Is In-Context Learning Feasible for HPC Performance Autotuning?

Thomas Randall
Clemson University
Clemson, South Carolina
tlranda@clemson.edu

Akhilesh Bondapalli
Clemson University
Clemson, South Carolina
abondap@clemson.edu

Rong Ge
Clemson University
Clemson, South Carolina
rge@clemson.edu

Prasanna Balaprakash
Oak Ridge National Laboratory
Oak Ridge, Tennessee
pbalapra@ornl.gov

***Abstract*—We examine whether in-context learning with Large Language Models (LLMs) can effectively address the challenges of High-Performance Computing (HPC) autotuning. LLMs have demonstrated remarkable natural language processing and artificial intelligence (AI) capabilities, sparking interest in their application across various domains, including HPC. Performance autotuning – the process of automatically optimizing system configurations to maximize efficiency through empirical evaluation – offers significant promise for enhancing application performance on larger systems and emerging architectures. However, this process remains computationally expensive due to the combinatorial explosion of configuration parameters and the complex, nonlinear relationships between configurations and performance outcomes.**

We pose a critical question: Can LLMs, without task-specific fine-tuning, accurately infer performance-configuration patterns by combining in-context examples with latent knowledge? To explore this, we leverage empirical performance data from real-world HPC systems, designing structured prompts and queries to evaluate LLMs' capabilities. Our experiments reveal inherent limitations in applying in-context learning to performance autotuning, particularly for tasks requiring precise mathematical reasoning and analysis of complex multivariate dependencies. We provide empirical evidence of these shortcomings and discuss potential research directions to overcome these challenges.

***Index Terms*—LLM, In Context Learning, HPC, Performance Autotuning**

I. Introduction

Large Language Models (LLMs) have transformed artificial intelligence, extending beyond their origins in natural language processing (NLP) to drive innovation across diverse fields. Trained on massive text corpora, LLMs leverage self-supervised learning and transformer-based attention to achieve human-like text generation, pattern recognition, and complex reasoning. Beyond their general capabilities, these models can be fine-tuned with domain-specific data to deliver tailored solutions in diverse domains including healthcare, finance, engineering, and scientific computing. A key strength of LLMs is their ability to perform few-shot or zero-shot learning, enabling them to adapt to novel tasks with minimal in-context labeled examples and without fine-tuning [1]. This adaptability makes them a potential tool for tackling challenges in HPC performance optimization, where exploring vast parameter spaces is essential.

HPC performance optimization is a necessity due to the ever-evolving system architectures. As HPC systems grow in size and heterogeneity to support grand challenges and business-critical applications, achieving maximum performance requires tuning both new and existing applications. Given the substantial number of potential optimizations, including loop unrolling, blocking, concurrency, and topology, manually exploring all possible configurations is impractical. Autotuning provides a systematic approach to optimizing performance by evaluating a small subset of configurations on the target platform. Using intelligent search algorithms, it efficiently navigates the vast parameter space, avoiding the impracticality of exhaustive searches. Notable approaches include Bayesian optimization, GPTune, YTOPT, and Bliss [2]–[4]. By leveraging autotuning, HPC applications can adapt to emerging architectures and scale effectively, ensuring high performance and power efficiency.

Despite significantly reducing the number of evaluations, autotuning typically requires tens to hundreds of empirical evaluations. Each evaluation involves generating and executing an executable with the specified parameter configuration. Even simple kernels may take hours to tune, while more complex applications with larger search spaces can require days. To further reduce costs, transfer learning methods leverage data from related autotuning tasks (e.g., similar input sizes or kernels). However, they still require dozens or more evaluations [5].

This paper explores a critical question: Is in-context learning with LLMs, without fine-tuning, feasible for HPC performance autotuning? In-context learning has demonstrated success across a wide range of tasks [6]–[9], particularly in natural language processing applications such as translation, summarization, arithmetic problem-solving, and code generation and programming assistance. Recently, in-context learning for mathematical functions has gained attention. Studies indicate that functions such as linear, polynomial, and more complex mathematical relationships can be learned through in-context learning [10]–[13]. However, most of this research focuses on training models from scratch using function-specific data rather than leveraging pre-trained LLMs. Moreover, studies on complex multivariate patterns typical in HPC application performance remain limited.

To explore possible answers to this question, we conduct an initial study using the open-source LLaMA model and empirical performance tuples consisting of multi-dimensional configurations and execution times collected from HPC workloads on real systems. To facilitate the use of LLMs, we present the performance data in a natural language format. We design various structured in-context prompts and queries, gradually increasing the number of in-context labeled examples.

