Predictive Scheduling for Efficient Inference-Time Reasoning in Large Language Models

Anonymous Authors[1]

Abstract

Large language models (LLMs) achieve state-of-the-art accuracy on complex reasoning tasks by generating multiple chain-of-thought (CoT) traces, but using a fixed token budget per query leads to over-computation on easy inputs and under-computation on hard ones. We introduce Predictive Scheduling, a plug-and-play framework that pre-runs lightweight predictors—an MLP on intermediate transformer hidden states or a LoRA-fine-tuned classifier on raw question text—to estimate each query's optimal reasoning length or difficulty before any full generation. Our greedy batch allocator dynamically distributes a fixed total token budget across queries to maximize expected accuracy. On the GSM8K arithmetic benchmark, predictive scheduling yields up to 7.9 percentage points of absolute accuracy gain over uniform budgeting at identical token cost, closing over 50% of the gap to an oracle with perfect foresight. A systematic layer-wise study reveals that middle layers (12–17) of the transformer carry the richest signals for size estimation. These results demonstrate that pre-run budget prediction enables fine-grained control of the compute–accuracy trade-off, offering a concrete path toward latency-sensitive, cost-efficient LLM deployments.

1. Introduction

Modern LLMs deliver exceptional chain-of-thought CoT reasoning capabilities, powering applications from real-time code autocomplete to interactive tutoring and decision support. Yet inference-time costs remain a critical bottleneck: applying a fixed token budget per query either wastes tokens on trivial inputs or starves hard ones, leading to unnecessary latency and inflated cloud bills—key concerns for production LLM services and on-device deployments.

Prior approaches either rely on few-shot heuristics to guess a single optimal trace length (Han et al., 2024), perform single-checkpoint early-termination checks during generation (Li et al., 2024; Fu et al., 2024a), or schedule batches based on surface-level signals such as queue length or past runtimes (Liu et al., 2023; Damani et al., 2024). None leverage the rich internal features of transformer hidden states to learn fine-grained, pre-run estimates of required trace length and query difficulty, nor integrate these estimates into a global batch allocator under a fixed total token budget.

In this paper, we present Predictive Scheduling, a plug-and-play framework that performs pre-run estimation of reasoning length and query difficulty, and then applies a greedy allocator to distribute a fixed total token budget across a batch so as to maximize expected accuracy.

Our main contributions are: (1) a lightweight MLP predictor that maps intermediate transformer hidden states to an estimate of the *token length* required for a reasoning trace to achieve correctness; (2) a lightweight *difficulty classifier*—using LoRA fine-tuning—to categorize queries as easy, medium, or hard before generation; (3) a greedy batch allocation algorithm that dynamically assigns per-trace token budgets to maximize expected accuracy gains under a fixed total budget; and (4) results demonstrating that on the GSM8K arithmetic reasoning benchmark predictive scheduling yields up to 7.9% absolute accuracy improvement at equal token cost relative to nonadaptive scheduling, closing over half the gap to an oracle with perfect size and difficulty estimates.

2. Related Work

2.1. Chain-of-Thought and Meta-Decoding

LLMs improve multi-step reasoning through intermediate "thought" steps: Chain-of-Thought prompting decomposes problems to boost arithmetic and logical accuracy (Wei et al., 2022), self-consistency aggregates multiple chains to reduce error (Wang et al., 2022), and Tree-of-Thoughts explores partial solutions (Yao et al., 2023). These methods yield

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

strong gains but use fixed chain lengths or uniform budgets, requiring substantial extra computation.

2.2. Batch Scheduling and Token Reordering

For mixed-difficulty requests, vLLM predicts remaining tokens to reorder and preempt queries (Liu et al., 2023), related pre-scheduling techniques reduce latency in multi-tenant serving (Zhang et al., 2021), and Dynasor allocates extra compute to "hard" queries at fixed checkpoints (Fu et al., 2024a). These external schedulers improve average latency but rely on surface-level heuristics rather than learned per-query estimates of needed reasoning length.

2.3. Adaptive Compute Within Transformers

Internal model computation adapts based on input difficulty: Depth-adaptive Transformers vary executed layers (Elbayad et al., 2020) and Mixture-of-Experts architectures route tokens through expert sub-modules (Fedus et al., 2021). These approaches adjust per-input compute "on the fly" but don't provide pre-run, per-query length estimates or batch-level scheduling under fixed token budgets.

2.4. Per-Query Token-Budget Prediction and Greedy Allocation

Wu et al. (2025) demonstrate optimal CoT length exists per problem as accuracy first increases then plateaus with chain length. TALE uses few-shot prompts and small post-trained models to predict optimal token budgets (Han et al., 2024). Fu et al. (2024b) train length-ordering predictors via learning-to-rank. Li et al. (2024) introduce learned early-stopping criteria during generation. Damani et al. (2024) train lightweight predictors to estimate marginal reward of additional reasoning traces with greedy allocation under fixed budgets. These methods forecast token usage but don't leverage the LLM's hidden states or identify which transformer layers carry the strongest predictive signal.

3. Methods

3.1. Dataset Preprocessing and Experimental Setup

We conducted our experiments on the GSM8K dataset (Cobbe et al., 2021), a benchmark of grade school math word problems. Each example consists of a natural language math question and its solution expressed as a series of reasoning steps followed by a final numerical answer.

For each question in both the training and test sets, we produced a 16-dimensional early-stopping probability vector by first tokenizing the question with DeepSeek's tokenizer to record the input length Q. We then generated one hundred independent reasoning traces per question using temperature 0.7 and top-p 0.95. Within each trace we inserted a fixed probe string at every 16 tokens up to a maximum of 256 tokens (see Appendix A.1.2); this probe string forces the model to emit a final answer at that point. After each probe insertion we extracted the model's answer and compared it to the ground truth. Finally, for each probe point (16, 32, ..., 256 tokens) we computed the fraction of the one hundred traces that were correct, yielding the early-stopping probability for that token budget.

3.1.1. Training Data Generation

For each question in the train set, we extracted the 1536-dimensional hidden state at the [CLS] token from every transformer layer (layers 1 through 28), yielding 28 feature vectors per example. The target labels comprised two sets of values. We derived difficulty labels—easy, medium, or hard—by computing each question's correctness probability under a 256-token per-query reasoning budget, and assigning the bottom 20 percentile to "hard," the top 20 percentile to "easy" (with thresholds $p_{20} = 0.18$ and $p_{80} = 0.84$), and all others to "medium." These features and labels for GSM8K queries formed the train and test sets for both our early-stopping and difficulty-classification models.

3.2. MLP-based Early Stopping Prediction

Allocating too few tokens to difficult problems leads to incorrect answers, while allocating too many tokens to simple problems wastes computational resources. By accurately predicting early stopping probabilities—the likelihood that a model can correctly solve a problem after generating a specific number of reasoning tokens—we can make informed decisions about resource allocation. These predictions serve as a foundation for our batch allocation strategies (detailed in Section 3.2.1), enabling us to dynamically distribute a fixed token budget across diverse queries to maximize overall accuracy.

We train simple MLP models that map hidden state representations from each hidden layer to early stopping probability vectors.

3.2.1. Early Stopping Probability Prediction Based Token Budget Allocation

The learned MLP or finetuned LORA models (see Appendix A.2.2) are trained on the GSM8K train set, and tested out of sample on the GSM8K test set.

Starting with a minimum allocation of 16 tokens per problem, the algorithm greedily assigns additional 16-token windows to problems where the predicted gain in accuracy (based on the predicted solution size) is highest. This process continues until either the token budget is exhausted or no further positive gains are expected. This approach allows us to adaptively allocate more tokens to problems predicted

to require longer solutions while maintaining the average token usage within the specified budget constraint. The pseudocode for this greedy algorithm is left to Section A.6.1 of the appendix.

