LLM-Guider: A Language-Guided Discovery of Symbolic Pruning Metrics for Post-Training Sparsity in LLMs

Anonymous EMNLP submission

Abstract

Large Language Models (LLMs) have achieved remarkable advancements in natural language understanding, yet their mammoth size coupled with substantial training and inference costs can make them difficult to use in environments with limited resources. To address both memory and efficiency concerns, post-training unstructured sparsity techniques have emerged focusing on developing optimal pruning criteria to eliminate redundant weights while maintaining performance. However, these approaches often rely on manually crafted pruning criteria, leading to sub-optimal solutions due to heuristic oversimplifications. Therefore, we introduce LLM-Guider, a language-guided symbolic formula optimization framework that seeks to discover optimal pruning criteria through a transparent and systematic process. LLM-Guider comprises three interrelated stages: example selection, formula generation, and formula evaluation, which collectively enable the efficient exploration of the formula space. In addition, LLM-Guider enables incorporation of intuition, domain and mathematical knowledge through role prompts, hints and in-context examples. We also extend the standard set of aggregation strategies over calibration dataset, resulting in never-seen-before pruning metrics. Through extensive experiments, we demonstrate that formulas discovered through LLM-Guider is able to find formulas, which outpeform established baselines.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable natural language understanding and generation abilities with unprecedented accuracy and depth. The impressive performance of LLMs can be largely attributed to their scale, which depends on model parameters, dataset size, and amount of compute used for training (Kaplan et al., 2020). The scaling laws have enabled the development of large models such as GPT-175B (Brown et al., 2020) and beyond, boasting hundreds of billions of parameters. Although this has led to the emergence of new abilities in LLMs (Wei et al., 2022), the associated extraordinary training and inference costs pose major challenges for practical use, especially in resource-constrained settings. In order to address both memory and efficiency concerns, multiple model compression methods such as sparsity, quantization, and knowledge distillation have been proposed in the literature (Zhu et al., 2024). Model sparsity, either structured or unstructured, essentially focuses on pruning redundant weights while balancing performance versus model size trade-off. As model sparsity involves either training from random initialization (Hoang et al., 2023), retraining (Chen et al., 2023a), or extensive iterative pruning (Tanaka et al., 2020), post-training sparsity approaches (Sun et al., 2024; Zhang et al., 2024; Dong et al., 2024; Frantar and Alistarh, 2023) have become increasingly popular.

The essence of post-training unstructured sparsity techniques involves the development of an optimal pruning criterion that quantifies the significance of each weight, subsequently eliminating those weights that exhibit the lowest significance scores. A vast majority of approaches proposed in the literature require manual crafting of pruning criterion by utilizing weight magnitude (Cheng et al., 2024), activations (Sun et al., 2024; Zhang et al., 2024), and first/second order gradient information (Das et al., 2024; Dong et al., 2024). Moreover, development of these approaches relies heavily on the domain knowledge and inductive biases of the researchers, thus requiring extensive trial and error experimentation. In addition, these heuristic approaches are susceptible to oversimplifications often leading to locally sub-optimal solutions. Therefore, symbolic formula optimization (e.g., (Dong et al., 2024; Chen et al., 2024; Ruan et al., 2024)) has been gaining ground as it explores the search space more efficiently using search algorithms. PrunerZero (Dong et al., 2024) is one such post-training unstructured sparsity technique that employs a genetic algorithm to discover new symbolic formulas, outperforming manually crafted ones. Although genetic algorithms possess significant optimization capabilities, they focus primarily on the evolutionary process, but there's ample opportunity to enhance them by leveraging large language models, which have absorbed vast mathematical knowledge and intuitions and by dynamically tuning their behavior through natural language instructions.

Given the paramount importance of designing an optimal pruning criterion, we ask whether it is possible to develop a transparent, well-reasoned, language-guided discovery process leveraging the remarkable abilities of state-of-the-art LLMs. Language, a meticulously structured and codified form of human communication, uniquely characterizes human evolution, facilitating the preservation and exchange of ideas. Emulating humans, LLMs trained on high-quality data can store a vast amount of scientific knowledge, endowing them with the ability to write high-quality code, solve complex reasoning problems, and utilize tools out-of-the-box or in a zero-shot manner (Abhimanyu Dubey, 2024). The abilities of LLMs go far beyond standard information retrieval: they enable dynamic reasoning, which includes learning from evaluated examples (Brown et al., 2020) and external context (Lewis et al., 2021). With these capabilities, LLMs have already been successfully used to guide the discovery process in fields such as reward modeling (Ma et al., 2024) and preference optimization (Lu et al., 2024). Search algorithms, particularly those guided by LLMs, possess reasoning abilities along with mathematical and coding skills, rendering them indispensable for uncovering novel solutions and circumventing the local minima that frequently hinder heuristic methods.

To this end, we introduce LLM-Guider, a generic language-guided symbolic formula optimization framework aimed at discovering optimal pruning criterion for post-training unstructured sparsity in LLMs. LLM-Guider operates through three distinct yet interrelated stages: In the *Example Selection* stage, we identify the most promising k-shot seed formulas based on a predefined policy. The subsequent *Formula Generation* stage leverages the capabilities of LLMs, geared with customized hints and k-shot examples, to produce novel symbolic formulas tailored to our pruning objectives. Finally, in the *Formula Evaluation* stage, these generated formulas are rigorously assessed for their effectiveness in enhancing model sparsity while balancing the performance trade-off, with the highest-scoring formulas fed back into the generation pool for iterative refinement. Unlike genetic algorithms, this structured approach not only enables the dynamic exploration of the formula space but also allows for the integration of domain-specific insights through sampled hints, ensuring that our generated formulas are both novel and relevant. Our contributions are as follows:

- We propose LLM-Guider, a generic language-guided symbolic formula optimization framework, tailored for discovering optimal pruning criterion for post-training unstructured sparsity in LLMs. With minimal modifications, LLM-Guider can be re-oriented towards other applications that involve structured discovery of symbolic formulas.
- We conduct extensive analysis on post-training unstructured sparsity benchmarks and show that formulas discovered through LLM-Guider outperform considered baselines.
- Unlike genetic algorithms, LLM-Guider enables the incorporation of domain and mathematical knowledge through hints and in-context examples. We also extend the aggregation strategies over the calibration dataset, resulting in never-seen-before novel pruning metrics.