Our experiments lead us to conclude that trivial in-context learning with LLMs is ill-suited for performance analysis of HPC applications and systems. Our in-depth analysis suggests that the model's output tends to parrot traits taken from the prompt without insight into what traits should be prioritized. It fails to generalize underlying patterns and cannot infer meaningful insights.

Our main contributions include:

- We conduct an initial study on the feasibility of using in-context learning with LLMs – without task-specific fine-tuning – to generalize performance-configuration patterns in HPC applications and systems.
- We design structured prompts and queries, leveraging actual HPC performance data to evaluate in-context learning. We also analyze the results using LLM internal logits.
- We find that trivial in-context learning fails for HPC performance autotuning, as it falls outside the spectrum of successful few-shot learning.
- We discuss potential approaches to leveraging in-context learning and LLMs for HPC performance autotuning.

Our source code and related artifacts for experiments are publicly available at https://github.com/tlranda/LM-Peel/.

II. Related Works

A. Evaluation and Diversification of LLM Capabilities

The study of Natural Language Processing (NLP) includes diverse language-oriented tasks and evaluations, with many new additions designed specifically for LLMs [14]. LLMs are designed to generate text, with most applications involving some degree of translation and summarization. Summarization is primarily evaluated by automatic techniques such as ROUGE [15], where directly recalling subsequences of words in the reference text is rewarded. This means that models are trained to repeat sub-sequences of reference text, which is often a desirable trait to represent in natural language communications. Conversational agents repeat and rephrase questions, key points, and prior context to indicate a sense of "understanding" which frequently correlates with higher scores from human evaluators and standard evaluation datasets such as those used in OpenAI's evaluations of GPT-2 [16].

One of the first breakthrough examples of using LLMs for numeric prediction came from OpenAI's GPT-3 white paper [1]. This evaluation focused on two- to five-digit addition and subtraction, as well as two-digit multiplication and single-digit arithmetic with three operations. For each of the above, the authors randomly generate 2,000 evaluation problems where all inputs are nonnegative integers, and the models are evaluated in zero-shot, one-shot, and few-shot settings. The authors repeat these experiments with various sizes of models, ranging from 100 million to 175 billion parameters, finding that models in the multi-billion parameter ranges begin to have nontrivial accuracy in their evaluations, and the largest models begin to approach 100% accuracy on the simplest problems.

The authors attempt to check the training data for regular expressions that directly match two possible representations of three-digit arithmetic that they evaluate. They found 17 exact matches for addition and two for subtraction that could have been memorized during training. These results lead to a conclusion that the $< 1\%$ chance of directly memorizing answers and the "human-like" mistakes of forgetting to carry a one may indicate that very large models develop some level of arithmetic capability at scale to generalize their next-token-prediction better rather than rote-memorize arithmetic examples from the training data.

Since this point, many additional techniques have built upon this foundational hypothesis. Vector Databases and Retrieval Augmented Generation leverage the recency bias of LLMs to augment available knowledge bases with in-context learning (ICL) rather than fine-tuning or retraining [17]–[19]. Other efforts attempt to improve LLM capabilities by training new foundational models for particular tasks, such as ChipNEMO [20] and TabPFN [21].

B. HPC Performance Analysis and Autotuning

The HPC community frequently utilizes performance predictions to expand the scope of optimization efforts and maximize potential gains. These predictions must be accurate despite the diversity of hardware configurations and complexity of programs, especially in the absence of extensive data and custom-tuned models. There are myriad efforts within HPC to approach the problem of performance modeling and optimization. To meet the broadest optimization needs in HPC, autotuning techniques [2], [3] treat hardware configurations, program source code and compiler interfaces as tunable black boxes. These techniques learn the relationship between tunable components and the performance objective through surrogate modeling based on a limited number of empirical observations.

a) LLAMBO as an LLM Autotuner: LLM-Assisted Bayesian Optimization (LLAMBO) [22] demonstrated a prompting methodology to permit LLMs to perform autotuning in several practical manners:

- **Discriminative surrogate model**: Presents several configurations and their corresponding performance, then predict the performance of an unknown configuration.
- **Generative surrogate model**: Performs the same task as the discriminative model but uses N-ary classification labels instead of regression.
- **Candidate sampling**: Inverts the discriminative relationship by proposing a configuration expected to produce a given performance value. This is a novel means of search relative to other techniques in the field.

LLAMBO is evaluated on common machine learning datasets from Scikit-Learn rather than HPC tuning problems. However, it lays a foundation that can be broadly applied to HPC autotuning and many other domains. The discriminative surrogate approach has also been applied to Neural Architecture Search [23], Bayesian statistical modeling [24], and even cryptocurrency stock price predictions [25].