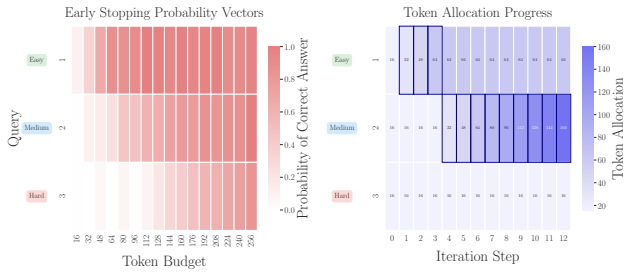

Figure 1: **Left:** Early stopping probability vectors showing the likelihood of generating a correct answer given different token budgets for each query. **Right:** Progressive token allocation across queries during algorithm execution.

3.3. Lora-based Difficulty Prediction

While early stopping probability vectors provide fine-grained predictions of reasoning success at various token budgets, they may be unnecessarily detailed for practical allocation strategies. We hypothesize that a simpler, more interpretable approach—classifying problems into discrete difficulty categories—could provide sufficient signal for effective token budget allocation while reducing algorithmic complexity.

3.3.1. LoRA Fine-tuned Classification Model

While few-shot in-context classification of problem difficulty may provide an efficient baseline (see Appendix A.3), we predict that a base model fine-tuned on our classification dataset would achieve higher accuracy, and the resulting classifications can serve as direct inputs to our token budget allocation strategies. To create this classifier, we finetuned DeepSeek-R1-Distill-Qwen-1.5B with LoRA.

3.4. Difficulty-Based Token Budget Allocation

We partition the inference-time queries into three classes—easy, medium, and hard—using the difficulty predictions described in Section 3.1.1. Let p_1, p_2, and p_3 denote the fractions of queries in the current inference batch predicted as easy, medium, and hard, respectively. For each category $k \in \{\text{easy}, \text{medium}, \text{hard}\}$, define

$$\text{acc}_k(b) = \frac{1}{N_k} \sum_{q_i \in \mathcal{C}_k} \Pr\big(\text{correct } q_i \text{ answer} \;\big|\; b \text{ tokens}\big)$$

where $\mathcal{C}_k$ is the set of all training examples in category k and $N_k = |\mathcal{C}_k|$. Thus $\text{acc}_k(b)$ is the expected accuracy when allocating b reasoning tokens per query to problems of difficulty k, computed from the element-wise average of all early-stopping probability vectors of problems of difficulty k in the GSM8K training set.

Under an average per-query budget B and a window size W, we search for the optimal per-category budgets $(b_1, b_2, b_3) \in \{W, 2W, \ldots, 16W\}^3$ by solving

$$\begin{aligned} \max_{b_1,b_2,b_3} \quad & p_1 \,\text{acc}_{\text{easy}}(b_1) + p_2 \,\text{acc}_{\text{medium}}(b_2) \\ & + p_3 \,\text{acc}_{\text{hard}}(b_3) \\ \text{s.t.} \quad & p_1\, b_1 + p_2\, b_2 + p_3\, b_3 \le B\,. \end{aligned}$$

The inputs to Algorithm 2 are the set of inference queries Q, the budget B, the window size W, the vector of difficulty predictions, and the batch proportions p_1, p_2, p_3. After computing (b_1, b_2, b_3), each query in class k receives b_k reasoning tokens. The pseudocode for this algorithm is left to Section A.6.2 of the appendix.

4. Results

We perform an extensive analysis of which hidden states help our MLP predictors to best perform for early stopping prediction. We find that hidden states—particularly from middle layers—encode significant information about reasoning complexity, with predictions produced by the best MLP trained on hidden states from layer 16 having a maximum correlation of **0.742** (Fig 8) with ground truth early stopping probabilities. We find that a LoRA-based predictor (A.2.2) demonstrated lower performance compared to our best MLP model (correlation of 0.444) for continuous early stopping prediction.

4.0.1. Token Budget Allocation Using Early Stopping Predictions

We implemented a greedy allocation algorithm that leverages our MLP predictions from layer 16 (our best-performing layer) to distribute a fixed token budget across a batch of problems. The algorithm first assigns a minimum budget to all problems, then iteratively allocates additional tokens to problems predicted to gain the most accuracy from increased reasoning budget.

As baselines, we examine the accuracy of the non-adaptive strategy, where the token budget is uniform across all queries, as well as the accuracy of the oracle strategy, where we use the known early stopping correctness probabilities and allocate tokens using the same greedy strategy. In order for our proposed method to be useful, it must outperform the non-adaptive baseline. The oracle method provides a notion of the best possible accuracy performance given perfect predictive abilities using the greedy method.

Figure 2 presents the comparative performance of these allocation strategies across different token budgets. For smaller token budgets between 16-96 reasoning tokens on average

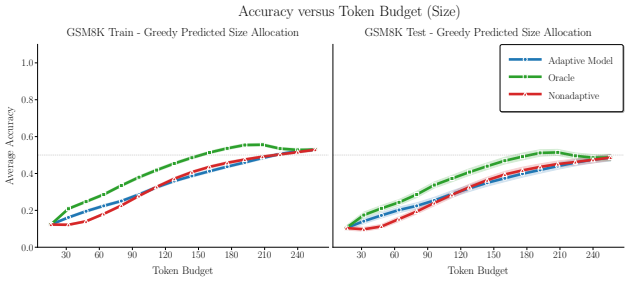

Figure 2: Accuracy vs. Token Budget for adaptive allocation using early stopping predictions from a 2-layer MLP trained on layer 16 hidden states. The plot compares our adaptive allocation based on MLP predictions, a non-adaptive baseline with uniform allocation, and an oracle allocation using ground truth early stopping probabilities.

per query, the adaptive allocation method using MLP model predictions outperforms the non-adaptive method. This improvement is most pronounced in the most constrained budget regimes (16-48 tokens).

For higher average token budgets, however, the model predictions prove insufficiently accurate, and the adaptive model underperforms the non-adaptive baseline. This performance inversion occurs around 96-128 tokens per query, suggesting a crossover point where the cost of prediction errors begins to outweigh the benefits of adaptive allocation. As budget constraints relax, the uniform allocation strategy eventually allocates sufficient tokens to most problems, reducing the advantages of adaptivity.

The gap between our adaptive allocation and the oracle performance indicates substantial room for improvement in early stopping predictions. This gap is particularly pronounced at higher token budgets, suggesting that while our current predictors capture enough signal for effective allocation in constrained settings, they struggle to identify the optimal stopping points for problems requiring longer reasoning chains.

4.1. Discrete Difficulty Classification Results

In addition to the early stopping prediction work, we investigate the effectiveness of classifying problems into discrete difficulty categories (easy, medium, hard) for token budget allocation. We find that the LoRA fine-tuned model significantly outperformed the few-shot in-context approach, achieving **66.3% test** accuracy compared to **41.6%** for few-shot in-context classification. Moreover, we examine and support the validity of using averaged early stopping vectors for difficulty-based categorization in Appendix (A.4.2).

4.2. Early Stopping Probability vs. Difficulty: Which Prediction Framework Yields Better Allocation?

To address this question directly, we conducted a comparative analysis of allocation strategies based on both approaches under identical token budget constraints.