2 Related Work

2.1 Sparsity in LLMs

Deep neural networks are typically dense and over-parameterized, leading to enormous computation and memory costs. Sparsity has emerged as a leading approach to a creation of more efficient models that function within high-dimensional feature spaces, while simultaneously reducing representational complexity by utilizing only a subset of dimensions at a given time (Hoefler et al., 2021). There are two main forms of sparsity: structured and unstructured sparsity. Structured sparsity focuses on removing larger structures, which for LLMs includes layers (Men et al., 2024), attention heads (Venkataramanan et al., 2023), neurons, weight blocks, N:M-structures, embeddings, and hidden dimensions (Liu et al., 2023; Xia et al., 2024; Zhou et al., 2021). On the other hand, unstructured sparsity prunes individual weights without regard to their structural grouping. Unstructured sparsity is usually associated with an importance matrix, which ranks the weights based on certain criteria. To compute concrete importance scores, a small calibration dataset drawn from the training distribution is first passed through the unpruned model; for each weight, we collect local statistics for activations and gradients, aggregate them using aggregation functions over the calibration set, and feed those aggregated features into the pruning formula under test—producing per-weight scores that are then sorted and thresholded to achieve the target sparsity.

Different unstructured sparsity algorithms vary in how they compute this weight importance matrix and which input parameters they depend on. Standard magnitude pruning (Cheng et al., 2024) uses the absolute weight magnitude for pruning decisions, based on the intuition that weights with smaller magnitudes contribute less to the network's output. However, the outcome of neural network is not solely decided by the weight magnitudes. Even when a weight has a small magnitude, it can significantly contribute to the result if amplified by a large activation. To this end, WANDA (Sun et al., 2024) and RIA (Zhang et al., 2024) proposed activation-based unstructured sparsity criterion leveraging the fact that output activations depend on both weight and input values. Additionally, methods like GBLM-Pruner (Das et al., 2024) and Pruner-Zero (Dong et al., 2024) incorporate gradients into the pruning decision, often outperforming activation-based methods. Large gradients indicate that the network is learning and is sensitive to parameter changes. SparseGPT (Frantar and Alistarh, 2023) goes one step further and uses the Hessian matrix approximation using activations, which is used both in pruning and optimal recovery. The key to designing effective unstructured sparsity algorithms lies in defining an appropriate weight importance formula / pruning metric. Research suggests that we can move beyond traditional approaches based solely on heuristics, instead leveraging a mixture of inputs to achieve optimal performance. This shift allows us to redefine the process as a search over possible formulas, enabling a more nuanced and data-driven approach to weight importance.

2.2 Symbolic formula optimization

While there is some clarity regarding what an effective sparsity formula should depend on, the specific symbolic formula remains an area of active research. Most works focus on manually crafting weight importance formulas based on heuristics (Dong et al., 2024). However, these heuristic approaches are prone to oversimplifications and can result in locally sub-optimal solutions. Therefore, it is a common practice to use global model-based approaches that explore the search space efficiently using search algorithms. For example, PrunerZero (Dong et al., 2024) employs genetic algorithms to discover unstructured sparsity formulas, which have outperformed existing heuristic-based methods. Similarly, the successful application of genetic algorithms led to the discovery of the Lion optimizer (Chen et al., 2023b). Reinforcement learning has also been used to discover novel neural network architectures (Zoph and Le, 2017) and activation functions (Ramachandran et al., 2017). With recent breakthroughs in language modelling, LLMs have been readily used for language guided search process. For instance, LLM-guided search found code functions for reinforcement learning rewards in Eureka (Ma et al., 2024). A similar approach was also used to discover novel alignment formulas in (Lu et al., 2024). Search algorithms, in general and LLMs in particular entail tremendous potential for the discovery of novel solutions while avoiding local minima arising from heuristic approaches. Search algorithms, particularly those guided by LLMs, possess reasoning abilities along with mathematical and coding skills, making them valuable assets in discovering novel solutions while avoiding local minima that often challenge heuristic approaches.

2.3 Prompting

LLMs effectively learn the nuances of token distributions, significantly improving their capability to tackle a diverse array of tasks during training (Wei et al., 2022). However, to fully optimize LLM performance, it is necessary to use inference-time optimization techniques, applied after the model has been trained. Inference-time optimization involves both enhancing the prompt itself and employing techniques that solve problems using a multi-step approach. Role prompts (Kong et al., 2024) allow the LLM to adjust its style and focus on specific target tasks. Additionally, K-shot prompting

(Brown et al., 2020) enables models to learn from in-context information and generate output based on examples. Progressive hints in prompts (Zheng et al., 2023) have also been shown to enhance the reasoning abilities of LLMs. A common practice is to decompose problems into a Chain of Thought (CoT) (Wei et al., 2023), which allows the model to solve complex tasks step-by-step. Furthermore, the Reflection approach (Schulhoff et al., 2024) is often used to assess and improve generated responses. To go beyond the sequential nature of CoT, the Tree of Thoughts (ToT) (Yao et al., 2023) introduces a combination of parallel and sequential generation. To optimize LLM usage during inference, it is beneficial to enhance prompts with hints and cues for additional in-context information. Single-turn prompting under-performs compared to multi-step approaches, which effectively decompose prompts and verify each step's outcome. In conclusion, complex tasks require advanced prompting techniques for optimal results.

