AutoThink: efficient inference for reasoning LLMs

Asankhaya Sharma, Patched Codes, Inc
asankhaya@patchedcodes.com

Abstract

We explore different aspects of improving inference efficiency for reasoning LLMs. In particular, we study the impact of reasoning budgets, guided decoding and controlled steering on the accuracy of reasoning LLMs. Our approach is called AutoThink and it consists of two parts – a query complexity classifier which determines the number of reasoning tokens we allow during inference before generating the final response, and a dataset of control vectors we use to steer the response during inference. We derive the dataset of control vectors from pivotal tokens for the LLM that are discovered using a search procedure over the distribution of responses generated by the LLM on a calibration dataset. Our findings show that AutoThink can reduce the average number of output tokens by 55% while improving accuracy by 43% on GPQA-Diamond. Our implementation is open-source and available at https://github.com/codelion/optillm.

Introduction

Large language models (LLMs) have demonstrated remarkable reasoning capabilities, particularly when guided to "think step by step" or produce chains of thought [2,12,15]. However, these reasoning processes often generate excessive tokens, leading to computational inefficiency and occasionally producing redundant or even harmful content [7,8,9]. The challenge of optimizing inference for reasoning models has become increasingly important as LLMs are deployed in resource-constrained environments and time-sensitive applications.

This paper introduces AutoThink, a novel approach to improving inference efficiency for reasoning LLMs. AutoThink addresses two critical aspects of efficient reasoning: determining the appropriate reasoning budget and steering the model toward more effective reasoning paths. Our approach comprises two key components:

1. A query complexity classifier that dynamically determines the optimal number of reasoning tokens before generating the final response.
2. A dataset of control vectors derived from pivotal tokens that steer the response during inference.

Our experiments demonstrate that AutoThink significantly reduces the average number of output tokens while simultaneously improving accuracy on challenging reasoning tasks. The implementation is open-source and available at https://github.com/codelion/optillm.

Related Work

There have been a number of papers recently that address the issue of improving accuracy of reasoning LLMs [20,21] using inference time techniques. In this section, we briefly review most closely related papers, doing a more detailed survey on the topic is beyond the scope of this work.

Reasoning Capabilities in LLMs

Recent work has focused on enhancing reasoning capabilities in LLMs through techniques like reinforcement learning from human feedback (RLHF) [1,2]. DeepSeek-R1 [2] has shown how reinforcement learning can incentivize reasoning behavior in LLMs. However, Yang et al. [1] question whether RLHF genuinely improves reasoning or merely optimizes for surface-level patterns that mimic reasoning.

Inference Optimization

Several approaches aim to optimize inference efficiency for LLMs. Test-time scaling [3] provides a simple yet effective method for improving model performance without retraining. Sharma [5] developed optiLLM, a framework for optimizing inference proxies for LLMs, showing significant performance improvements on diverse tasks [4,10].

Thinking Efficiency in LLMs

Recent research has identified both "overthinking" and "underthinking" issues in LLMs [7,8,22]. Models like o1 have been observed to exhibit underthinking, where they rush to conclusions without sufficient deliberation [7]. Conversely, other studies have identified overthinking, where models generate excessive tokens that don't contribute to improved reasoning [8].

ThinkPrune [9] and LightThinker [13] address this by pruning redundant reasoning steps. Yu et al. [14] propose inference-aware optimization to achieve adaptive reasoning, while Ma et al. [15] question whether explicit reasoning is necessary at all. Lin et al. [16] introduce "sleep-time compute" as an alternative to traditional inference scaling.

Token Budget Approaches

Han et al. [17] propose token-budget-aware reasoning for LLMs, demonstrating that strategic allocation of token budgets can improve efficiency. Similarly, Chuang et al. [18,19] explore confidence-based routing of LLMs to optimize resource allocation during inference.