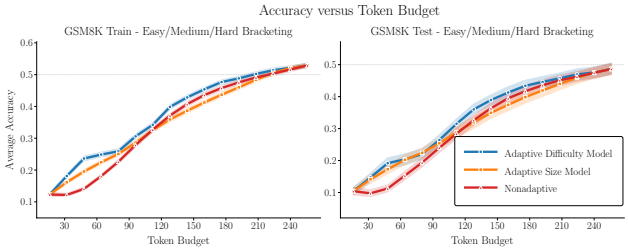

Figure 3: Comparative analysis of allocation strategies across fixed per-query token budgets. Difficulty-based allocation (using LoRA fine-tuned predictions), size-based allocation (using MLP layer 16 predictions), and the non-adaptive baseline (uniform allocation).

Figure 3 presents a side-by-side comparison of accuracy achieved by different allocation methods across fixed per-query token budgets. The results reveal a consistent pattern: difficulty-based allocation outperforms size-based allocation across all evaluated token budgets for both train and test sets. This performance advantage persists despite the seemingly coarser granularity of the tripartite difficulty classification compared to the continuous early stopping probability vectors.

5. Conclusion

Predictive Scheduling reframes inference with large language models as a budget-constrained optimization problem, demonstrating that internal hidden-state signals from middle transformer layers and lightweight difficulty classifiers enable more effective token allocation than uniform or heuristic schemes. On the GSM8K benchmark, our method achieves up to a 7.9 percentage-point accuracy improvement over uniform budgeting while closing more than half the gap to an ideal oracle, all without any modifications to the underlying language model.

This plug-and-play framework is, to our knowledge, the first to treat token-budget assignment as a pre-run optimization over internal model representations, offering a general template for resource-aware inference in large models.

Future work should evaluate the approach across diverse benchmarks, investigate adaptive window sizes for finer-grained allocation, and examine robustness under domain shifts, adversarial inputs, and fairness constraints.

Impact Statement

Predictive scheduling for large language model inference dynamically allocates a fixed token budget across queries based on per-query difficulty or estimated reasoning length. By reducing wasted computation on trivial inputs and avoiding under-allocation on hard ones, this approach can substantially lower inference latency and resource usage, leading to reduced energy consumption in data centers and edge devices. The lower compute requirements also make high-quality chain-of-thought reasoning more affordable and practical for smaller organizations, researchers, and users in resource-constrained environments. Faster responses for simpler queries can enhance the usability of interactive applications such as tutoring systems, code assistants, and conversational agents.

There are ethical considerations that warrant attention. Prediction errors may systematically under-allocate computation to certain categories of queries, potentially degrading performance for underrepresented dialects or novel problem types unless fairness evaluations and monitoring are in place. Inputs could be adversarially crafted to exploit the scheduling predictors, causing degraded accuracy or denial-of-service–type behavior in critical systems. In deployments where human users rely on model outputs for high-stakes decisions, transparency about the use of adaptive scheduling is important to maintain user trust and allow stakeholders to understand potential limitations.

We believe that predictive scheduling offers a practical route to more sustainable and widely accessible language model deployments. Future work should address fairness auditing, robustness to adversarial inputs, and transparent reporting practices to ensure that the benefits of adaptive compute allocation are realized responsibly and equitably.

References

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Damani, M., Shenfeld, I., Peng, A., Bobu, A., and Andreas, J. Learning how hard to think: Input-adaptive allocation of lm computation, 2024. URL https://arxiv.org/abs/2410.04707.

Elbayad, M. et al. Depth-adaptive transformers. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Fedus, W. et al. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Fu, Y., Chen, J., Zhu, S., Fu, Z., Dai, Z., Qiao, A., and Zhang, H. Efficiently serving llm reasoning programs with certaindex. *arXiv preprint arXiv:2412.20993*, 2024a.

Fu, Y., Zhu, S., Su, R., Qiao, A., Stoica, I., and Zhang, H. Efficient llm scheduling by learning to rank, 2024b. URL https://arxiv.org/abs/2408.15792.

Han, T., Fang, C., Zhao, S., Ma, S., Chen, Z., and Wang, Z. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*, 2024.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Li, Y., Yuan, P., Feng, S., Pan, B., Wang, X., Sun, B., Wang, H., and Li, K. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning, 2024. URL https://arxiv.org/abs/2401.10480.

Liu, C. et al. vllm: Fast inference for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Wang, X. et al. Self-consistency improves chain-of-thought reasoning in language models. In *Advances in Neural Information Processing Systems*, 2022.

Wei, J. et al. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Wu, Y., Wang, Y., Du, T., Jegelka, S., and Wang, Y. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*, 2025.

Yao, Z. et al. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2303.11171*, 2023.

Zhang, Y. et al. Efficient inference scheduling for transformer models. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

A. Appendix

A.1. Data & Preprocessing

A.1.1. Data Splits and Validation

We maintained the original GSM8K train/test split to ensure comparability with previous work. The final test set of 1,294 examples was used only for final evaluation. The processed dataset statistics are as follows:

Split	Total	Easy	Medium	Hard
Train	7,450	1,506	4,437	1,507
Test	1,294	271	760	263

Table 1: Dataset statistics after preprocessing and difficulty stratification

To ensure robustness, we verified that the difficulty distribution in the test set closely matched that of the training set. We also confirmed that the average question length and vocabulary distribution were consistent across splits.

On GSM8K data, we display the early stopping probabilities. Notice that for lower token budgets, the probability that the generated answer will be correct is clustered almost wholly around 0. For higher reasoning budgets, the probabilities that the generated answer will be correct for the given reasoning budget are concentrated near 1. Furthermore, there are some questions for which even the greatest reasoning budgets almost always result in an incorrect answer–we would refer to these as hard/impossible questions. For other questions, even a moderate increase in the reasoning budget leads to significant expected accuracy gains.

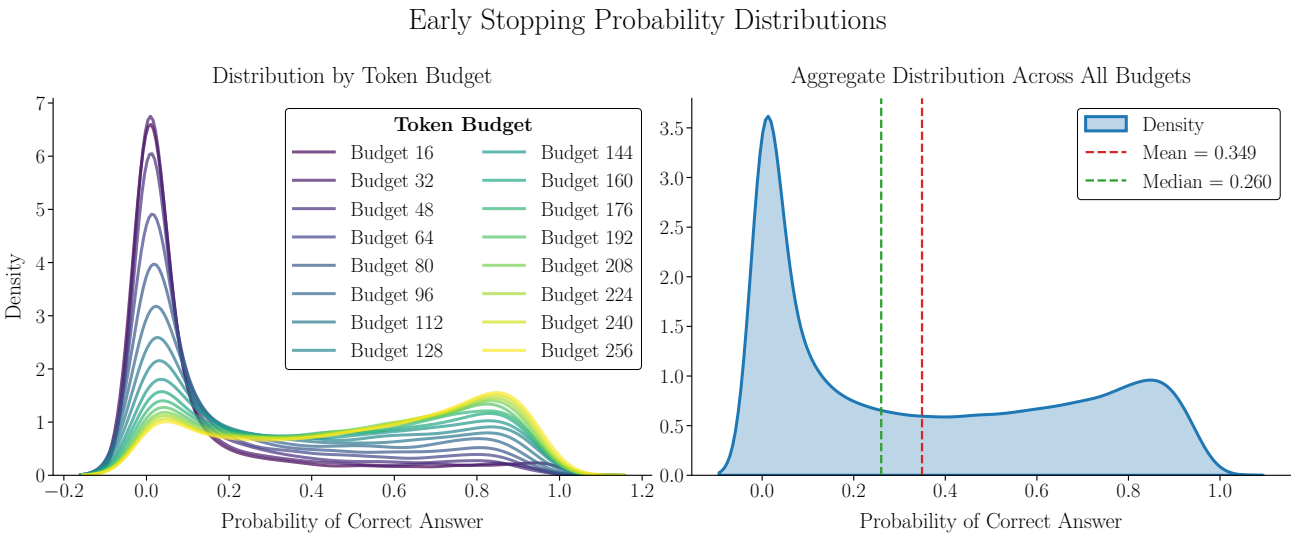

Figure 4: (a) Distribution of correct-answer probabilities at each token budget; (b) Aggregate KDE over all budgets.

