also ask a generative AI model to generate a prompt for a given task.

6. Other ways to influence the output from a generative AI model

6.1. **Design your outputs.** In generative AI models, there is a 'system instruction' feature which can be used to influence the output. Users can enter a 'pre-prompt' with their own personal profile that governs the model's behavior for the entire chat session. Some models allow users to build AI agents that have persistent memory of settings. This capability allows for a type of 'back-end' engineering, where writing specific system prompts gives the user more direct control over the model's performance. Utilizing this feature can significantly impact the amount of detail provided in the response and the level of rigor maintained in a mathematical proof.

6.2. **Changing the 'knobs'.** System settings in certain generative AI models can also be manually changed in order to make mathematical proofs more rigorous and less random.

For mathematical rigor, the most important setting is 'temperature'. This controls the randomness of the output by reshaping the probability distribution. For research questions and formulating conjectures, a high temperature (such as 0.9) is preferable, as it encourages more diverse and novel responses. A low temperature (such as 0.2 or 0.1) makes the output more deterministic.

'Back-end' engineering, such as writing system prompts in a generative AI model gives more direct control and can impact the amount of detail and rigor in a proof.

6.3. **Fact-checking: retrieval-augmented generation.** The default output of a generative AI model is an output that is based on its training data. The ability to access the internet to fact-check in real time is a feature built on top of an LLM, through a mechanism called 'retrieval-augmented generation', or RAG. It may require a specific prompt in order to invoke this feature. Users can also explicitly request a list of websites and references used.

TABLE 2. Optimized settings in Google AI studio for research in mathematics

Temperature	**Medium-High** (0.7 - 1.0) A higher temperature encourages exploration.
Top P	**High** (0.95 - 1.0) Allows the model to consider a wider, more diverse set of 'next words'.
Thinking mode	**Advanced** Research problems require the deepest level of reasoning.
Set thinking budget	**Maximum** Gives the model time to explore the 'search space'.
Code execution	**On** Crucial for running checks with Python tools such as SymPy.
Grounding (Google Search)	**On** Allows the model to search outside its training data.
URL Context	**On** Allows you to give the model specific URLs as context prompts.
Structured output	**Off** Allows unstructured brainstorming.
Function calling	**On** Connect to external research tools such as a university library API or a computational algebra system like SageMath or WolframAlpha.
Media resolution	**High** Essential for correct reading of symbols in handwritten notes or complicated diagrams.
Output Length	**Maximum** Allows for an unencumbered output.

When asked a question that requires fact-checking, Gemini automatically queries the Google Search index in real time. It then synthesizes the information from the top-ranked pages to give an answer and often provides direct links.

The current version of ChatGPT has a 'reasoning mode' in its new model which autonomously decides when to use its web-browsing tools. Before finalizing an answer, the model evaluates and corrects its own results. It internally grades its answer against criteria to ensure it meets a high-quality standard.

Microsoft Copilot is designed to be an AI-powered search assistant. Most queries initiate a live search on Bing and links are usually provided.

7. CASE STUDY: COMBINATORIAL GROUP THEORY

7.1. **Why AI is a useful tool in Combinatorial Group Theory.** We discuss some examples from group theory. However, many of the underlying principles are applicable to other topics.

Let G be a group defined by a presentation $G = \langle X \mid R \rangle$, where X is a set of generators and R is a set of relations. This is a compact way to define a group, but it hides immense complexity.

The word problem (determining if a word in the generators is equal to the trivial element of the group) is unsolvable in general [MT73].

Computer Algebra Systems such as GAP, Magma, and SageMath can solve the word problem for specific families of groups using rewriting methods and structural properties.

A simple example: let $G = BS(1,2) = \langle x, y \mid x^{-1}yx = y^2 \rangle$. This is the so-called *Baumslag-Solitar group* $BS(1,2)$. Its word problem is known to be solvable.

A solution of the word problem involves deciding on an efficient sequence of applications of the group relations to achieve the goal of trying to get to the identity element of the group.

As an example, we solve the word problem for the word $w = yxy^{-2}x^{-1}$:

We first rewrite $x^{-1}yx = y^2$ as $yx = xy^2$. Then

$$yxy^{-2}x^{-1} = xy^2y^{-2}x^{-1} = xx^{-1} = 1.$$

Thus the word $w = yxy^{-2}x^{-1}$ represents the identity element in G.

7.2. **Solving algorithmic research questions with AI.** Algorithmic search problems in group theory, such as the word problem, conjugacy problem (deciding if two words are conjugate in a given group), and triviality problem (deciding if a group presentation presents the trivial group), are undecidable in general. Even in decidable cases, the search space for groups with many generators and intertwined relations is often astronomically large. The number of ways to apply relations to a word grows exponentially with the length of the word.

AI can address this by effectively simplifying words. It has strategies to navigate the search space by deciding which group relations are the most useful to apply.