3 Methodology

LLM-Guider is a generic symbolic formula optimization framework that operates through three distinct stages: k-shot example selection, formula generation, and evaluation. It leverages the capabilities of a large language model (LLM) to efficiently explore the formula space. In the subsequent subsections, we provide detailed descriptions of each component within our framework.

3.1 Example selection

3.1.1 Search Space Design

The search space design serves as the foundation for LLM-guided formula discovery. The search space is composed of weights, activations, and gradients, combined with operations that define the pruning metrics. Weight magnitudes are derived from the target LLM while activations and gradients are aggregated over a small calibration dataset to capture the required input statistics. Table 1 provides an overview of extended list of aggregation strategies over the calibration dataset. LLM-Guider treats these strategies as input variables and autonomously determines which operations to use when generating the symbolic formulas through iterative refinement. Unless constrained by hints, the operations over these input variables are not set explicitly.

The pruning metric defines the importance of weights in a model, determining which are retained or pruned under a predefined sparsity threshold. These formulas are represented as code with a predefined header, ensuring seamless integration and execution during the evaluation process. This coding approach also mitigates format conversion issues, promoting consistency and efficiency.

3.1.2 K-Shot Example Selection

Seed formulas are essential for initializing the pruning metric discovery process. They provide a structured foundation for subsequent LLM-guided exploration by defining initial examples. Seed formulas are provided upfront as code, including classical magnitude pruning and custom formulas that combine weights, activations and gradients. These formulas are evaluated to obtain initial performance scores. By serving as a reliable starting point, seed formulas guide LLMs to generate meaningful pruning metrics and uphold a consistent response format like the role of correct examples in few-shot learning.

K-shot examples are an integral component of the LLM-guided framework. As they are included in the prompt, they enable in-context learning, supporting reasoning and iterative improvement. These examples form the foundation of each generation, helping the model build upon past successes and avoid previous failures. Each generation relies on k-shot examples, as described in (Liu et al., 2021), to enhance performance and draw conclusions based on previously evaluated attempts. Specifically:

- **Generation References:** Each new generation references examples from the pool of past attempts.
- **Evaluation Scores:** Examples are paired with their evaluation scores, providing clear insights into what strategies worked and which failed.
- **Selection Strategies:** The process for selecting k-examples includes:
  - Randomly selecting and expanding past node
  - Selecting best node
  - Using the node from the previous evaluation and iteratively expanding it
  - Selecting the top-n individual generations with the best scores to form a context.

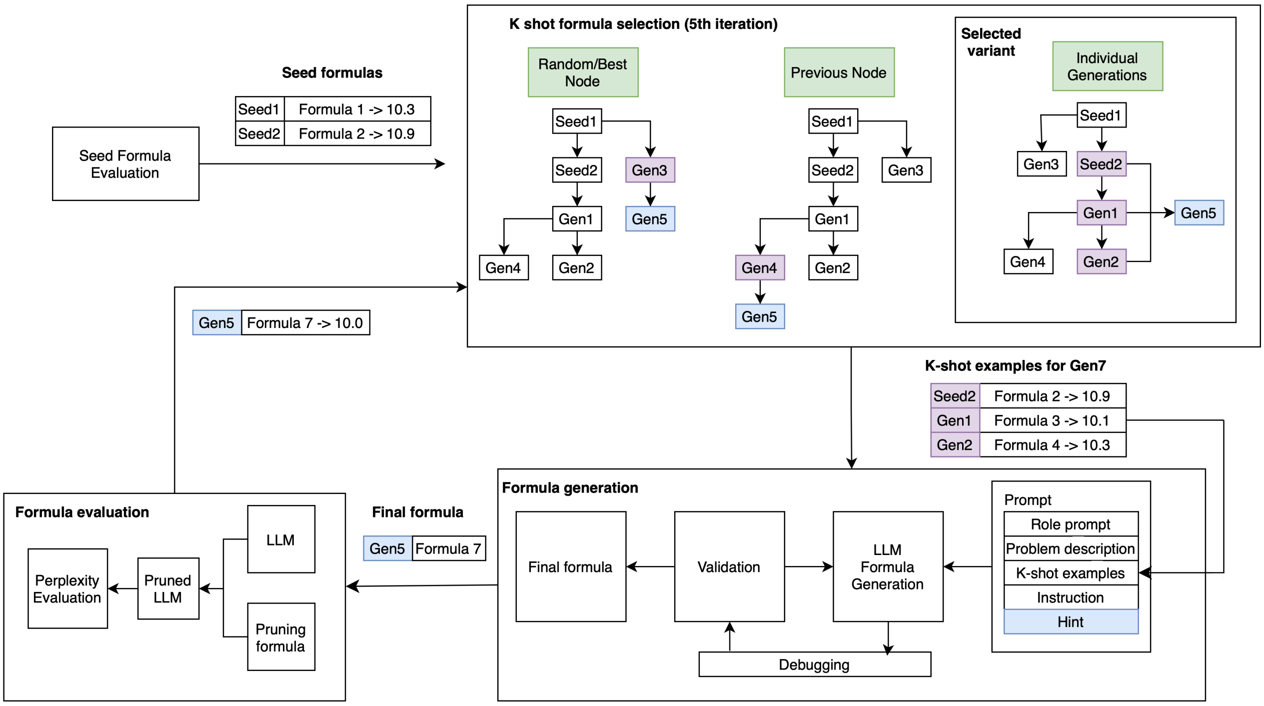

Figure 1: Overview of LLM-Guider, a generic framework for symbolic formula optimization, tailored for discovering optimal pruning criterion for post-training unstructured sparsity in LLMs: In the *Example Selection* stage, we identify the most promising k-shot seed formulas based on a predefined policy. The subsequent *Formula Generation* stage leverages the capabilities of LLMs, geared with customized hints and k-shot examples, to produce novel symbolic formulas tailored to our pruning objectives. In the *Formula Evaluation* stage, these generated formulas are rigorously assessed and the highest-scoring formulas fed back into the generation pool for iterative refinement.