A.1.2. Probe String

Within each trace we inserted a fixed probe string

> "Oh, I suddenly got the answer to the whole problem, **Final Answer**\n\n\[\boxed{"

at every 16 tokens up to a maximum of 256 tokens; this probe string forces the model to emit a final answer at that point. After each probe insertion we extracted the model's answer and compared it to the ground truth. Finally, for each probe point (16, 32, . . . , 256 tokens) we computed the fraction of the one hundred traces that were correct, yielding the early-stopping probability for that token budget.

A.2. Model Architectures

A.2.1. MLP-Based Early Stopping Predictors Architecture and Training

Our MLP predictors follow a consistent architecture comprising two fully connected layers with 256 hidden units and ReLU activation functions, and a final sigmoid activation to constrain outputs to [0,1]. The input dimension corresponds to the hidden state size (1536 for DeepSeek-R1-Distill-Qwen-1.5B), and the output dimension matches the number of prediction points (16, corresponding to token budgets from 16 to 256 in steps of 16). We train separate MLPs for each transformer layer (1-28) to systematically evaluate which layers encode the most predictive signals for reasoning length requirements (we compare the best layers in Appendix A.4.1).

A.2.2. LoRA Early-Stopping Predictor: Model & Implementation

As an alternative to extracting and processing hidden states externally, we hypothesize that early stopping prediction is fundamentally a linguistic task, where semantic understanding of the problem statement correlates with reasoning complexity. Unlike our MLP approach that operates on extracted hidden states, we explore a more integrated approach using Low-Rank Adaptation (LoRA) (Hu et al., 2022). Our LoRA implementation processes the raw question text as input and directly produces early stopping success probabilities across different token budgets.

This method enables parameter-efficient fine-tuning by inserting trainable low-rank matrices into the transformer architecture while keeping pre-trained weights frozen.

Moreover, this design leverages the model's pre-trained linguistic knowledge to identify semantic patterns that we hypothesize correlate with reasoning complexity. For instance, we expect that mathematical questions containing terms like "prove," "derive," or "show that" typically signal higher reasoning complexity requiring longer token budgets, while problems with directives such as "calculate," "find," or "evaluate" often indicate more straightforward computational tasks that can succeed with earlier stopping points. Our LoRA fine-tuning approach aims to capture these linguistic indicators directly from the input text, potentially enabling more accurate early stopping predictions based on the semantic content of the problem statement.

Architecture and Implementation. Our implementation fine-tunes the DeepSeek-R1-Distill-Qwen-1.5B model directly on the early stopping prediction task. The architecture consists of the base language model augmented with LoRA adapters (rank $r = 16$, scaling factor $\alpha = 32$) targeting the query and value projection matrices in the attention mechanism. This targeted approach allows us to efficiently adapt the model's reasoning capabilities to our specific task while minimizing parameter count.

Prediction Pipeline. The prediction pipeline consists of the LoRA-adapted language model followed by a regression head implemented as a two-layer MLP with a hidden dimension of $(h_{\text{model}}/2)$, where h_{model} is the base model's hidden size (1536). We extract the final hidden state corresponding to the last token of the input sequence and process it through layer normalization, dimensionality reduction to $h_{\text{model}}/2$, ReLU activation, final linear projection to output dimension (16), and sigmoid activation to constrain predictions to $[0, 1]$.

Training Details. We optimize the model using mean squared error loss between the predicted early-stopping probabilities and the ground truth. Training is carried out with the AdamW optimizer at a learning rate of 1×10^{-4} for ten epochs, using a batch size of 32 for training and 8 for evaluation. Hyperparameter tuning explores LoRA rank values of 8, 16, or 32; LoRA scaling factors of 16, 32, or 64; learning rates of 5×10^{-5}, 1×10^{-4}, or 2×10^{-4}; and alternative choices for the regression head architecture.

A.2.3. LoRA Difficulty Classification Head & Training

Architecture and Implementation. This classification model follows the same input processing as the early-stopping predictor but produces a discrete label instead of a probability vector. The base language model is extended with LoRA adapters of rank $r = 16$ and scaling factor $\alpha = 32$, applied to the query and value projection matrices in the attention layers. The classification head begins by applying layer normalization to the final hidden state of the input sequence, then projects it linearly to a dimension of $h_{\text{model}}/2$. A ReLU activation is applied next, followed by dropout with probability 0.1 for regularization. The final step is a linear projection that maps to three output logits corresponding to the easy, medium, and hard classes.

Training Methodology. We employed cross-entropy loss during training and evaluated performance using accuracy, precision, recall, and F1 score—with particular attention to balanced performance across all difficulty classes. As discussed in Section 3.1.1, the ground truth difficulty labels were derived from the 256-token performance data, using the 20th and 80th percentile thresholds from the training set (specifically, $p_{20} = 0.18$ and $p_{80} = 0.84$) to establish the boundaries between easy, medium, and hard problems.

Our implementation uses a batch size of 32 for training and 8 for evaluation, with an AdamW optimizer and learning rate of 1×10^{-4}. We trained the model for up to 50 epochs with early stopping based on validation accuracy, allowing sufficient time for the model to learn difficulty patterns while preventing overfitting.

A.3. Few-Shot Classification with LLMs

We investigate whether state-of-the-art LLMs can perform effective few-shot difficulty classification without specialized training. This approach is motivated by the hypothesis that commercially available models have already internalized substantial knowledge about mathematical problem complexity through their pre-training on diverse corpora.

Implementation. We implemented a classification pipeline using GPT (o4-mini) to analyze each problem statement and assign a difficulty rating based on the question text. The prompt instructed the model to classify the question as easy, medium, or hard, to provide a brief justification for its choice, and to return the result in a structured JSON format suitable for automated processing (see Appendix A.7.2).

The prompt defined the difficulty categories using a single labeled example of an easy question, a single labeled example of a medium question, and a single labeled example of a hard question.

This approach leverages the pre-existing mathematical knowledge and ICL abilities embedded in the LLM to identify linguistic markers of difficulty. For example, the model recognizes that problems containing terms like "calculate the sum" or "find x" typically represent easier tasks, while those involving "prove that" or requiring multi-step reasoning with multiple variables indicate higher difficulty levels. The JSON-formatted responses facilitated automated processing and integration with our token budget allocation system.

A.3.1. Comparative Analysis of Difficulty Classification Models

To test the hypothesis that linguistic features in problem statements correlate with reasoning complexity, we implemented and evaluated two approaches for difficulty classification: (1) few-shot classification using the o4-mini-high model and (2) LoRA fine-tuning of a DeepSeek-1.5B model.

The results provide substantial evidence supporting the linguistic complexity hypothesis. The LoRA fine-tuned model significantly outperformed the few-shot approach, achieving 66.3% test accuracy compared to 41.6% for few-shot classification. This 24.7 percentage point improvement demonstrates that linguistic patterns in problem statements contain sufficient signal for meaningful difficulty classification.