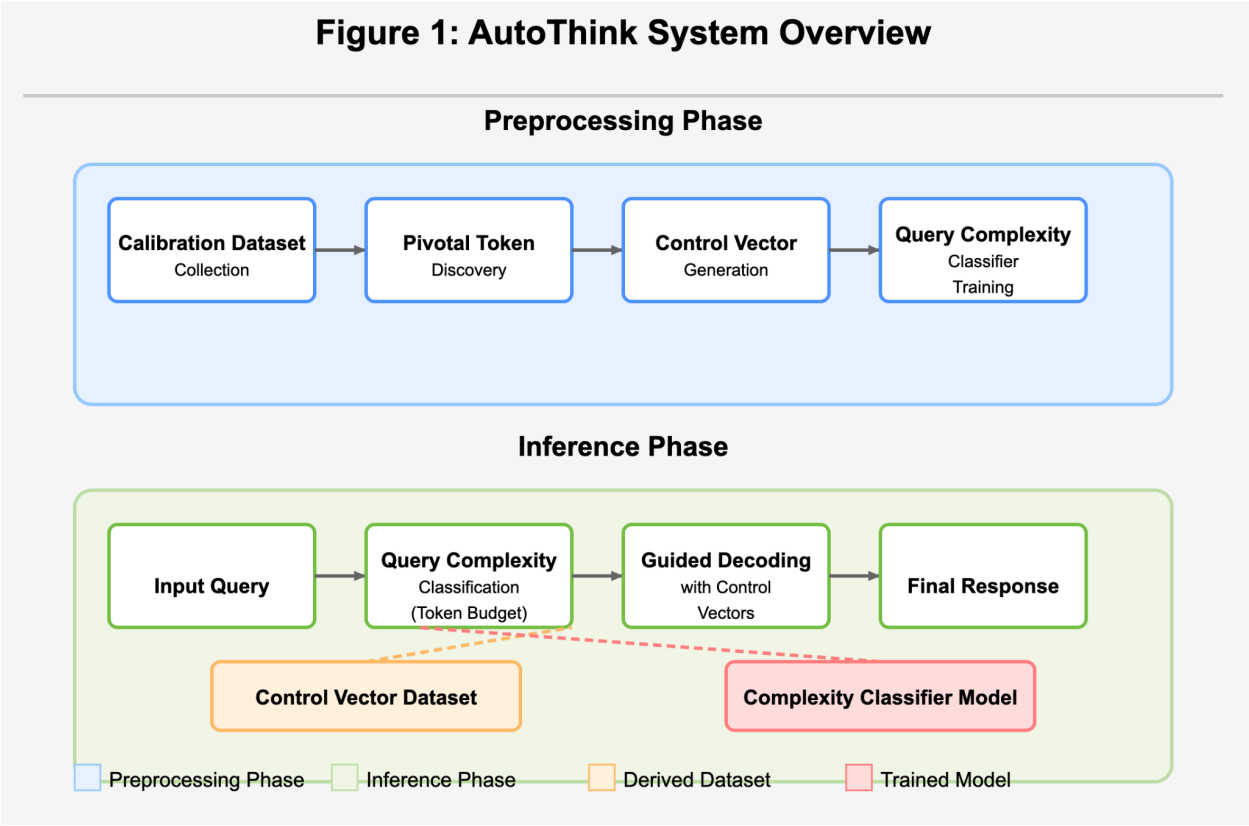
Model Steering and Control

Recent advances in model steering and control [23,24] provide mechanisms for guiding LLM outputs. Konen et al. [23] introduced style vectors for steering generative LLMs, while Im and Li [24] offer a unified understanding and evaluation of steering methods. These approaches inform our development of control vectors for guiding reasoning paths.

Methodology

AutoThink is designed to optimize inference efficiency while maintaining or improving reasoning accuracy. The system operates in two phases: a preprocessing phase and an inference phase (Figure 1).

Figure 1: AutoThink System Overview



In the preprocessing phase:

1. We collect a calibration dataset of query-response pairs.
2. We conduct a search procedure to identify pivotal tokens that influence reasoning quality.
3. We derive control vectors from these pivotal tokens.
4. We train a query complexity classifier on the calibration dataset.