Category	**Names**
Weights	W
Activations	A_{mean}, A_{M^2}, $A_{\text{sum\_squares}}$, $A_{\text{sum\_abs}}$, A_{min}, A_{max}, $A_{\text{mean\_abs}}$, $A_{\text{mean\_squared}}$, A_{variance}, A_{std}
Gradients	G_{mean}, G_{L1}, G_{L2}, G_{M^2}, $G_{\text{sum\_gradients}}$, $G_{\text{sum\_abs\_gradients}}$, $G_{\text{sum\_gradients\_squared}}$, $G_{\text{mean\_gradients}}$, $G_{\text{mean\_abs\_gradients}}$, $G_{\text{mean\_gradients\_squared}}$, G_{variance}, G_{std}

Table 1: An exhaustive list of input variables employed in the search space design of LLM-Guider. More explanation on these can be found in Appendix B.

Initially, no prior generations exist to select as examples. Following existing work (Chen et al., 2024), an initial pool of predefined examples is created and evaluated. In our approach, we employ a single Wanda (Sun et al., 2024) seed formula that simultaneously integrates both weights and activations.

Through carefully managed k-shot example selection, the framework achieves a balance between exploration of new possibilities and refinement of high-performing approaches. This balance ensures efficient and effective formula discovery, leveraging prior knowledge while encouraging innovation.

3.2 LLM-Based Symbolic Formula Generation

Unstructured sparsity operates by applying a mask over weights. To enhance this process, an LLM is employed to generate new weight importance matrices through prompting. The LLM leverages prior knowledge and dynamic reasoning to create novel sparsity formulas.

The prompts used to guide the LLM consist of the following structured parts:

- **Role Prompt:** Defines the task's context to focus the LLM on pruning objectives.
- **Instruction Prompt:** Provides formula objectives, emphasizing interpretability and detailing available variables (e.g., weights, activations, and gradients aggregations). For each variable, the size of its corresponding tensor is explicitly provided, ensuring the LLM can appropriately handle and process the input data.
- **k-Shot Examples:** Supplies selected past

evaluations, enabling reasoning over previously successful attempts.

- **Hints:** Offers domain-specific guidance such as normalization or variable constraints. These are sampled from a predefined pool using strategies like uniform or weighted sampling and are included as textual parts of the prompt.

The LLM combines its internal knowledge with external hints and reasoning over k-shot examples to dynamically generate innovative and effective sparsity formulas. By integrating structured prompts and leveraging both static and dynamic knowledge sources, the LLM serves as a central tool for discovering novel sparsity formulas. This approach ensures adaptability and precision in tailoring pruning strategies to specific tasks.

When symbolic formulas are generated in natural language by the LLM, there is no guarantee that the resulting code will compile or execute correctly. This introduces a need for a robust validation and refinement process to ensure the correctness of the formulas. Validation begins by running the generated formulas in an evaluation environment. If execution fails, a debugging phase is triggered.

The LLM mimics human debugging by verifying tensor sizes step-by-step. It augments the initial code with a detailed walk through of tensor sizes to identify inconsistencies or unsupported operations. Once inconsistencies are detected, the LLM attempts to fix the issues using the gathered size information. This process is repeated for a predefined number of attempts to validate and correct the formulas efficiently.

The iterative validation and refinement process ensures the correctness of LLM-generated formulas. By systematically identifying and resolving errors, this approach guarantees reliable symbolic pruning metrics, even when initially created through natural language.

3.3 Formula evaluation

Having generated a candidate formula, the next step is to evaluate its performance so that LLM-Guider can quantify its effectiveness.

In our framework, we execute evaluation in three sequential steps. We begin by applying the selected symbolic formula to our precomputed statistics on weights, activations, and gradients, which yields a binary mask indicating which parameters to keep and which to remove. Using this mask, we prune the language model by zeroing out the designated weights, producing a leaner version of the network without any additional fine-tuning. Finally, we assess the pruned model's quality by running it on the WikiText-2 test set and recording its perplexity.

Once a formula has been evaluated and its performance recorded, it is added to the k-shot example pool for the next generation. This cycle of formula generation, evaluation, and example selection repeats until the predefined number of iterations has been completed.

4 Discovered formulas

Our framework introduces novel pruning metrics derived through extensive experimentation on SmolLM2 model, described in details Appendix A. Specifically, it discovered two effective pruning formulas:

1. **Best-performing formula:** This metric combines normalization of activations, average gradients, and gradient variability, hypothesizing that parameters with higher gradient variability play critical roles in optimization. It was found using by modifying LLM-Guider baseline configuration with diversity hints

$$I = W \odot \left[\left(\frac{A_{\text{mean}}-A_{\text{min}}}{A_{\text{max}}-A_{\text{min}}+\epsilon}\right)G_{\text{mean}}^{\top}+G_{\text{mean\_abs}}\right] \odot G_{\text{std}}$$

2. **Second-best formula:** This formulation emphasizes gradient variability weighted by parameter magnitudes, capturing critical gradient variations essential for robust generalization across models.

$$I = |W| \odot G_{\text{std}} \quad (1)$$

Traditional pruning methods typically rely on mean or magnitude-based norms of activations and gradients, potentially overlooking parameters exhibiting small but significant variability. In contrast, our proposed metrics explicitly incorporate statistical aggregates such as gradient standard deviation and activation variability, capturing parameter importance more effectively. However, it is important to note that metrics specifically tailored to individual models may risk overfitting, thereby diminishing their generalization capabilities.

Interestingly, the best-performing formula can be interpreted as a generalized version of the second-best performing one, as it adds an additional factor to the multiplication. This observation suggests that, while the LLM was capable of discovering a formula with a simple structure, it also demonstrated the ability to refine and extend it into a more complex and effective form.