Figure 5 presents confusion matrices for both classification approaches. The LoRA fine-tuned model shows stronger diagonal elements, indicating better classification across all difficulty levels. Notably, both models struggle most with the medium category, frequently misclassifying these problems as either easy or hard. This confusion likely stems from the inherent ambiguity at category boundaries, as our difficulty categorization imposes discrete labels on what is fundamentally a continuous spectrum of reasoning complexity.

These results, when compared with our earlier early stopping prediction findings, provide important nuance to our earlier linguistic complexity hypothesis. While linguistic features proved insufficient for precise prediction of continuous early stopping probability vectors (achieving correlation of only 0.444), they demonstrate much stronger predictive power for discrete difficulty classification (66.3% accuracy). This contrast suggests a refinement of our original hypothesis: linguistic patterns in problem statements do correlate with reasoning complexity, but this relationship is more effectively captured through coarse-grained categorization than through fine-grained continuous prediction.

A.3.2. Difficulty-Based Token Allocation Performance

Leveraging our difficulty classifiers, we implemented a greedy token allocation algorithm that distributes a fixed token budget across problems based on their predicted difficulty levels. The algorithm begins by assigning each problem a

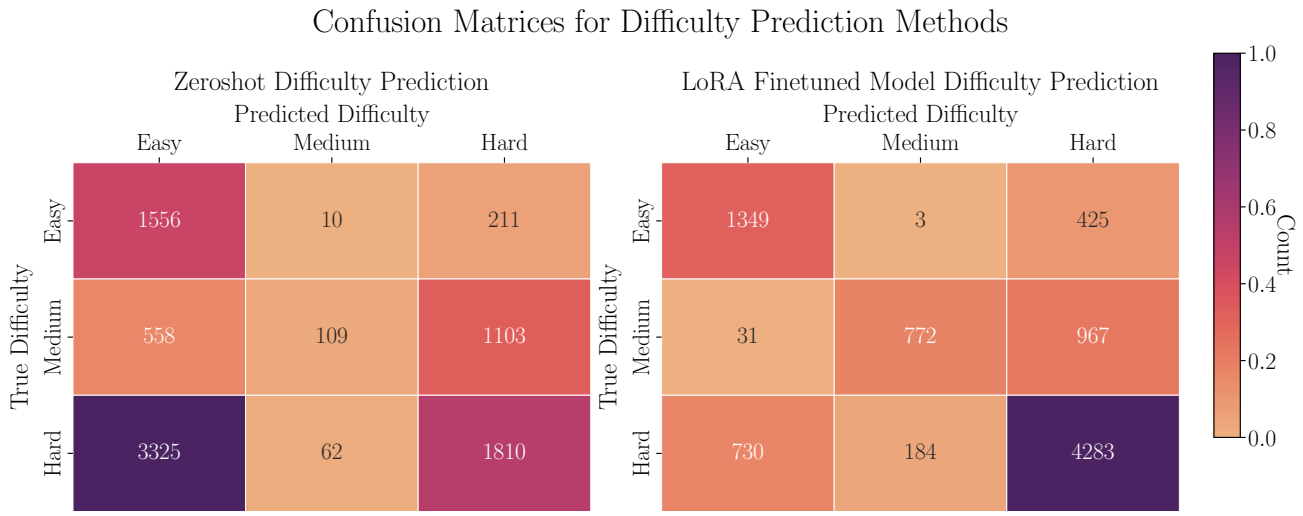

Figure 5: Confusion matrix of predicted vs actual difficulties for both the few-shot and LoRA finetuned prediction models. The matrices reveal that both models struggle most with the medium category, though the LoRA finetuned model (right) shows substantially stronger diagonal elements (66.3% overall accuracy) compared to the few-shot approach (left, 41.6% accuracy).

minimum allocation of 16 tokens, then iteratively allocates additional 16-token windows based on expected accuracy gains for each difficulty class.

This allocation strategy relies on pre-computed mappings from difficulty levels to expected accuracy improvements per token window, derived from the averaged early stopping vectors shown in Figure 10. At each iteration, the algorithm prioritizes allocating tokens to problems where the predicted marginal accuracy gain is highest, continuing until either the token budget is exhausted or no further positive gains are expected.

Figure 6 presents the accuracy-versus-token-budget curves for both few-shot and LoRA-based difficulty prediction. The LoRA-based approach outperforms few-shot classification across most token budgets, with the performance gap widening at intermediate budgets (approximately 80-160 tokens per problem).

The allocation patterns themselves reveal insights about optimal resource distribution. Figure 7 shows that at constrained token budgets, the algorithm heavily favors easy problems, which offer the highest marginal accuracy gains per token. As the budget increases, allocation gradually shifts toward medium problems and finally to hard problems at the highest budgets. This behavior emerges naturally from the expected accuracy curves—easy problems show steep initial gains but quick saturation, while hard problems require substantial token investment before showing meaningful improvements.

Comparing across allocation strategies, our difficulty-based approach proves particularly effective at intermediate token budgets, where the constraints force non-trivial allocation decisions. In these regimes, difficulty-based allocation outperforms uniform allocation by identifying which problems would benefit most from additional computation. Interestingly, despite using coarser categorization than our continuous early stopping predictors, the difficulty-based allocation achieves comparable or better performance in many budget regimes, suggesting that the simplification to discrete categories preserves most of the signal needed for effective allocation while being more robust to prediction noise.

The effective performance of difficulty-based allocation, particularly with the LoRA-based classifier, suggests that linguistic patterns in problem statements do indeed correlate with reasoning complexity, but this relationship is more robustly captured through discrete categorization than continuous vector prediction.

A.4. Layer-Wise Analysis & Validation

A.4.1. Layer-wise Analysis of Hidden State Features

We posit that specific layers of the transformer architecture would encode the strongest predictive signals about reasoning complexity. To explore this, we trained identical MLPs on hidden states extracted from each of the 28 layers of DeepSeek-

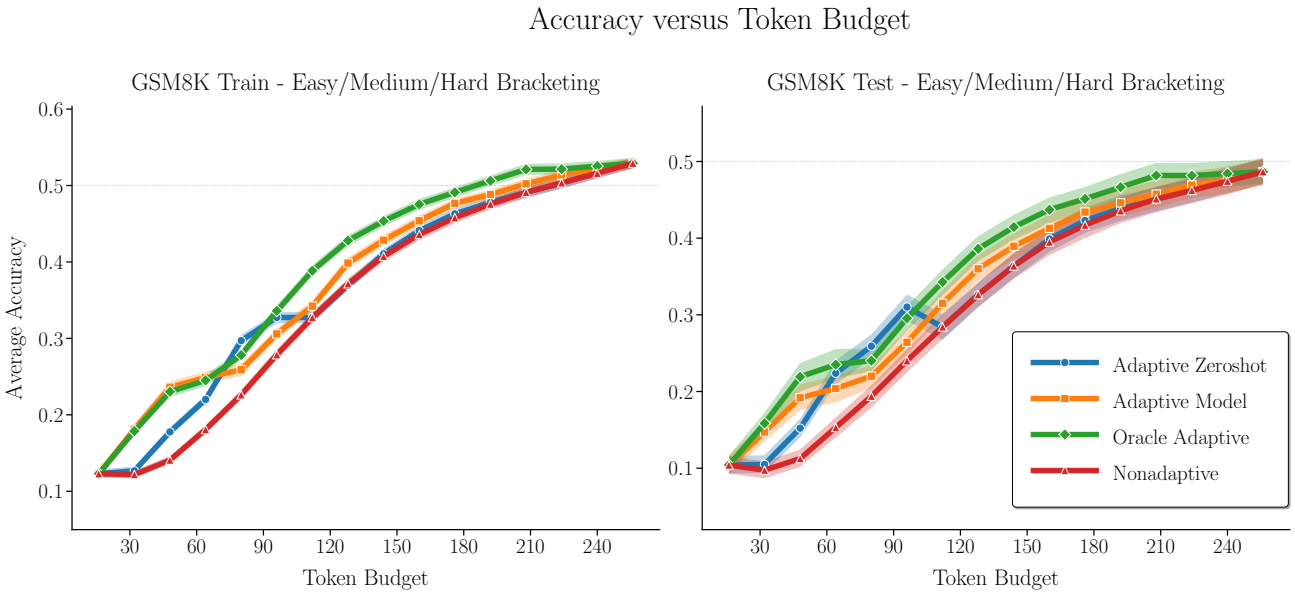

Figure 6: Accuracy vs. Token Budget using few-shot or LoRA-finetuned difficulty predictions for token allocation. The plot demonstrates that LoRA-finetuned difficulty classification consistently leads to more effective token allocation across all budget levels, with the performance advantage becoming particularly pronounced at intermediate token budgets (80-160 tokens per problem).