During inference:

1. The query complexity classifier determines the appropriate token budget.
2. The control vectors guide the decoding process toward effective reasoning paths.
3. Once the token budget is reached, the model generates the final response.

Query Complexity Classification

The query complexity classifier estimates the number of reasoning tokens needed for a given query. We formulate this as a regression problem, training a classifier on a calibration dataset where each query is associated with the number of tokens in successful reasoning paths.

The classifier architecture is based on an adaptive classification approach [11], allowing continuous learning and adjustment as new data becomes available. We use a pre-trained language model as the feature extractor and fine-tune it to predict token budgets.

The classifier categorizes queries into complexity levels:

- Low complexity: 0-500 tokens
- Medium complexity: 500-1500 tokens
- High complexity: 1500+ tokens

Pivotal Token Discovery

Pivotal tokens [25] are those that significantly influence the reasoning path quality. We identify these tokens through a search procedure over the distribution of responses generated by the LLM on a calibration dataset. For our experiments, we use the optiLLMbench dataset as it contains a diverse set of queries from logic, reasoning and math problems. Any other similar dataset can be used to discover the pivotal tokens.

Our pivotal token search algorithm, is open-source and implemented in the https://github.com/codelion/pts repository [6], it works as follows:

1. Generate multiple reasoning paths for each query in the calibration dataset.
2. Classify paths as successful or unsuccessful based on accuracy.
3. Compare token distributions between successful and unsuccessful paths.
4. Identify tokens that appear significantly more often in successful paths.
5. Rank tokens by their impact on reasoning quality.

Control Vector Generation

From the identified pivotal tokens, we derive control vectors that can steer the model during inference. These vectors represent directions in the embedding space that guide the model toward more effective reasoning.

For each pivotal token, we:

1. Extract its corresponding embedding from the model.
2. Compute the difference between this embedding and the average embedding of non-pivotal tokens.
3. Normalize the resulting vector.
4. Store the vector in our control vector dataset.

During inference, these control vectors are applied selectively based on the query type and current decoding state.

Guided Decoding with Token Budget

During inference, we implement a guided decoding process with a token budget constraint:

1. The query complexity classifier determines the token budget T for the query.
2. As decoding proceeds, we maintain a counter of tokens generated.
3. At each decoding step, we apply the relevant control vectors to steer the generation.
4. When the token counter reaches T, we signal the model to generate a final answer.
5. The final answer is extracted from the generated text.

This approach ensures that the model allocates computational resources efficiently while maintaining reasoning quality.

Experimental Setup

Datasets

We evaluate AutoThink on two challenging reasoning datasets:

1. **GPQA-Diamond**: A subset of GPQA focusing on difficult questions requiring advanced reasoning.
2. **MMLU-Pro**: An enhanced version of the MMLU benchmark with more complex questions across various domains.

These datasets were chosen because they require sophisticated reasoning and are sensitive to the quality of the reasoning process.

Models and Baselines

We use the following model for our experiments:

- **DeepSeek-R1-Distill-Qwen-1.5B** [2]: A distilled version of DeepSeek-R1, specifically designed for reasoning tasks.

We compare AutoThink against the following baselines:

1. **Vanilla**: The base model with recommended temperature (0.6) setting.
2. **Fixed Budget**: Vanilla with a token budget based on query complexity.

Evaluation Metrics

We evaluate our approach using the following metrics:

1. **Accuracy**: The percentage of correctly answered questions.
2. **Token Efficiency**: The average number of tokens generated per query.

Implementation Details

Our experiments use the following configuration:

- Temperature: 0.6
- Top-p: 0.95
- Maximum query processing time: 600 seconds
- Each experiment is repeated 10 times, and we report the average results.

The implementation is built on top of the optiILLM framework [5], with extensions for query complexity classification and control vector application.