The underpinning hypothesis for why LLMs can succeed at these diverse tasks remains similar across these works, namely sourcing a belief that the vast general pre-training permits these models to capture and meaningfully internalize some concept of "world knowledge" relevant to the problem. This world knowledge is connected to the provided context, whereupon the LLM can produce surprisingly effective results despite the unusual setting relative to its trained use case.

III. Experimental Setting and Approach

Our experiments use LLMs to predict runtimes for a fixed program and hardware platform with configurable optimizations at the source-code level that offer different performance tradeoffs. This is a common application tuning problem in HPC. However the complexity of HPC systems and programs often necessitates efforts from teams of domain scientists and system experts to achieve meaningful improvements. Even with the aid of optimization tools and profiling techniques, these efforts do not scale across a large number of applications. Given the capabilities demonstrated by LLMs and the vast amount of text available on hardware, systems, and program optimization, it is reasonable to expect that these models could perform certain amounts of optimization.

Our experiments use the Meta-Llama 3.1 8B instruction-tuned model [26]. We run the model locally to maintain complete control over its operations and facilitate analyses requiring direct access to model logits from generation. We focus on predicting program runtime performance in a discriminative surrogate manner.

A. HPC Application and Performance Dataset

Our experiments consider simple source-code level modifications to improve the performance of a compute-bounded loop nest from the Polybench/C syr2k kernel [27] via loop optimizations. The configuration space includes loop tile sizes, an optional loop interchange and two independent and optional packing operations to prefetch portions of the input arrays accessed by the loop. The pseudocode of the loop nest is presented in Algorithm 1.

The best configurations of this kernel vary based on hardware and the scale of the array operands specified by M and N. We utilize previously collected data [5] that provides performance measurements for all 10,648 unique configurations at two different sizes for M and N on a fixed hardware platform. The empirical data was collected on a Linux machine with 320GB 2x AMD EPYC 7742 64-core processor (128 total core), 1 TB DDR4 running Ubuntu 20.04.2 LTS. The source code configurations are based on Polly [28] LLVM loop optimizations for Clang compiler version 13.0.0. The Polly documentation and discussion could be present in LLM pretraining data, but we expect the model to rely on general optimization knowledge rather than specific knowledge of this framework. The performance data we utilize was not available

Algorithm 1 Pseudocode for the loop-nest to be optimized from Polbench/C's syr2k application

```
Require: Arrays $A[N, M], B[N, M], C[N, N]$
Require: Tile sizes $ii, jj, kk$
  (Optional: Pack array $A$)
  (Optional: Pack array $B$)
  (Optional: Interchange order of $i$ and $j$ loops)
  for $i = 0$ to $N$ in tiles of size $ii$ do
    for $j = 0$ to $M$ in tiles of size $jj$ do
      for $k = 0$ to $i$ in tiles of size $kk$ do
        $C[i, k] \leftarrow A[k, j] * \alpha * B[i, j] + B[k, j] * \alpha * A[i, j]$
      end for
    end for
  end for
```

during pretraining and cannot be memorized due to model's training cutoff date of December 2023.

B. Design of User Prompts and In-context Examples

Following the LLAMBO [22] prompting methodology for discriminative surrogate modeling, we provide the LLM with one or more in-context examples of configuration-runtime relationships and task it with predicting a runtime as a decimal digit sequence for a previously unseen configuration. The prompt consists of three parts: system instructions, problem description, and user ICL and query; an example prompt is shown with highlights for each component in Figure 1. The system prompt instructs the model on how to interact with the user. The problem description provides relevant context about the code and expected runtime conditions. The user ICL examples and query include at least one example configuration along with its corresponding runtime, followed by a request for the LLM to predict the runtime of an additional configuration. The problem and configurations are described in natural language to convey context, intent, and constraints. This allows the model's pretrained knowledge to bias how a configuration change might impact the expected runtime for the given configuration.

We provide the LLM with increasing amounts of configuration-runtime pairs, ranging from one to one hundred examples before requesting a runtime prediction. We select both the in-context learning examples and the query configuration randomly. We form five disjoint datasets with the same number of in-context learning examples to limit the possibility of poor examples biasing the results. To further reduce the impact of variance, we also evaluate the LLM's performance where all examples and the prediction task have minimal configuration-space editing distance. That is to say, all configurations are nearly identical to one another so that the query is as well-defined by the ICL as possible. We evaluate each prompt with three random seeds to minimize the effects of LLM sampling variance during generation. Finally, we attempt to account for the impacts of different underlying relationships and distributions in the training data by repeating the above with two distinct array sizes. Changing the array size

Example System Instructions

The user may describe their optimization problem to give specific context.
Then they will demonstrate hyperparameter configurations for a regression problems in a feature-rich text-based CSV format.
Following the examples, the user will provide a number of configurations without performance values; you will need to infer the objective based on their prior examples.
Do not alter the user's proposed configurations.
Do NOT explain your thought process. ONLY respond with your answer following the format that the user demonstrated for you.