R1-Distill-Qwen-1.5B, with each MLP predicting the probability of correct answers at different reasoning budgets.

Figure 8 presents comprehensive evidence supporting our hypothesis. Middle layers (particularly 12-17) significantly outperform both early and late layers, with layer 16 achieving the highest test correlation of 0.742. This confirms our prediction that intermediate representations capture an optimal balance of syntactic and semantic features relevant to reasoning difficulty.

The performance distribution follows a clear inverted U-shape across model depth, with early layers (1-6) showing correlations below 0.6 and late layers (21-28) dropping to similar levels. This pattern aligns with our hypothesis that early layers primarily capture surface-level features insufficient for reasoning complexity assessment, while later layers become too specialized toward output generation to retain general reasoning signals.

Figure 9 examines prediction efficiency using the correlation-to-loss ratio, which measures predictive power per unit of error. This analysis further confirms the effectiveness of middle layers (12-17), with these layers achieving 15-20% higher efficiency than early or late layers. This quantitative result strongly supports our hypothesis about the advantageous position of middle transformer layers for reasoning complexity prediction.

While our results validate that hidden states—particularly from middle layers—encode significant information about reasoning complexity, the maximum correlation of 0.742 suggests this relationship is more complex than initially theorized. The moderate correlation indicates that while transformer hidden states contain substantial predictive signals, additional factors not captured in these representations likely influence reasoning difficulty as well.

A.4.2. Validation of Difficulty-Based Categorization

Before implementing difficulty-based allocation strategies, we needed to verify that our discrete categorization meaningfully captures differences in early stopping behavior. Specifically, we investigated whether element-wise averaging of early stopping probability vectors within each difficulty class provides a reasonable basis for allocation decisions.

Figure 10 presents a comparative analysis of early stopping probability curves across the three difficulty categories. The plots reveal distinct patterns: easy problems show rapid improvement in success probability with increasing token budget, medium problems demonstrate more gradual improvement, and hard problems maintain consistently low success probabilities across most token budgets.

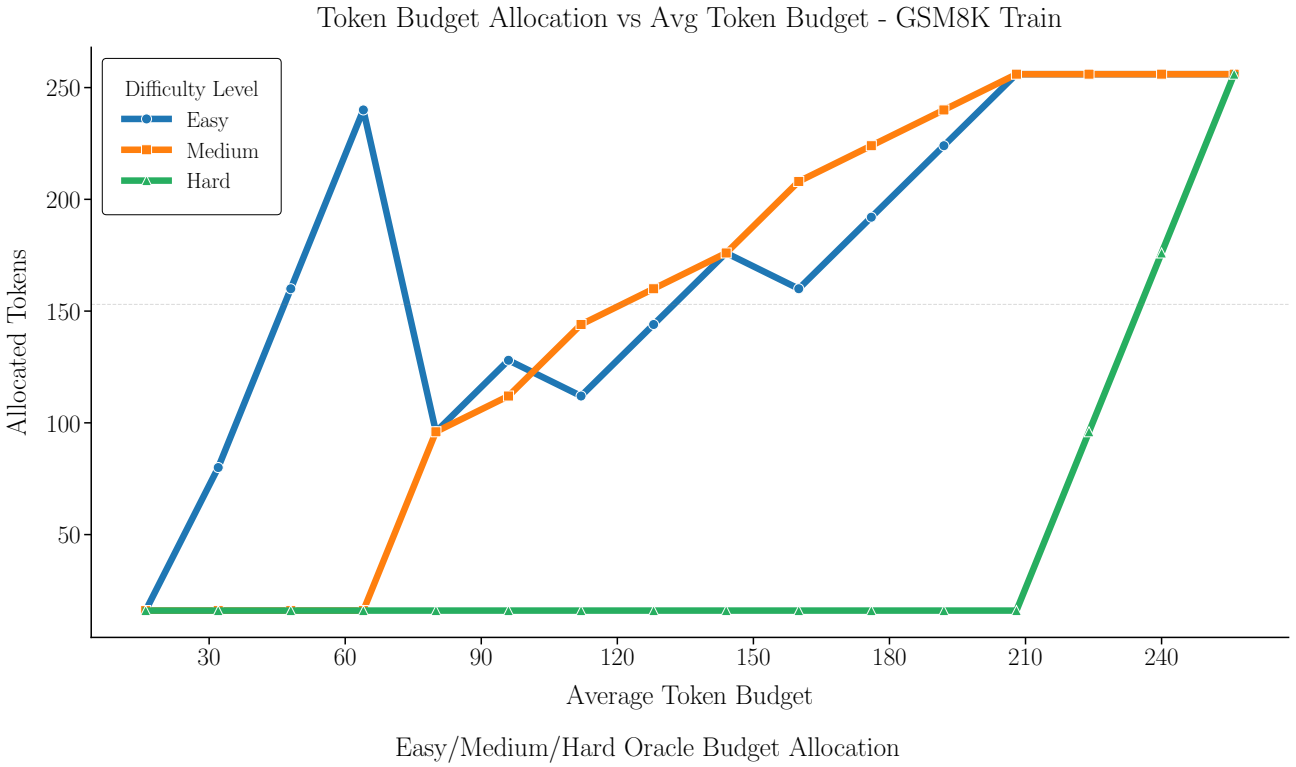

Figure 7: For lower token budgets, in the oracle greedy algorithm, most reasoning budget is allocated to easy questions to maximize accuracy gains. For increasing token budget, more tokens are allocated to medium questions and then last to hard questions to maximize expected accuracy gains. This allocation pattern emerges naturally from the different expected accuracy trajectories of each difficulty category, where easy problems offer high initial returns but quick saturation, while hard problems require substantial investment before showing meaningful improvement.

The averaged vectors for train and test sets (blue and red lines, respectively) track closely within each difficulty category, indicating that these categorizations capture consistent patterns that generalize across data splits. The hard category shows the most consistent behavior across different token budgets, with success probabilities remaining near zero for most budget levels before showing minimal improvement at the highest budgets. In contrast, the easy category exhibits the greatest variability, with success probabilities rising sharply between 16 and 96 tokens before plateauing.

A.5. Hyperparameters

A.5.1. MLP Hyperparameter Search

The MLP models trained in 3.2 used the Adam optimizer with MSE as the loss. We employ early stopping based on validation performance to prevent overfitting. To ensure optimal performance for each layer, we conduct a hyperparameter sweep over the number of hidden layers (1 to 3), hidden layer size (128, 256, or 512 units), learning rate (1×10^{-4}, 5×10^{-4}, or 1×10^{-3}), and dropout rate (0.0, 0.1, or 0.2) to select the best configuration for each layer's predictor.