Results and Analysis

Performance Comparison

Table 1 shows the performance comparison between AutoThink and the baselines on GPQA-Diamond and MMLU-Pro.

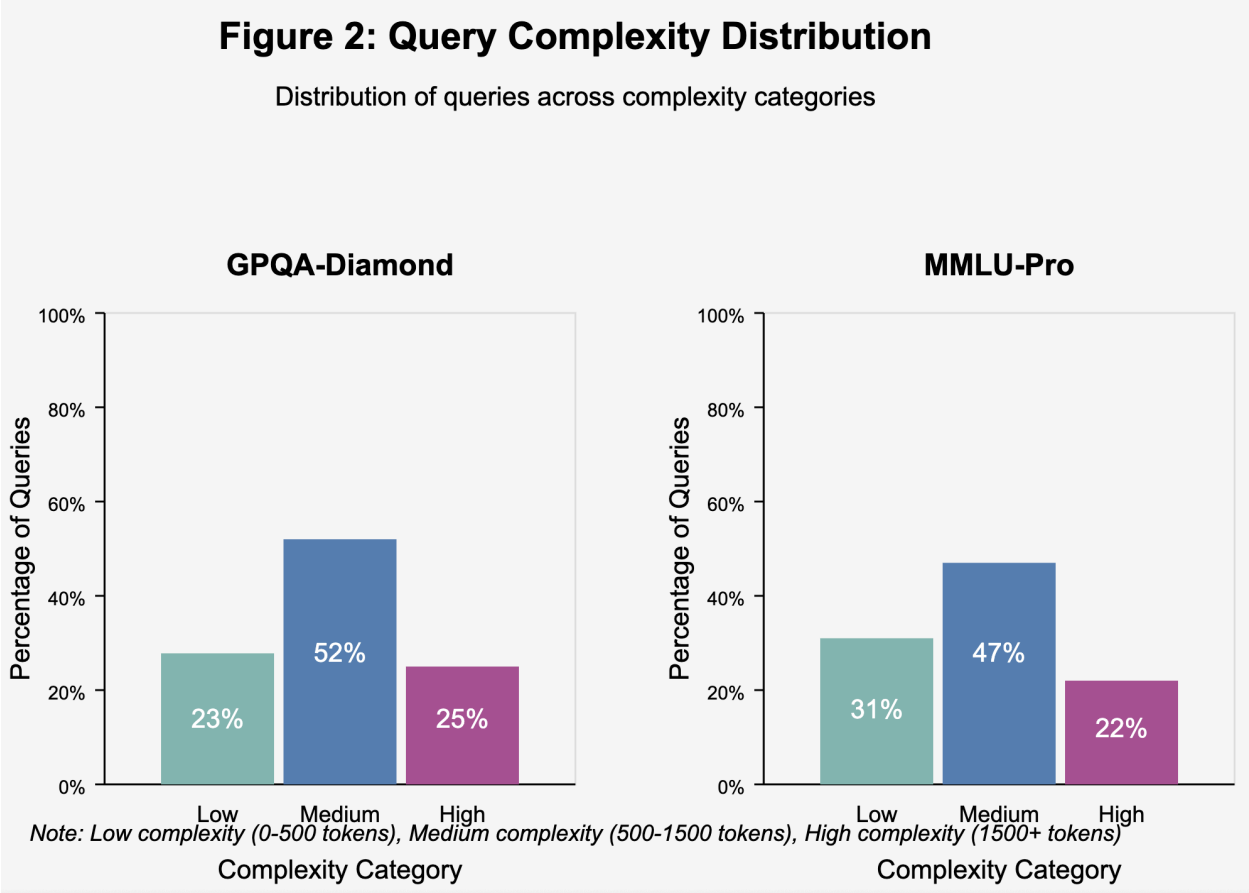
Table 1: Performance Comparison on GPQA-Diamond and MMLU-Pro

Method	**GPQA-Diamond**		**MMLU-Pro**	
	Accuracy (%)	Avg. Tokens	Accuracy (%)	Avg. Tokens
Vanilla	21.72	7868.26	25.58	2842.75
Fixed Budget	28.47	3570.00	26.18	1815.67
AutoThink	**31.06**	**3520.52**	**26.38**	**1792.50**

On GPQA-Diamond, AutoThink achieves a 43% improvement in accuracy over the vanilla baseline while reducing the average number of tokens by 55%. We also observe improvements on MMLU-Pro, with a 3% accuracy improvement and 37% token reduction.