Gemini uses Chain-of-Thought processes for working with group presentations. The model does not just guess the final simplified word. It generates intermediate steps internally using a type of depth-first search where the model explores a simplification path. It can self-correct if the word complexity increases rather than decreases.

Gemini translates the group presentation $G = \langle X \mid R \rangle$ into a Python script using the `sympy.combinatorics` module.

- It defines the free group on generators X.
- It inputs the relations R.
- It uses the library's implementation of the Knuth-Bendix completion algorithm (or similar rewriting systems) to attempt to find a normal form for the word.

The following is an example of a prompt explicitly asking the model to follow Tree-of-Thought reasoning.

Example prompt:

Your task is to determine if the Higman group, defined by the presentation

$$H = \langle a, b, c, d \mid a^{-1}ba = b^2,\ b^{-1}cb = c^2,\ c^{-1}dc = d^2,\ d^{-1}ad = a^2 \rangle$$

is trivial or non-trivial. Proceed with the following tree of reasoning, evaluating each step before proceeding to the next:

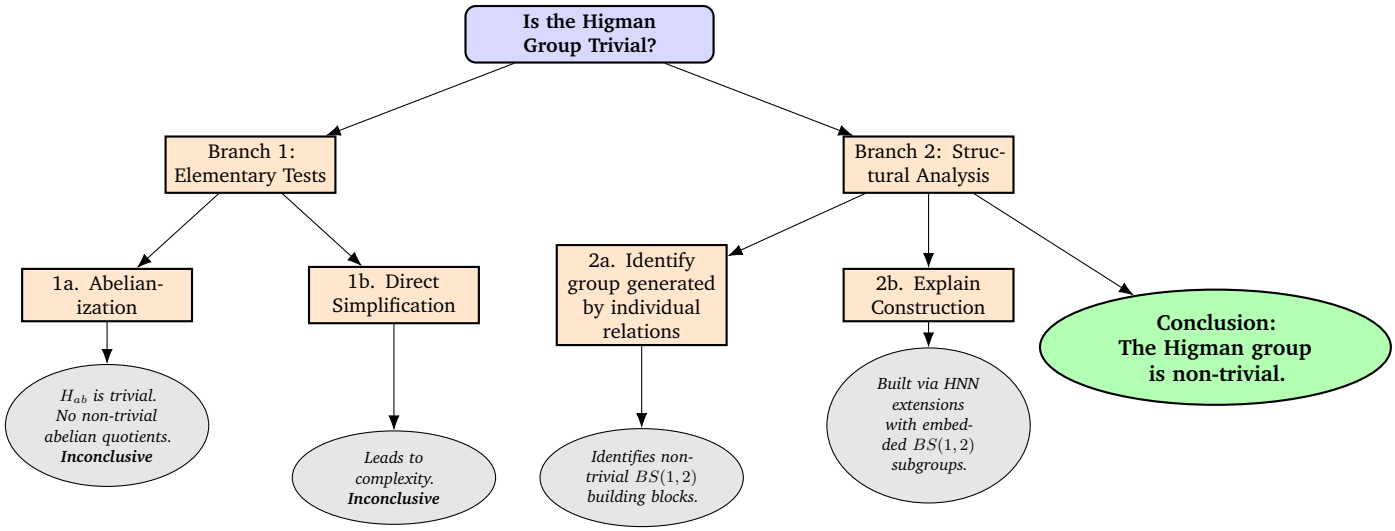


FIGURE 8. Tree-of-Thought prompt diagram for non-triviality of the Higman group.

8. HYBRID SYSTEMS: GENERATIVE AI MODELS WITH COMPUTER ALGEBRA SYSTEMS OR FORMAL PROOF ASSISTANTS

Because generative AI models are not substitutes for formal proof or verified computation, their outputs require rigorous human scrutiny. The primary way to offset this limitation is to use them in conjunction with formal tools like Computer Algebra Systems or formal proof assistants such as Lean. For example, the professional version of ChatGPT can integrate with WolframAlpha. The user can explicitly prompt ChatGPT in the Wolfram chat window, or allow it to decide when to outsource a computation.

An active area of research is to develop tools that link generative AI models with Computer Algebra Systems [KG25].

Formal proof assistants like Lean allow for the expression and mechanical verification of mathematical proofs. Lean implements a version of Dependent Type Theory known as the Calculus of Inductive Constructions (CIC). This is a constructive type theory, where in the Lean kernel (or core verifier) proofs are given explicitly, aligning with computation and the Curry–Howard correspondence. That is, every proof encodes an algorithm or a construction. However, in practice, Lean contains a model of ZFC set theory. It does not use ZFC axioms to check proofs, but with some additional axioms, it behaves like ZFC.

Type Theory is particularly advantageous for formalization because it embeds mathematical meaning directly into its syntax. Its system of Types prevents false statements and builds properties directly into the definition of its objects. In ZFC set theory, many of these logical statements require separate proofs.