Example User Problem Description

The problem considers source-code optimization for a loop nest in C++ code.
The 'size' parameter is invariant, but denotes a relativistic measure of the size of data inputs to the loop nest. Sizes can be represented by the following values sorted smallest-to-largest: S, SM, M, ML, L, XL
For size 'SM', M=130 and N=160.
Size is NOT a tunable component of the problem.
Tunable options in the configuration space are:
* The first and second array inputs to the problem can be independently packed, represented as True/False for each
* The outermost two loops in the nest may be interchanged, represented as True to perform interchange, else False
* Each loop (outer, middle, and inner) are tiled, and the tile sizes can all be independently specified.
The performance objective is the runtime of a program compiled with the modified source, so lower is better.
A pseudocode representation of the problem is:

```
input: Arrays A[N,M], B[N,M], C[N,N], scalar constant alpha
code segement:
# Optional packing array A
# Optional packing array B
# Optional interchange on outermost two loops
for i=0...N in tiles of size outer_loop_tiling_factor
  for j=0...M in tiles of size middle_loop_tiling_factor
    for k=0...i in tiles of size inner_loop_tiling_factor
      C[i,k] = A[k,j]*alpha*B[i,j] + B[k,j]*alpha*A[i,j]
```

Example User ICL Examples and Query

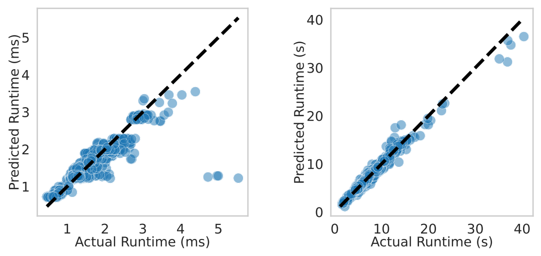
Here are the examples:
Hyperparameter configuration: size is SM, first_array_packed is True, second_array_packed is False, interchange_first_two_loops is False, outer_loop_tiling_factor is 80, middle_loop_tiling_factor is 64, inner_loop_tiling_factor is 100
Performance: ## 0.0022155 ##

Please complete the following:
Hyperparameter configuration: size is SM, first_array_packed is False, second_array_packed is True, interchange_first_two_loops is False,outer_loop_tiling_factor is 128,middle_loop_tiling_factor is 80, inner_loop_tiling_factor is 80
Performance:

Fig. 1: Example of the LLM system and user prompts for the Polybench/C Syr2k SM task

TABLE I: XGBoost Prediction Metrics

Training Examples	R^2		MARE		MSRE	
	SM	XL	SM	XL	SM	XL
100	0.44	0.69	0.17	0.13	0.073	0.058
500	0.67	0.87	0.12	0.09	0.038	0.036
1000	0.72	0.88	0.11	0.07	0.025	0.027
5000	0.80	0.97	0.09	0.04	0.015	0.007
8519 (80% Train)	**0.80**	**0.98**	**0.08**	**0.04**	**0.013**	**0.003**



(a) size SM (b) size XL

Fig. 2: XGBoost runtime predictions with 8519 training examples

changes the importance of features, their relationships to one another, and the output domain for the runtimes, representing a highly similar yet novel prediction task.

C. Evaluation Method

To generalize our understanding of LLM performance on this task, we locally execute the model and record all generated nonzero logit values. This allows us to construct all "feasible" generation alternatives in the given scenario. An exhaustive enumeration of this space would require restarting the model generation with each candidate token to observe different behaviors in the "next-token-prediction" learned objective, which immediately becomes combinatorically exponential with the sequence length. Instead, we consider all combinations reachable via alternative decodings of the original generation.

While instruction-tuned LLMs, including the model we utilize, generally follow directions, minor deviations in natural language can make harnessing model outputs challenging. We consider improvements to format consistency and instruction-following capabilities to be out-of-scope for this work. In our experiments, we manually identify all relevant portions of all outputs produced by the LLM.