5 Experiments

5.1 Evaluation Setup

We evaluate the effectiveness of LLM-guided search using a single model family. Specifically, we employ SmolLM2-135M (Allal et al., 2025), a lightweight language model designed for computationally efficient experimentation. We use GPT-4o mini for formula generation. Model performance is assessed across two primary benchmarks. For language modeling, we report perplexity on the WikiText2 test set (Merity et al., 2016). For zero-shot generalization, we evaluate using EleutherAI's LM Harness framework, which includes a diverse set of tasks: ARC Challenge (Clark et al., 2018), ARC Easy (Clark et al., 2018), BoolQ (Clark et al., 2019), OpenBookQA (Mihaylov et al., 2018), RTE (Wang et al., 2019), Winogrande (Sakaguchi et al., 2019), and HellaSwag (Zellers et al., 2019).

LLM-Guider is designed to search for optimal symbolic formulas, with 100 generations evaluated per run. Our empirical studies, using greedy search, led to the framework configuration in Appendix D. The process takes approximately 1.5 hours on a single A100 GPU.

To compute pruning metrics at each iteration, we rely on weights, activations, and gradients. Multiple statistical measures for these components are precomputed, as detailed in the Appendix B. We used a fixed set of 128 calibration samples to precompute statistics, following the Wanda approach (Sun et al., 2024), which ensures that the unstructured sparsity stabilizes at optimal levels. During each iteration, an importance matrix is computed on the basis of these statistics and pruning is applied based on sparsity ratio of 0.5. Similarly to previous work, we use the first fragment of the C4 dataset (Raffel et al., 2023) for evaluation.

5.2 Baselines

We evaluated the performance of our approach against several established pruning methods, each taking advantage of different combinations of weights, activations, and gradients to compute pruning metrics. Specifically, we compare with standard magnitude pruning, Wanda (Sun et al., 2024), which incorporates both weights and activations, PrunerZero (Dong et al., 2024), a state-of-the-art method based on weights and gradients, and SparseGPT (Frantar and Alistarh, 2023), which uses Hessian information and error propagation to update weights. We also include the dense (unpruned) model corresponding to a sparsity ratio of 0 as an additional baseline. These diverse baselines serve as strong reference points, allowing us to assess the effectiveness of our LLM-guided search in achieving greater sparsity while preserving competitive model performance.

5.3 Language Modeling

Language modeling using perplexity allows for assessing how well a language model predicts a given sequence of text. Lower perplexity indicates that the model assigns higher probabilities to the correct words, meaning that it has a better understanding of the language and generates a more fluent, coherent text.

Based on the results in Table 2, LLM-Guider outperforms all other methods that do not require weight updates. Furthermore, it significantly narrows the performance gap with SparseGPT, the only algorithm that utilizes weight updates. This trend aligns with findings from the Pruner-Zero experiment conducted on OPT models, which showed that SparseGPT remains competitive with Pruner-Zero due to its ability to update weights. Notably, the same experiment also demonstrated that the performance difference between these methods becomes more pronounced in models with smaller parameter counts.

Method	Weight Update	Perplexity ↓
LLM-Guider	✗	**30.99**
Magnitude	✗	536.44
Wanda	✗	31.66
Pruner-Zero	✗	33.09
SparseGPT	✓	**30.83**

Table 2: Comparison to state-of-the-art methods for SmolLM2-135M using WikiText-2 perplexity with sparsity ratio 0.5

Method	arc_challenge	arc_easy	boolq	hellaswag	openbookqa	rte	winogrande	Mean ↑
Dense	26.11 ± 1.28	54.12 ± 1.02	42.91 ± 0.87	34.94 ± 0.48	22.00 ± 1.85	51.26 ± 3.01	51.62 ± 1.40	40.42 ± 1.42
LLM Guider (Best)	20.73 ± 1.18	44.07 ± 1.02	**62.42** ± 0.85	29.82 ± 0.46	16.80 ± 1.67	**57.04** ± 2.98	**51.22** ± 1.40	**40.30** ± 1.37
LLM Guider (Second)	**21.08** ± 1.19	43.90 ± 1.02	62.35 ± 0.85	29.82 ± 0.46	16.40 ± 1.66	56.32 ± 2.99	**51.22** ± 1.40	40.16 ± 1.37
PrunerZero	19.03 ± 1.15	**44.19** ± 1.02	56.73 ± 0.87	29.81 ± 0.46	15.60 ± 1.62	54.51 ± 3.00	49.88 ± 1.41	38.54 ± 1.36
Wanda	20.65 ± 1.18	43.10 ± 1.02	60.92 ± 0.85	29.94 ± 0.46	15.20 ± 1.61	51.26 ± 3.01	51.14 ± 1.40	38.89 ± 1.36
SparseGPT	20.82 ± 1.19	41.67 ± 1.01	57.89 ± 0.86	**30.56** ± 0.46	**17.00** ± 1.68	52.71 ± 3.01	50.83 ± 1.41	38.78 ± 1.37
Magnitude	19.37 ± 1.15	35.56 ± 0.98	38.01 ± 0.85	26.78 ± 0.44	13.20 ± 1.52	54.51 ± 3.00	50.59 ± 1.41	34.00 ± 1.34

Table 3: Accuracies (%) of SmoLM2-135M for 7 zero-shot tasks with unstructured 50% sparsity.

5.4 Zero-shot evaluation

We conducted extensive experiments to evaluate our model across a comprehensive suite of zero-shot commonsense reasoning tasks. As detailed in Table 3, evaluation performance varied considerably across tasks. Notably, on benchmarks such as BoolQ, RTE, and WinoGrande, our method demonstrated a clear advantage over baseline approaches. With an overall mean accuracy of 40.30%, our approach significantly surpasses the Wanda baseline (38.89%) and compares favorably with the Dense model (40.42%). These findings underscore that pruning based on a calibration dataset yields robust improvements in downstream performance.