Query Complexity Distribution

Figure 2 shows the distribution of queries across complexity categories as determined by our classifier.



For GPQA-Diamond:

- Low complexity: 23%
- Medium complexity: 52%
- High complexity: 25%

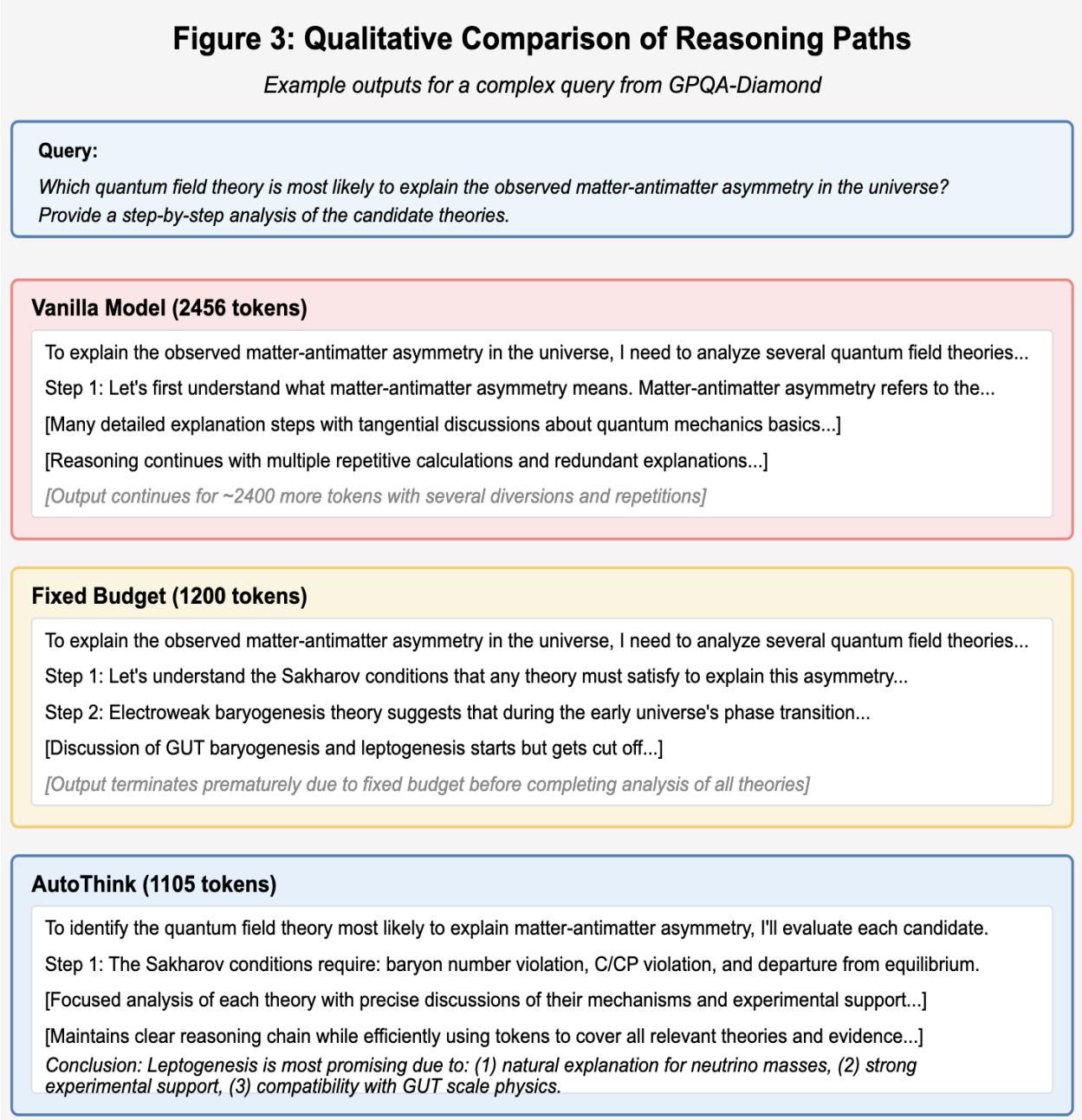
For MMLU-Pro:

- Low complexity: 31%
- Medium complexity: 47%
- High complexity: 22%

This distribution highlights the diversity of reasoning requirements across different queries, supporting the need for adaptive token budgeting.

Qualitative Analysis

We conducted a qualitative analysis of the reasoning paths produced by AutoThink compared to the baseline methods. Figure 3 illustrates example outputs for a complex query from GPQA-Diamond.



The analysis reveals that:

1. The vanilla model often produces redundant reasoning steps and sometimes meanders without reaching a conclusion.
2. Fixed budget approaches may cut off reasoning prematurely for complex queries.
3. AutoThink produces more focused reasoning paths, targeting the most relevant aspects of the problem and efficiently reaching accurate conclusions.

Discussion

Efficiency-Accuracy Tradeoff

Our results demonstrate that the traditional tradeoff between efficiency and accuracy can be overcome with appropriate techniques. By dynamically allocating resources based on query complexity and steering the model toward effective reasoning paths, AutoThink achieves both higher efficiency and better accuracy.

This challenges the assumption that more computation necessarily leads to better reasoning [15]. Instead, we show that strategic allocation of computational resources is more important than the total amount of computation.

Implications for Model Scaling

The success of AutoThink has implications for model scaling strategies. Rather than simply increasing model size or computation, focusing on inference optimization can yield substantial improvements in both efficiency and effectiveness [16]. This is particularly relevant for deploying reasoning LLMs in resource-constrained environments.

Limitations and Future Work

Despite its strong performance, AutoThink has several limitations:

1. The query complexity classifier's effectiveness depends on the quality and diversity of the calibration dataset.
2. The pivotal token discovery process is computationally intensive and may need to be repeated for different domains or models.
3. The control vectors may not generalize well across substantially different model architectures.

Future work could address these limitations by:

1. Developing more efficient methods for pivotal token discovery.
2. Exploring transfer learning approaches for the query complexity classifier.

3. Investigating domain-adaptive control vectors that adjust to different application contexts.
4. Extending the approach to multi-modal reasoning tasks.

Conclusion

This paper introduced AutoThink, a novel approach to improving inference efficiency for reasoning LLMs. By combining query complexity classification and control vector steering, AutoThink achieves significant improvements in both efficiency and accuracy on challenging reasoning tasks.

Our results demonstrate a 55% reduction in token generation while improving accuracy by 43% on GPQA-Diamond, challenging the notion that more computation necessarily leads to better reasoning. The open-source implementation of AutoThink provides a foundation for further research and applications in efficient reasoning with LLMs.

As LLMs continue to be deployed in diverse environments with varying resource constraints, approaches like AutoThink that optimize the reasoning process will be increasingly important. Future work will focus on improving the generalizability and adaptability of the approach across different domains and model architectures.

References

[1] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song and Gao Huang. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? (https://arxiv.org/abs/2504.13837), 2025.

[2] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning (https://arxiv.org/abs/2501.12948), 2025.

[3] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès and Tatsunori Hashimoto. s1: Simple test-time scaling (https://arxiv.org/abs/2501.19393), 2025.