We evaluate the success of the LLM by comparing its generated value for the query configuration against the ground truth value, focusing on Mean Absolute Relative Error (MARE), Mean Squared Relative Error (MSRE) and R^2 score as key success metrics. We focus on relative metrics to improve the comparability of our results across all experimental settings. By applying the Central Limit Theorem across all of our experiments, we can approximate the generalized capability of the LLM at this task.

D. A Baseline For Comparison

We consider the prediction of traditional ensemble machine learning techniques, namely XGBoost, a gradient-boosted ensemble of decision trees [29], [30], as a reasonable baseline for success. The XGBoost ensemble has tunable hyperparameters, including the number of estimators, learning rate, maximum tree depth and minimum number of samples per leaf node. We find the best-fitting model through a randomized search with 1000 iterations for varying amounts of available training data; the results for the best-fitting models are shown in Table I. Figure 2 demonstrates the high degree of accuracy by XGBoost across the domain of observations. The accuracy of XGBoost improves with more training data. However, both array sizes are amenable to surprisingly high-quality fits even with minimal training data, permitting the possibility for LLM-aided techniques to succeed despite limited context.

IV. Experimental Results

A. Quality of LLM Predictions

Contrary to our expectations, the LLM fails to provide many accurate predictions for the syr2k runtime, regardless of the training data it receives. The highest R^2 score our LLM achieves is 0.4643 on the SM dataset with 50 in-context learning examples, roughly equivalent to what XGBoost can achieve with 100 examples. Unfortunately, the LLM's performance does not continue to scale with additional data, and the LLM produces a non-negative R^2 score in only a quarter of our experiments, with an average R^2 score of -6.643 and a standard deviation of 22.766.

Our experiments span different output domains, data availability, data cohesion, and randomization seeds such that we can attempt to generalize the Mean Absolute Relative Error (MARE) and Mean Squared Relative Error (MSRE) of the model across all settings via the Central Limit Theorem [31] as the mean of MARE and MSRE gradually converge to the model's expected true capability.

In our experiments, the mean MARE is 0.3593 with a standard deviation of 0.2474, while the mean MSRE is 0.1021 with a standard deviation of 3.2609. These results are not accurate enough to recommend using LLMs in this setting. However, the error is small enough to warrant further investigation into how a model not explicitly designed for this task can achieve nontrivial success.

We observe that LLM prediction error often increases with additional ICL examples. At first, we believed that randomly sampling many examples may be confusing, which motivated our evaluation setting with maximally similar configurations. While this does not guarantee similar objective values, limiting the configuration variability generally reduces objective variability and could align better with the natural language modality of LLMs. Unfortunately, the LLM did not improve under these conditions. Instead, the generated values strongly cluster around the most common ICL values, but very few exact copies are generated. Slightly over 10% of the generated values in all experiments are directly copied from ICL, and our post-hoc analysis proves that the model was unlikely to copy any other values from the context. This suggests that the model attempts to produce a value similar to the ICL without

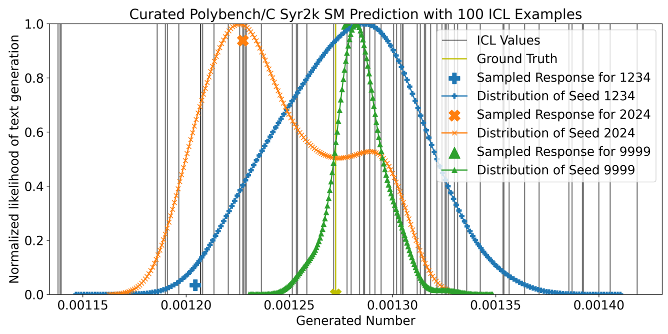


Fig. 3: When given a curated ICL dataset with minimal edit-distance, the LLM's responses still cluster around common prefixes of ICL values.

keen insight into what portion of the input should be given attention.

Figure 3 demonstrates this phenomenon, with the peak probabilities occurring near highly dense in-context examples. We also find that different seeds often produce identical token sets with slightly altered logit probabilities, supporting the hypothesis that the knowledge expression is primarily based on the prompt rather than a randomizable component of the model. This behavior is demonstrated in Figure 4, where the same sets of tokens are produced with only trivial deviations in logit probability, leading to highly similar sampling patterns.