However, writing Lean code is notoriously difficult, requiring not just programming skill but the precise formalization of abstract mathematical ideas in Type Theory. In this framework, even small logical gaps or unstated assumptions must be made explicit and rigorously verified. Precise use of syntax is required.

Most users use Lean *tactics*. These are metaprograms that automate steps and give instructions on how to construct proofs. Meanwhile, Lean generates the rigorous low-level code for proof-checking.

One of the benefits of the Lean environment is Mathlib: a vast, open-source digital encyclopedia of formally verified mathematics.

Lean Copilot is a framework that integrates LLMs into the Lean theorem prover to assist with formal proofs. It functions as an AI assistant by providing features like context-aware tactic suggestions. It selects relevant premises from Mathlib and searches for proofs.

Research labs are actively working on ways to make the task of generating Lean code easier: from using generative AI models to write Lean code, to developing interfaces that accept inputs in natural language.

For example, *Numina* is an assistant that acts as a bridge between natural language mathematics proofs and Lean. It uses AI to generate formal proofs and uses Lean to verify their correctness.

9. Mathematician + generative AI collaboration

9.1. **Human-LLM collaborative research.** Using a generative AI model as a collaborative research partner, rather than as a search engine or text generator, involves an iterative dialogue that must be repeatedly corrected and refined. The user can improve the model's output through a series of increasingly specific constraints, corrections, and questions, often in a loop that involves downloading, correcting, and re-uploading the model's work for further prompting.

For open research questions, this can be a lengthy process, sometimes involving hundreds of prompts in a single chat, to obtain a final product that is useful. A key technique is 'in-context learning', where researchers provide background information like papers or books via prompts. The LLM keeps this material in its short-term memory. It becomes more adept at handling the research topic as the conversation history forms an expanding context window.

While not necessarily a time-saving endeavor, this collaborative process can reveal ideas and connections that lie beyond a human researcher's own spectrum.

9.2. **Why mathematicians should collaborate with generative AI models.** As we have discussed, generative AI, on its own, should not be viewed as an oracle or an authoritative source for mathematical output. Verification of output is a necessary part of using an AI model for mathematics. But it can be used as a creative collaborator that provides ideas, perspective and potential new directions.

The most effective research strategy uses all available tools. When combined with a Computer Algebra System, AI can reliably perform laborious tasks and calculations. When paired with formal verification tools, it transforms the research process itself.

The author of this work now employs this collaborative framework in several LLM+CAS+Lean-assisted research projects in infinite dimensional algebra and group theory. The use of generative AI models has been particularly revealing in this context. There have been multiple instances where AI models have suggested novel research directions and ideas that had initially been overlooked or dismissed as implausible or irrelevant. Yet, these suggestions (once verified) turned out to yield unexpected paths forward.

Mathematicians could consider AI not merely as a search engine, but as a creative partner that can act as a catalyst for discovery.

References

[Ana25] Ananya, *AI models are using material from retracted scientific papers*, 2025. https://www.technologyreview.com/2025/09/23/1123897/ai-models-are-using-material-from-retracted-scientific-papers/.

[Dai25] DairAI, *Prompt engineering keywords and techniques*, 2025. https://promptengineering.org/what-are-prompt-keywords-or-magic-words/.

[Gad25] V. Gadesha, *Tree of Thoughts: IBM Topic Overview*, 2025. https://www.ibm.com/think/topics/tree-of-thoughts.

[JGZ20] A. Jakubowski, M. Gasic, and M. Zibrowius, *Topology of word embeddings: Singularities reflect polysemy*, 2020. https://arxiv.org/pdf/2011.09413.

[KG25] A. Khaitan and V. Ganesh, *O-Forge: An LLM + computer algebra framework for asymptotic analysis*, 2025. https://arxiv.org/abs/2510.12350.

[KGR+22] T. Kojima, S. Gu, M. Reid, Y. Matsuo, and Y. Tanaka, *Large Language Models are Zero-Shot Reasoners*, arXiv preprint arXiv:2205.11916 (2022). https://arxiv.org/abs/2205.11916.

[MCCD13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013). https://arxiv.org/abs/1301.3781.

[MT73] R. McKenzie and R. J. Thompson, *An elementary construction of unsolvable word problems in group theory*, Studies in logic and the foundations of mathematics, 1973, pp. 457–478.

[Ram25] S. Ramlochan, *Prompting techniques guide*, 2025. https://www.promptingguide.ai/techniques.

[RDC25] M. Robinson, S. Dey, and T. Chiang, *Token embeddings violate the manifold hypothesis*, 2025. https://arxiv.org/pdf/2504.01002v3.

[TdSL00] J. Tenenbaum, V. de Silva, and J. Langford, *A global geometric framework for nonlinear dimensionality reduction*, Vol. 290, 2000. https://www.science.org/doi/10.1126/science.290.5500.2319.

Department of Mathematics, Rutgers University, Piscataway, NJ 08854-8019, USA

E-mail address: lisa.carbone@rutgers.edu