[4] Asankhaya Sharma. Patched RTC: evaluating LLMs for diverse software development tasks (https://arxiv.org/abs/2407.16557), 2024.

[5] Asankhaya Sharma. Optillm: Optimizing inference proxy for LLMs (https://github.com/codelion/optillm), 2024.

[6] Asankhaya Sharma. PTS: Pivotal Token Search (https://github.com/codelion/pts), 2025.

[7] Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian

Yu, Juntao Li, Zhuosheng Zhang, et al. Thoughts Are All Over the Place: On the Underthinking of o1-Like LLMs (https://arxiv.org/abs/2501.18585), 2025.

[8] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do NOT Think That Much for 2+3=? On the Overthinking of o1-Like LLMs (https://arxiv.org/abs/2412.21187), 2024.

[9] Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas and Shiyu Chang. ThinkPrune: Pruning Long Chain-of-Thought of LLMs via Reinforcement Learning (https://arxiv.org/abs/2504.01296), 2025.

[10] Asankhaya Sharma. Patched MOA: optimizing inference for diverse software development tasks (https://arxiv.org/abs/2407.18521), 2024.

[11] Asankhaya Sharma. Adaptive Classifier: Dynamic Text Classification with Continuous Learning (https://github.com/codelion/adaptive-classifier), 2025.

[12] Junjie Yang, Ke Lin and Xing Yu. Think When You Need: Self-Adaptive Chain-of-Thought Learning (https://arxiv.org/abs/2504.03234), 2025.

[13] Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen and Ningyu Zhang. LightThinker: Thinking Step-by-Step Compression (https://arxiv.org/abs/2502.15589), 2025.

[14] Zishun Yu, Tengyu Xu, Di Jin, Karthik Abinav Sankararaman, Yun He, Wenxuan Zhou, Zhouhao Zeng, Eryk Helenowski, Chen Zhu, Sinong Wang, et al. Think Smarter not Harder: Adaptive Reasoning with Inference Aware Optimization (https://arxiv.org/abs/2501.17974), 2025.

[15] Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min and Matei Zaharia. Reasoning Models Can Be Effective Without Thinking (https://arxiv.org/abs/2504.09858), 2025.

[16] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica and Joseph E. Gonzalez. Sleep-time Compute: Beyond Inference Scaling at Test-time (https://arxiv.org/abs/2504.13171), 2025.

[17] Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma and Zhenyu Chen. Token-Budget-Aware LLM Reasoning (https://arxiv.org/abs/2412.18547), 2024.

[18] Yu-Neng Chuang, Helen Zhou, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki and Xia Hu. Learning to Route LLMs with Confidence Tokens (https://arxiv.org/abs/2410.13284), 2024.

[19] Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Zirui Liu, Xuanting Cai, Yang Sui, Vladimir Braverman and Xia Hu. Confident or Seek Stronger: Exploring Uncertainty-Based On-device LLM Routing From Benchmarking to Generalization (https://arxiv.org/abs/2502.04428), 2025.

[20] Gaurav Srivastava, Shuxiang Cao and Xuan Wang. Towards Reasoning Ability of Small Language Models (https://arxiv.org/abs/2502.11569), 2025.

[21] Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan and Lu Hou. Quantization Hurts Reasoning? An Empirical Study on Quantized Reasoning Models (https://arxiv.org/abs/2504.04823), 2025.

[22] Chenrui Fan, Ming Li, Lichao Sun and Tianyi Zhou. Missing Premise exacerbates Overthinking: Are Reasoning Models losing Critical Thinking Skill? (https://arxiv.org/abs/2504.06514), 2025.

[23] Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz and Tobias Hecking. Style Vectors for Steering Generative Large Language Model (https://arxiv.org/abs/2402.01618), 2024.

[24] Shawn Im and Yixuan Li. A Unified Understanding and Evaluation of Steering Methods (https://arxiv.org/abs/2502.02716), 2025.

[25] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 Technical Report (https://arxiv.org/abs/2412.08905), 2024.