6 Conclusions

In this work, we introduced LLM-Guider, a language-guided symbolic formula optimization framework designed to discover novel pruning metrics for post-training unstructured sparsity in large language models. Our approach leverages the advanced reasoning and coding capabilities of modern LLMs by combining domain-specific hints, k-shot examples, and iterative refinement to generate and validate effective symbolic formulas.

Through extensive experiments on SmolLM2-135M, we demonstrated that the formulas discovered by LLM-Guider outperform traditional methods such as magnitude pruning, Wanda, and PrunerZero—achieving lower perplexity and competitive zero-shot performance without requiring weight updates. Detailed ablation studies further highlighted the impact of key components such as seed formula selection, generation strategy, and tailored hint configurations, confirming that even minimal human guidance can significantly enhance the discovery process.

Our best-performing formula, which integrates normalized activation statistics, average gradients, and gradient variability, underscores the benefit of incorporating richer statistical aggregates beyond standard mean-based approaches. Overall, LLM-Guider not only advances the state-of-the-art in unstructured sparsity but also establishes a transparent and systematic methodology for symbolic optimization in neural networks.

Limitations

With extensive experiments and analysis, we show that LLM-Guider framework can successfully find a state-of-the-art solution in the SmolLM2-135M model at 50% sparsity. This highlights the potential of language guided search for automating sparsity-aware optimization. Applying our method to different model sizes or pruning levels may require re-running the framework to find an optimal solution tailored to each specific configuration. Future work could explore what factors enable solutions to transfer across models and sparsity levels, helping reduce the need for full recalibration.

References

Abhinav Pandey Ab-hishek Kadian Ahmad Al-Dahle Aiesha Letman Akhil Mathur Alan Schelten Amy Yang Angela Fan et al. Abhimanyu Dubey, Abhinav Jauhri. 2024. The llama 3 herd of models.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. Smollm2: When smol goes big – data-centric training of a small language model.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Tianyi Chen, Luming Liang, DING Tianyu, Zhihui Zhu, and Ilya Zharkov. 2023a. Otov2: Automatic, generic, user-friendly. In *International Conference on Learning Representations*.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. 2023b. Symbolic discovery of optimization algorithms.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. 2024. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. 2024. Beyond size: How gradients shape pruning decisions in large language models.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. 2024. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot.

Duc NM Hoang, Shiwei Liu, Radu Marculescu, and Zhangyang Wang. 2023. Revisiting pruning at initialization through the lens of ramanujan graph. In *The Eleventh International Conference on Learning Representations*.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. 2024. Better zero-shot reasoning with role-play prompting.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient llms at inference time.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J. Chan, Jakob Foerster, Mihaela van der Schaar, and Robert Tjarko Lange. 2024. Discovering preference optimization algorithms with and for large language models.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions.

Kai Ruan, Ze-Feng Gao, Yike Guo, Hao Sun, Ji-Rong Wen, and Yang Liu. 2024. Discovering symbolic expressions with parallelized tree search. *arXiv preprint arXiv:2407.04405*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389.

Shashanka Venkataramanan, Amir Ghodrati, Yuki M. Asano, Fatih Porikli, and Amirhossein Habibian. 2023. Skip-attention: Improving vision transformers by paying less attention.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence?

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Cannistraci. 2024. Plug-and-play: An efficient post-training pruning method for large language models.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2021. Learning n:m fine-grained structured sparse neural networks from scratch.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning.

A Ablations

LLM Guider framework consists of multiple variable parts including seed formula, k-shot generation strategy, role prompt and hints. In this section we run multiple ablations on baseline to find the most impactful decisions. Each of ablation is run of 100 generations, using 128 calibration samples with sparsity ratio 0.5 and SmolLM2-135M-Instruct (Allal et al., 2025) as model. Each generation denotes one complete selection–generation–evaluation loop. We repeat experiment over 3 seeds.

First, we begin by investigating the effect of different seed formulas. We consider rules that utilize weights, activations, and gradients. Empirically, we have found that the activation-based metric, Wanda, outperforms other options. The most unstable runs occur when using the magnitude formula alone. Our results from Table 4 indicate that formulas combining at least two of the three components—weights, activations, and gradients—significantly outperform using magnitude alone. Additionally, incorporating PrunerZero into the magnitude formula leads to notable improvements in evaluation metrics. Among all the metrics tested, Wanda is mathematically the most complex, as it involves matrix and row multiplications, whereas other approaches rely on simpler element-wise operations. We hypothesize that this added complexity enables the framework to generate more effective formulas.

Seed formulas	Best Perplexity	Mean Perplexity
Wanda (baseline)	**31.02**	**31.74 ± 1.01**
Magnitude + PrunerZero	**31.02**	32.69 ± 2.89
PrunerZero	31.30	35.10 ± 6.59
Magnitude + Wanda + PrunerZero	31.03	39.58 ± 7.49
Magnitude	31.30	82.67 ± 85.82

Table 4: Comparison of seed formulas for LLM-Guider init on SmolLM2-135M (WikiText-2, sparsity 0.5)

We analyze the impact of different generation strategies based on Table 5. Specifically, we evaluate best node selection, random node selection, selection of top individual generations, and a uniform strategy that combines best, random, and previous nodes. An effective generation strategy should allow the LLM to explore both high-quality and diverse solutions. Our results indicate that, under the tested conditions, diversity plays a crucial role. Notably, random node generation slightly outperforms the best node selection strategy, highlighting the importance of exploration in addition to exploitation.

Generation Strategy	best_metric	mean_metric
Best Node (baseline)	31.02	31.74 ± 1.01
Random Node	31.02	**31.54** ± 0.90
Top 8 Individual Generations	31.02	35.73 ± 8.17
Uniform (Best, Random, Previous Node)	31.50	33.71 ± 2.25

Table 5: Comparison of generation strategies

Role prompting is the technique to tune style of the reponse. In our experiments from Table 6 we have found that using role prompt did not improve generation results. We hyphotesize that style of response is not that important for task of unstructured sparsity formula generation.