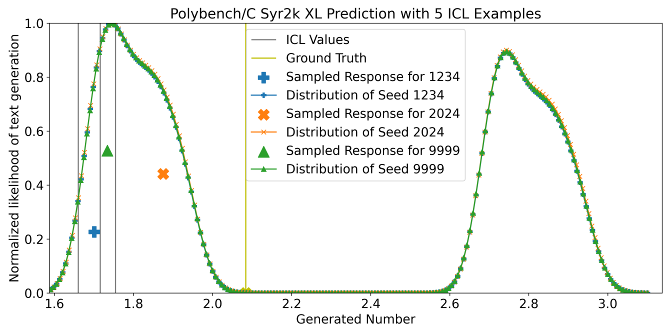


Fig. 4: Bi-modal value distributions commonly arise from different string prefixes (ie, 1.7 vs 2.7), even across different seeds.

B. *Understanding How LLMs Generate Decimal Sequences*

Text generated by LLMs is intended to meaningfully mimic prior content, which has several intriguing properties. In our experiments, a decimal digit sequence representing runtime requires a distinct token for the "." separator. As a result, each value string consists of at least three distinct tokens. Because tokens are generated sequentially, one after another, the initial prefix digits have the most significant influence on both its magnitude and all subsequent tokens in the sequence.

Table II displays the mean and standard deviation of the number of selectable tokens for each generated response across all of our experiments and the number of values the LLM can produce. Because the whole-number magnitude in our datasets is almost exclusively less than ten seconds, it is unsurprising that the second token is always a single choice (the period delimiter), and the first token has relatively limited options. Variation in the first token selection only exists for prompts with the XL array size, as all SM objective values are less than one, and the LLM appropriately reflects this. The third and fourth tokens form most of the error and variability in our study, each with an average of hundreds of possible options. The combinatoric space of unique digit sequences from these first four tokens immediately reaches and sometimes surpasses the cardinality of the configuration search space (10,648 possible configurations), making the optimal decoding process almost as challenging as identifying the global minimum in the original optimization problem.

TABLE II: Variability in amount of selectable tokens across all experiments.

	Mean # Possibilities	Std # Possibilities	# Samples
1st Token	4.176	8.805	284
2nd Token	1.000	0.000	284
3rd Token	318.835	353.677	284
4th Token	537.629	327.731	283
5th Token	10.164	45.333	201
6th Token	1.000	0.000	14
7th Token	1.143	0.515	14
8th Token	2.273	1.355	11
9th Token	4.000	0.000	1
Permutations	43,562,830	354,291,070	284

Not all tokens have equivalent string lengths, so the fourth token's decimal magnitude is often increased or decreased based on the string length of the third token. This poses a challenge to the model's modality, which is unique to LLMs as opposed to traditional optimization methods.

C. *Searching Within Distributions*

Despite the failures of the LLM so far, we attempt to extract useful information from the distribution of values it could produce. The first and most obvious strategy would be to utilize the mean or median of the distribution of possible values, as the ground truth value usually falls between the minimum and maximum generable values.

We repeat the same evaluations from Section IV-A using the mean and median value of the distribution rather than the sampled value. Both the mean and the median have worse errors than the observed samples, meaning that the distribution is not statistically centered in a meaningful manner. Previously, in Figure 4, we called out how the LLM often produces a bimodal distribution due to its textual modality. We find that the logit weights are often higher in the mode closer to the ground truth, but not to such a degree that this method resolves enough ambiguity to improve the model's response.

1) Needles in a Haystack: We use the distribution of generable values as a "haystack" where a hypothetical post-hoc decoder may search for "needles" or values within a given error-bound. This metaphor allows us to determine how a sophisticated technique may achieve error-bounded success at different thresholds. Across our experiments, over half of all LLM-generated values have 50% or less relative error. This is too broad to be useful; for comparison, XGBoost trained on 100 samples has 95% of all test values within the same error bound. The trend of XGBoost exceeding LLM performance persists at tighter error constraints. The LLM has 20% of its generated values that fall within 10% relative error compared to 52% for XGBoost. At the extremely tight 1% relative error bound, merely 3% of LLM values qualify as "needles" versus 6% for XGBoost. Neither technique excels beyond the 1% relative error threshold, meaning that XGBoost strongly outperforms the LLM's optimal capability across all error thresholds.

V. Discussion and Future Works

Our results lead to several key ideas regarding current and future uses of LLMs in HPC tuning and other quantitative applications based upon few-shot evaluations with in-context learning.

A. *On the Efficacy of Prior Evaluations*

While well-intended, our results indicate that several prior evaluations, such as those considering integer arithmetic and word problem solving, may not be as rigorous as previously believed. Other works have discovered that LLMs can memorize public-facing datasets [32] and can overfit and memorize training data even within a single epoch or less [33]. This means memorizing basic arithmetic appears more than feasible for the largest LLMs and may be even more accessible for more recent "reasoning" models. These models are permitted intermediate space to prepare a response, functioning as a scratch memory where information can be written down and rephrased into a representation that is more likely to have appeared in the training data. The intermediate space can also be leveraged for tool usage or to overcome some limitations of the next-token-prediction learning objective, ultimately producing a higher-quality final output for the user.