Our multilayer perceptron predictors use either one or two hidden layers. In the single-layer variant the hidden dimension is set to 128, 256, or 512 units; in the two-layer variant both layers have either 128 units each or 256 units each. We sample the learning rate from a log-uniform distribution over $[10^{-3}, 10^{-2}]$, select the dropout rate uniformly from $[0, 0.5]$, and vary the batch size among 8, 16, or 32. Each configuration is evaluated on a held-out validation split to identify the best performing architecture and training settings.

A.6. Algorithms & Pseudocode

A.6.1. Greedy Algorithm based on Early Stopping Probability Predictions Pseudocode

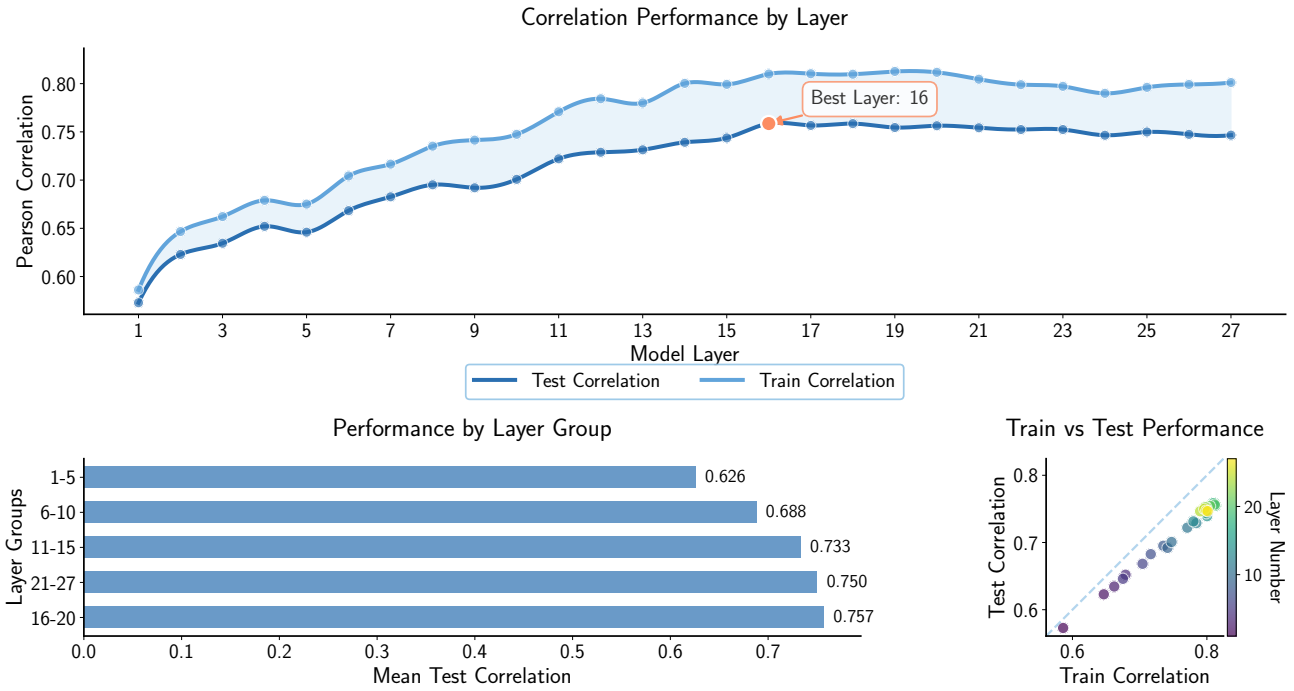

Figure 8: Correlation performance analysis across model layers. The top panel displays the Pearson correlation coefficients for both test and train datasets achieved by MLPs trained on different layer features of the DeepSeek-R1-Distill-Qwen-1.5B model. The middle layers (particularly layer 16) achieve the highest correlation for predicting early stopping performance, suggesting that intermediate representations offer the strongest signal for reasoning difficulty prediction. The bottom left panel shows aggregated performance by layer group. The bottom right panel illustrates the relationship between train and test correlation, with points colored by layer number.

Algorithm 1 Greedy Token Allocation using Predicted Early Stopping Correctness Probabilities

Require: Q, B, W, P {queries, budget, window size, probability vectors}
Ensure: allocations maximizing expected accuracy
0: allocations $\leftarrow [W] \times |Q|$
0: remaining $\leftarrow B \cdot |Q| - \sum$ allocations
0: **while** remaining $\geq W$ **do**
0: gains $\leftarrow$ ComputeGains(P, allocations, W)
0: **if** $\max(\text{gains}) \leq 0$ **then break**
0: **end if**
0: $i^* \leftarrow \arg\max(\text{gains})$
0: allocations[i^*] += W
0: remaining −= W
0: **end while**
0: **return** allocations =0

ComputeGains returns, for each query, the marginal expected-accuracy gain of adding one more window of size W (or $-\infty$ if no more windows remain).

A.6.2. Greedy Algorithm based on Difficulty Predictions Pseudocode

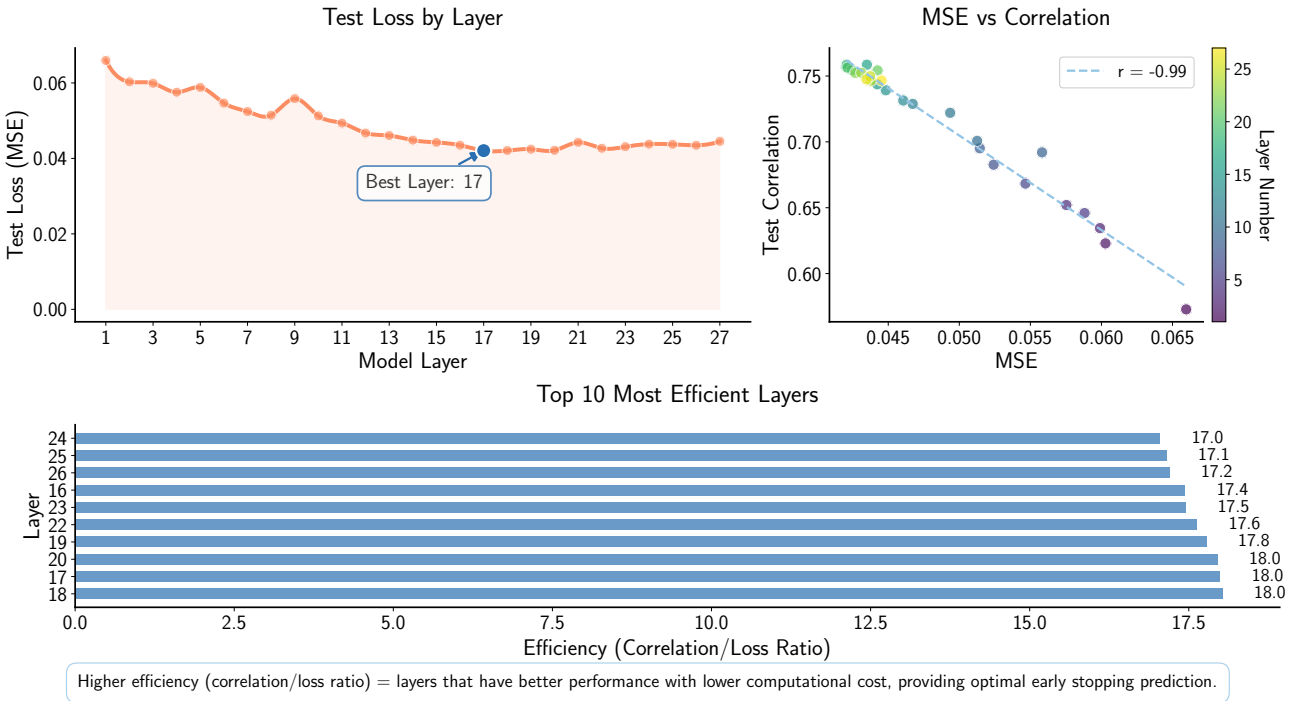

Figure 9: The top left panel shows test loss (MSE) by layer, demonstrating how prediction error varies across the model's depth. The top right panel illustrates the relationship between MSE and Pearson correlation, revealing that while these metrics are generally inversely related, some layers achieve better correlation despite similar loss values. The bottom panel presents correlation-to-loss ratio for the top performing layers, indicating which layers provide the most predictive value per unit of loss.