Use role prompt	Best Perplexity	Mean Perplexity
✓ (baseline)	**31.02**	31.74 ± 1.01
✗	**31.02**	**31.44** ± 0.74

Table 6: Ablation on role prompt

Hints are part of the framework that allows for the most flexibility and enhancements. As observed in Table 7, in our experimental regime, the strongest results were achieved by limiting the number of variables that the LLM uses. When it comes to top-generation diversity, only hints allowed improvements over the best metrics; however, they were unstable in repetitive runs. This indicates that the usage of limiting hints was critical for stability, whereas adding hints that promote variety was beneficial for improving the best metrics. Notably, the selection of hints focusing on complexity or employing complex operations, such as matrix multiplication alone, performed the poorest.

Hint Configuration	Best Perplexity	Mean Perplexity
Baseline	31.02	31.74 ± 1.01
Diversity Hints	**30.99**	36.54 ± 5.46
Limiting Hints	31.02	**31.21** ± 0.16
Domain-specific	32.04	32.54 ± 0.48
Inspiration Hints	34.31	35.31 ± 1.20
Reflection Hints	32.70	36.60 ± 6.25
Diversity and Limiting	31.30	34.31 ± 2.76
Reflection and Limiting	31.30	35.75 ± 7.71
Reflection and Domain-specific	31.02	36.19 ± 5.48
Complexity Hints	31.30	168.14 ± 234.32
Matrix Multiplication Hints	36.93	288.08 ± 217.70
Matrix Operations Correctness Hints	41.93	203.53 ± 273.89

Table 7: Ablation comparing effectiveness of different hint configurations

We studied the impact of retries during the debugging phase. This experiment demonstrates the effectiveness of debugging in improving formula generation. Our results, as shown in Table 8, indicate that including debugging retries enhances formula generation. This is a natural conclusion, as it leads to a higher number of correctly generated formulas. However, in our study, the impact was particularly visible when up to five retries were performed for each generation.

Number of Retries	Best Perplexity	Mean Perplexity
0	31.02	31.38 ± 0.41
2 (baseline)	31.02	31.74 ± 1.01
5	31.02	**31.21** ± 0.16

Table 8: Performance comparison on number of retries during debugging

B Inputs

Below, we provide an extensive list of variables used in LLM-Guider. These variables collectively define the search space for the LLM to find an optimal pruning metric that effectively sparsifies the target LLM with minimal loss in performance.

Weights W: Weights of the model.

Activations

- A_{mean}: Mean of the activations across batches.
- A_{M^2}: Accumulated sum of squared differences from the mean (used for calculating variance).
- $A_{\text{sum\_squares}}$: Sum of squares of the activations along the batch dimension.
- $A_{\text{sum\_abs}}$: Sum of absolute values of the activations along the batch dimension.
- A_{min}: Minimum value of the activations across batches.
- A_{max}: Maximum value of the activations across batches.

- $A_{\text{mean\_abs}}$: Mean of the absolute values of the activations along the batch dimension.
- $A_{\text{mean\_squared}}$: Mean of the squared values of the activations along the batch dimension.
- A_{variance}: Variance of the activations, computed from the accumulated sum of squared differences M^2.
- A_{std}: Standard deviation of the activations, computed as the square root of the variance.

Gradients

- G_{mean}: Mean of the gradients across batches.
- G_{L1}: L1 norm of the gradients across batches.
- G_{L2}: L2 norm of the gradients across batches.
- G_{M^2}: Accumulated sum of squared differences from the mean (used for calculating variance).
- $G_{\text{sum\_gradients}}$: Sum of gradients along the batch dimension.
- $G_{\text{sum\_abs\_gradients}}$: Sum of absolute values of the gradients along the batch dimension.
- $G_{\text{sum\_gradients\_squared}}$: Sum of squares of the gradients along the batch dimension.
- $G_{\text{mean\_gradients}}$: Mean of the gradients along the batch dimension.
- $G_{\text{mean\_abs\_gradients}}$: Mean of the absolute values of the gradients along the batch dimension.
- $G_{\text{mean\_gradients\_squared}}$: Mean of the squared gradients along the batch dimension.
- G_{variance}: Variance of the gradients, computed from the accumulated sum of squared differences M^2.
- G_{std}: Standard deviation of the gradients, computed as the square root of the variance.

C Baseline Pruning Metrics

1. Magnitude Pruning

The pruning score for each weight is

$$S_{ij} = \left|W_{ij}\right|$$

where

- W_{ij} is the weight of the connection from neuron j (input) to neuron i (output).

2. SparseGPT

SparseGPT approximates the influence of each weight via the inverse Hessian:

$$S_{ij} \;=\; \frac{W_{ij}^2}{\left[(H^{-1})\right]_{ij}}$$

where

- H is the (approximate) Hessian matrix of the loss w.r.t. the weights.
- (H^{-1}) denotes the diagonal of the inverse Hessian.

3. Wanda

Wanda scores combine weight magnitude with activation norm:

$$S_{ij} \;=\; \left|W_{ij}\right| \;\times\; \|X_j\|_2$$

where

- $X_j \in \mathbb{R}^N$ is the vector of activations at neuron j over a calibration dataset.
- $\|\cdot\|_2$ denotes the Euclidean norm.

4. PrunerZero

PrunerZero combines squared magnitude with scaled gradient magnitude:

$$S_{ij} = \; W_{ij}^2 \;\times\; \sigma\big(|G_{ij}|\big), \qquad \sigma(x) = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where

- $G_{ij} = \frac{\partial \mathcal{L}}{\partial W_{ij}}$ is the gradient of the loss $\mathcal{L}$ w.r.t. W_{ij}.
- $\sigma(\cdot)$ denotes min–max normalization applied across all absolute gradient values.