The community has also pushed to reduce or eliminate memorization at training time [34]. While most commercial LLM providers could benefit from incorporating such techniques, it cannot be assumed that current or future models are trained similarly. Therefore the possibility of memorization remains.

B. *Thoughts on Output Formats and LLM Post-Processing*

While some aspects of our experiments are likely to hold across related problem settings and datasets, some others may be partially addressable by other means. A stable output format can assist the LLM by providing predictable substrings, such as by expressing all values in scientific notation rather than decimals. However, scientific notation often makes the prefixes of values *less* similar, which our results indicate may *harm* the model's ability to generate useful answers.

We also observed many deviations from our prompt and example's imposed output format throughout our experiments, especially with large amounts of in-context learning examples. This is a common problem with LLMs that can sometimes be mitigated by techniques such as Langchain [35] and Guidance [36] or even cleaned up by subsequent LLM agents with instructions to reformat the text to match a given specification. While these techniques can be effective, the former often limit outputs in manners that may be destructive to task success, while the latter may also fail to follow directions.

C. *Reasoning Against Fine-Tuning*

We abstain from LLM fine-tuning for several reasons. Fine-tuning for specific optimizations may be feasible and should be expected to improve the model's capabilities. However, we do not expect fine-tuning and LLM inference to be more computationally efficient than existing non-LLM-based techniques suitable to such problems. Even with large models' fantastic capacity for memorization, collecting enough data to perform fine-tuning can often be too costly to perform in the first place or may involve producing enough data for other techniques to more efficiently solve the problem – as demonstrated by our baseline with XGBoost on this particular evaluation setting.

D. *Future Improvements to LLMs*

Our work identifies several problems that appear to be linked to the training objective of transformer models, the architecture's fundamental design, and limitations of textual modality. Despite these disappointments, a better understanding of the source of limitations creates opportunities for novel approaches to improve upon. In particular, an LLM can be given a unique token to signal to a supporting model that a number should be generated at a particular position within its response. This mimics modern LLM tool usage patterns by providing a hook for any number-generating process to transparently assist the LLM in providing higher-quality answers.

Separating this component permits fine-tuning and adaptation with smaller-scale models that only operate in quantitative domains, perhaps even interfacing with in-progress generation to still benefit from scenarios where "world knowledge" can be represented by the model in the prefix of its response.

VI. Conclusion

This work demonstrates several limitations of LLMs performing in-context learning for HPC performance tuning. Namely, the models struggle to incorporate "world knowledge" meaningfully, often parroting substrings from the contextual information that unexpectedly decreases performance as additional context is provided. We also find that producing numbers via textual representation has many less-than-ideal characteristics for precise and reasoned applications. Finally, we demonstrate that the distribution of tokens with nonzero logit values does not support a hypothetical improvement to the decoding process, necessitating different avenues of research for effective utilization of LLMs in this and similar applications.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[2] X. Wu, P. Balaprakash, M. Kruse, J. Koo, B. Videau, P. Hovland, V. Taylor, B. Geltz, S. Jana, and M. Hall, "ytopt: Autotuning scientific applications for energy efficiency at large scales," 2023. [Online]. Available: https://arxiv.org/abs/2303.16245

[3] Y. Liu, W. M. Sid-Lakhdar, O. Marques, X. Zhu, C. Meng, J. W. Demmel, and X. S. Li, "GPTune: multitask learning for autotuning exascale applications," in *Proceedings of PPoPP '21: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP'21. New York, NY, USA: Association for Computing Machinery, February 2021, pp. 234–246.

[4] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, *Bliss: Auto-Tuning Complex Applications Using a Pool of Diverse Lightweight Learning Models*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1280—1295. [Online]. Available: https://doi.org/10.1145/3453483.3454109

[5] T. Randall, J. Koo, B. Videau, M. Kruse, X. Wu, P. Hovland, M. Hall, R. Ge, and P. Balaprakash, "Transfer-learning-based autotuning using gaussian copula," in *Proceedings of the 37th ACM International Conference on Supercomputing*, ser. ICS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 37–49. [Online]. Available: https://doi.org/10.1145/3577193.3593712

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[7] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, "Metaicl: Learning to learn in context," *arXiv preprint arXiv:2110.15943*, 2022.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[9] A. H. Garg, S. Bhojanapalli, A. Kyrillidis, and P. Jain, "What can transformers learn in-context? a case study of simple function classes," *arXiv preprint arXiv:2208.01066*, 2022.