Algorithm 2 Difficulty-Based Token Budget Allocation

Require: Q, B, W, *difficulty_predictions*, p_1, p_2, p_3
Ensure: allocations per query maximizing expected accuracy
0: $(b_1, b_2, b_3) \leftarrow$ GETOPTIMALBUDGETS(B, p_1, p_2, p_3)
0: **for** $i = 1, \ldots, |Q|$ **do**
0: **if** difficulty_predictions[i]=`easy` **then**
0: allocations[i] $\leftarrow b_1$
0: **else if** difficulty_predictions[i]=`medium` **then**
0: allocations[i] $\leftarrow b_2$
0: **else**
0: allocations[i] $\leftarrow b_3$
0: **end if**
0: **end for**
0: **return** allocations =0

GETOPTIMALBUDGETS *exhaustively searches all triples in* $\{W, 2W, \ldots, 16W\}^3$ *and returns the one maximizing*

$$p_1 \operatorname{acc}_{\text{easy}}(b_1) + p_2 \operatorname{acc}_{\text{medium}}(b_2) + p_3 \operatorname{acc}_{\text{hard}}(b_3) \quad \textit{subject to} \quad p_1 b_1 + p_2 b_2 + p_3 b_3 \leq B.$$

A.7. Evaluation Details

A.7.1. ANSWER CORRECTNESS EVALUATION

For each GSM8K question we compare the model's predicted answer to the ground truth using a strict extraction and comparison procedure. First, we identify the final numerical answer by locating the last occurrence of text enclosed

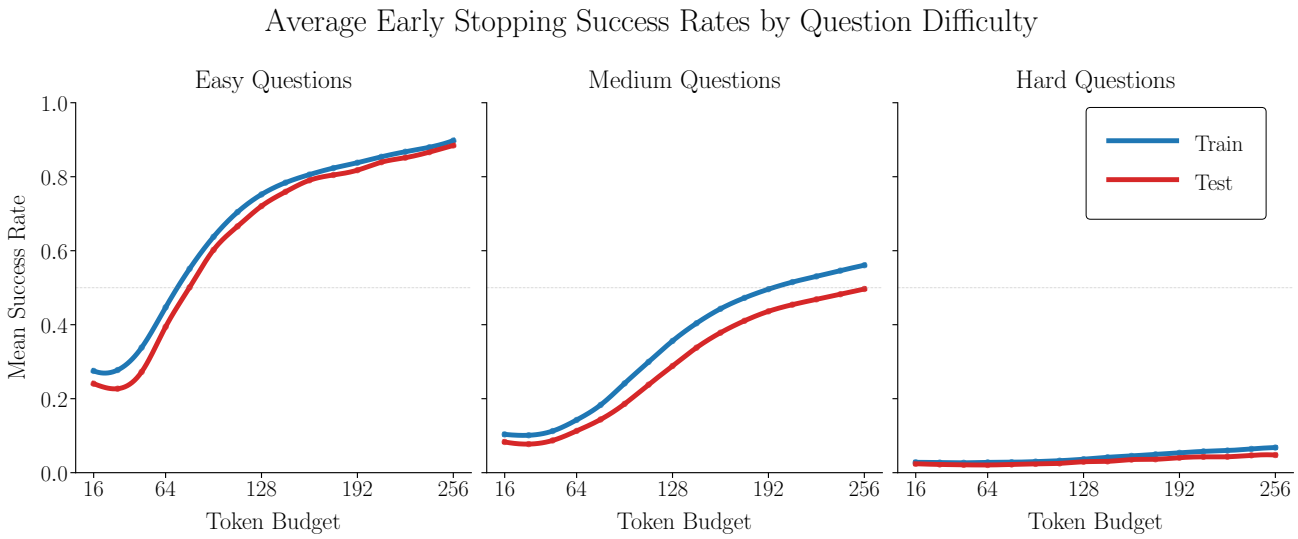

Figure 10: Early stopping probability vectors averaged across problems in each difficulty category (easy, medium, hard from left to right). The x-axis represents token budgets from 16 to 256 tokens in 16-token increments, while the y-axis shows the probability of generating a correct answer. Blue lines represent training set averages, and red lines represent test set averages. Note that the patterns are consistent between train and test sets within each difficulty category, validating our categorization approach.

in `\boxed{...}`. Next, we remove any non-numeric characters except for decimal points. The cleaned string is then converted to a floating-point value and compared to the ground truth answer under a relative tolerance of 10^{-4} to accommodate minor numerical discrepancies. If at any step the extraction or conversion fails—for example, if no `\boxed{...}` pattern is found—the answer is marked incorrect. This ensures that only exactly correct numerical outputs count as successes.

A.7.2. Few-Shot Classification of Question Difficulty Prompt

The following prompt is given to o4-mini-high in order to classify questions as easy, medium, or hard difficulty.

Prompt for o4-mini-high Difficulty Classification

```
You are an AI assistant tasked with categorizing math problems as easy,
medium, or hard.  You will be given some examples and then asked to
categorize a new question.  Here are some example questions with their
categorizations:

<examples>
"<User>Janet's ducks lay 16 eggs per day.  She eats three for breakfast
every morning and bakes muffins for her friends every day with four.
She sells the remainder at the farmers' market daily for $2 per fresh
duck egg.  How much in dollars does she make every day at the farmers'
market?<Assistant>", medium
"<User>A robe takes 2 bolts of blue fiber and half that much white fiber.
How many bolts in total does it take?<Assistant>", easy
"<User>James decides to run 3 sprints 3 times a week.  He runs 60 meters
each sprint.  How many total meters does he run a week?<Assistant>", hard
</examples>

Your task is to categorize the following new question as easy, medium, or
hard based on its similarity to the examples provided.  Assume that your
prior is that 20% of questions are easy, 60% of questions are medium, 20% of
questions are hard.  Be careful not to underestimate the difficulty of the
question you are categorizing--if it is possible to argue that it is hard,
classify it as hard, if it is possible to argue that it is medium, classify
it as medium.  If you want to say that the question is medium and not hard,
you should have a really strong justification for why it is medium but not
hard.
Here is the new question to categorize:
<new_question>
{Insert question here}
</new_question>
Please think about the complexity of the problem, the number of steps
required to solve it, and how it compares to the examples provided.  Then,
provide your categorization and reasoning in the following JSON format:

<output>
{ "reasoning":  "<your reasoning for the categorization>" "category":
"<category in {easy, medium, hard}>", }
</output>

Ensure that your reasoning is clear and concise, explaining why you chose
the specific category based on the question's characteristics and its
comparison to the provided examples.
```

A.8. Runtime and Hardware

All finetuning experiments were run on 1 H100 GPUs, and the API calls required for few-shot difficulty classification cost $< \$0.10$.