D LLM-Guider Baseline

Configuration Details	
Number of Rounds	100
Number of Retries	2
Eureka Seed	0
Temperature	1
Use Role Prompt	true
K Examples	wanda
Sparsity Ratio	0.5
Number of Samples	128
Model	HuggingFaceTB/SmolLM2-135M-Instruct
Evaluator Seed	0
Hint Sampler Type	UniformSampler
Generation Strategy Sampler Type	UniformGenerationStrategySampler
Generation Strategy Value Type	BestNodeStrategy
Hint Options	LimitVariablesHint (value: 2)
	LimitVariablesHint (value: 3)
	LimitVariablesHint (value: 4)
	ComplementMatchingSizeHint
	UnaryOperationsHint
	TryDifferentHint
	AlternativePerspectiveHint

Table 9: Configuration Table

E Hints

Hints Overview: This document provides a summary of the various hint types used to guide a problem-solving process. The hints are organized into several categories that serve distinct purposes: generating candidate solutions via ensemble reasoning, inspiring creative approaches, reflecting on previous attempts, adjusting the complexity of approaches, imposing problem constraints, and addressing domain-specific challenges.

Dynamic Hints

These hints use an internal LLM to dynamically generate multiple candidate solutions and refine them through debate and synthesis.

- **CandidateSelectionHint:** Generates several candidate solutions for a given problem and then uses a two-step process (first, detailed reasoning for each candidate; second, synthesis of the best solution) to present the top candidate. This hint leverages step-by-step reasoning to help decide among multiple possible approaches.

- **DebateHint:** Uses a debate format where opinions are generated from multiple historical figures (or personas) about a problem. It then synthesizes these divergent views into a concise, best possible solution. This hint is ideal when diverse perspectives might reveal hidden insights into the solution.

Inspiration Hints

These hints are designed to spark creativity by encouraging the solver to leverage domain-specific expertise or past successful strategies, including a prompt for getting inspired by prior approaches.

- **AlgebraHint:** Invokes algebraic techniques and principles, helping the solver to explore a variety of functions and relationships.

- **GameTheoryHint:** Draws on strategic decision-making principles from game theory, offering insights into competitive or adversarial problem settings.

- **RLRewardFunctionsHint:** Utilizes ideas from reinforcement learning, specifically around optimizing reward functions, to enhance solution approaches.

Reflection Hints

These hints encourage self-assessment and iterative improvement by prompting the solver to reflect on both successes and mistakes from prior attempts.

- **ReflectAndAvoidErrorsHint:** Advises reflecting on previous mistakes and learning from them to prevent similar errors in future attempts.

- **IdentifySuccessesHint:** Encourages the solver to pinpoint what worked well in earlier attempts and to replicate those successful strategies.

- **CombineIdeasHint:** Suggests merging two or more ideas to create a novel approach that benefits from multiple insights.

- **SeekDeeperInsightsHint:** Prompts the solver to look beyond the obvious and uncover hidden connections or deeper insights in the problem.

Complexity Hints

These hints help modulate the difficulty of the approach, suggesting strategies to simplify or to challenge the solver with more rigorous methods.

- **TryEasyHint:** Suggests trying a simpler or more straightforward approach.

- **TryEasierHint:** Recommends opting for an even simpler variant than before, reducing complexity further.
- **TryHardHint:** Encourages the solver to explore a challenging strategy that might lead to more robust solutions.
- **TryHarderHint:** Urges the solver to ramp up the challenge, trying an approach more difficult than previous attempts.

Diversity Hints

These descriptions are designed to provide clear guidance on how each hint supports diverse thinking and problem-solving techniques.

- **TryDifferentHint:** Advises experimenting with a markedly different strategy compared to those used before, potentially uncovering a new pathway.
- **AlternativePerspectiveHint:** Invites the solver to rethink the problem from a different angle, potentially revealing non-obvious solutions.

Limiting Hints

These hints impose specific constraints to ensure the solution remains within manageable or expected bounds.

- **LimitVariablesHint:** Directs the solver to restrict the formula to exactly a given number of variables, ensuring simplicity or focus in the formulation.

Sparsity Domain Specific Hints

Aimed primarily at problems involving matrix operations or when matching output dimensions is critical, these hints are tailored specifically to the task at hand.

- **ComplementMatchingSizeHint:** Advises a step-by-step approach: develop a novel formula, evaluate its size against an expected matrix size, and only proceed if sizes match—otherwise, adjust operations accordingly.
- **MatrixMultiplicationHint:** Recommends using matrix multiplication by listing potential components with their respective output shapes, ensuring that the final result meets the expected dimensions.
- **NormalizationHint:** Suggests incorporating normalization techniques (e.g., Min-Max Scaling, Z-Score, L2 Norm, L1 Norm) to refine the solution.
- **ResultDimensionHint:** Ensures that the final formula outputs a matrix or result with the precise dimensions required by the problem.
- **UnaryOperationsHint:** Proposes using one or more unary operations (such as squaring, negation, absolute value, logarithm, exponential, etc.) to adjust the result, emphasizing the importance of adapting operations to meet the problem's dimensional needs.

F Licenses

The datasets and tools used in this research are licensed as follows: WikiText is licensed under the Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0) License, allowing free use, modification, and distribution, with the requirement for attribution and the condition that derivatives must be shared under the same license. SmolLM2 is licensed under the Apache License 2.0, permitting free use, modification, and distribution, including for commercial purposes, provided that attribution is given, a notice of changes is included, and there is no warranty. C4 is licensed under the Open Data Commons Attribution License (ODC-BY), allowing for free use, modification, and distribution, with the condition that attribution is provided and compliance with the terms of the original Common Crawl dataset is ensured. Additionally, EleutherAI's LM Evaluation Harness is open-source software released under the MIT License, which permits free use, modification, and distribution, including for commercial purposes, with attribution required and no warranty.