[10] O. Naim and N. Asher, "Two in-context learning tasks with complex functions," *arXiv preprint arXiv:2502.03503*, 2025. [Online]. Available: https://arxiv.org/abs/2502.03503

[11] X. Wang, B. Jiang, H. Xu *et al.*, "In-context learning on function classes unveiled for transformers," in *International Conference on Machine Learning (ICML)*, 2024. [Online]. Available: https://jhc.sjtu.edu.cn/~bjiang/papers/Wang_ICML2024_ICL.pdf

[12] R. Zhang, S. Frei, and P. L. Bartlett, "Trained transformers learn linear models in-context," *Journal of Machine Learning Research*, vol. 25, no. 1042, 2024. [Online]. Available: https://www.jmlr.org/papers/volume25/23-1042/23-1042.pdf

[13] J. Zepeda-Núñez *et al.*, "In-context learning by linear attention: Exact asymptotics and double descent," in *International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://openreview.net/pdf?id=Jw3ck7FWZh

[14] D. Ganguli, N. Schiefer, M. Favaro, and J. Clark. (2023) Challenges in evaluating AI systems. [Online]. Available: https://www.anthropic.com/index/evaluating-ai-systems

[15] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013/

[16] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[17] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[18] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval augmentation reduces hallucination in conversation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, Nov. 2021, pp. 3784–3803.

[19] A. Salemi, S. Kallumadi, and H. Zamani, "Optimization methods for personalizing large language models through retrieval augmentation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024, pp. 752–762.

[20] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, A. Jindal, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, G. Pasandi, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, "Chipnemo: Domain-adapted llms for chip design," 2024. [Online]. Available: https://arxiv.org/abs/2311.00176

[21] N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. Körfer, S. B. Hoo, R. T. Schirrmeister, and F. Hutter, "Accurate predictions on small data with a tabular foundation model," *Nature*, 01 2025. [Online]. Available: https://www.nature.com/articles/s41586-024-08328-6

[22] T. Liu, N. Astorga, N. Seedat, and M. van der Schaar, "Large language models to enhance bayesian optimization," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=OOxotBmGol

[23] G. Jawahar, M. Abdul-Mageed, L. V. S. Lakshmanan, and D. Ding, "Llm performance predictors are good initializers for architecture search," 2024. [Online]. Available: https://arxiv.org/abs/2310.16712

[24] M. Franke, P. Tsvilodub, and F. Carcassi, "Bayesian statistical modeling with predictors from llms," 2024. [Online]. Available: https://arxiv.org/abs/2406.09012

[25] M. Soni, "Llm-performance-evaluation-for-price-prediction-in-crypto-stocks," "https://github.com/Mousami7/LLM-Performance-Evaluation-for-Price-Prediction-in-Crypto-Stocks", Last accessed, February 16, 2025.

[26] Meta, "Introducing llama 3.1: Our most capable models to date," *Meta AI*, July 2024, 15 minute read. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[27] T. Yuki and L.-N. Pouchet, "PolyBench 4.2," 2016. [Online]. Available: https://sourceforge.net/projects/polybench/

[28] T. Grosser, A. Grösslinger, and C. Lengauer, "Polly – Performing Polyhedral Optimizations on a Low-Level Intermediate Representation," *Parallel Processing Letters*, vol. 22, no. 04, 2012. [Online]. Available: http://polly.llvm.org

[29] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[30] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[31] E. Miller, "Adding error bars to evals: A statistical approach to language model evaluations," 2024. [Online]. Available: https://arxiv.org/abs/2411.00640

[32] H. Haresamudram, H. Rajasekhar, N. M. Shanbhogue, and T. Ploetz, "Large language models memorize sensor datasets! implications on human activity recognition research," 2024. [Online]. Available: https://arxiv.org/abs/2406.05900

[33] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting training data from large language models," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2633–2650. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting

[34] A. Hans, Y. Wen, N. Jain, J. Kirchenbauer, H. Kazemi, P. Singhania, S. Singh, G. Somepalli, J. Geiping, A. Bhatele, and T. Goldstein, "Be like a goldfish, don't memorize! mitigating memorization in generative llms," 2024. [Online]. Available: https://arxiv.org/abs/2406.10209

[35] LangChain-AI, "Langchain," "https://github.com/langchain-ai/langchain", Last accessed, February 15, 2025.

[36] Guidance-AI, "Guidance," "https://github.com/guidance-ai/guidance", Last accessed, February 15, 2025.