

Dynamic K-Head: Difficulty-Supervised Budget Selection for Continuous Latent Reasoning

Ying Li^{1,2}, Yuqi Chen^{3(✉)}, Sixuan Li⁴, and Yunfeng Cai^{1(✉)}

¹ Beijing Institute of Mathematical Sciences and Applications, Beijing, China

caiyunfeng@bimsa.cn

² Tsinghua University, Beijing, China

liying0928@tsinghua.edu.cn

³ The University of Hong Kong, Hong Kong, China

chenyuqi@hku.hk

⁴ Peking University, Beijing, China

sxli@pku.edu.cn

Abstract. Chain-of-thought (CoT) improves reasoning but incurs substantial autoregressive rationale cost. Continuous CoT moves intermediate computation into recurrent hidden states, yet typically uses a fixed number of updates even though the best budget varies across questions and accuracy need not improve monotonically with K . We introduce Dynamic K-Head, a question-conditioned, pre-rollout budget selector. It is jointly trained with the reasoner under selected budgets, using difficulty supervision based on scores assigned from the question alone. We also show that latent-update count can misrepresent end-to-end cost because a short latent rollout can shift computation into a longer explicit continuation. We therefore count latent updates, implementation-specific special tokens, and all decoded text. Instantiated with CoDI on Llama 3.2 1B and GSM8K, Dynamic K-Head reaches 54.44% exact-match accuracy at 12.03 average reasoning/output tokens, compared with 52.31% at 15.21 tokens for fixed- $K = 6$ CoDI—a 2.12-point gain with 20.9% lower token cost. Across three out-of-distribution arithmetic datasets, it reduces cost by 19.3–25.7% while improving accuracy by 0.30–2.22 points. These results demonstrate that difficulty-supervised budget selection can improve both accuracy and efficiency, while stable budget selection remains a central challenge.

Keywords: Latent reasoning · Continuous chain-of-thought · Adaptive computation · Efficient reasoning · Knowledge distillation

1 Introduction

Chain-of-thought (CoT) reasoning improves language-model performance by making intermediate computation explicit [1,2,3,4], but the resulting rationales can dominate decoding cost. Recent latent-reasoning methods instead feed hidden states back into the model as continuous thoughts. Coconut trains models to

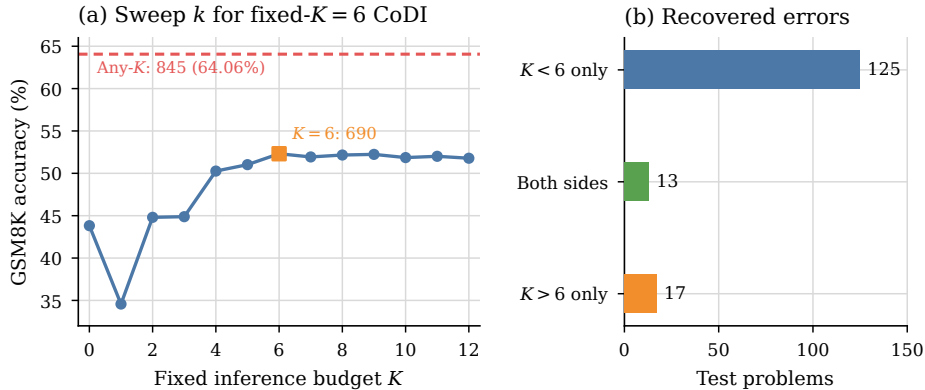


Fig. 1. Fixed-budget accuracy is non-monotonic and conceals instance-level recoveries. Panel (a) sweeps inference budget k for CoDI trained with fixed $K = 6$. Panel (b) partitions the 155 $K = 6$ errors recovered by another swept budget: 125 are recovered only below six, 17 only above six, and 13 on both sides. Thus the smaller- and larger-budget recovery totals are 138 and 30, respectively, with the 13 “both sides” problems included in each.

reason in a continuous latent space [5], while CoDI jointly trains explicit and implicit paths through self-distillation [6]. These methods reduce verbal output, yet most expose a fixed inference hyperparameter: the number K of latent updates.

Figure 1 illustrates why a single K is an awkward resource-allocation scheme. Easy questions may need little additional computation, whereas hard questions may benefit from more. More importantly, recurrent latent computation is not guaranteed to improve predictions monotonically. A later hidden state can overwrite a useful partial solution, amplify a shortcut, or move the final decoder into a less favorable region. With Llama 3.2 1B CoDI, $K = 6$ solves 690 of 1,319 GSM8K test problems. Sweeping $K = 0, \dots, 12$ with the same checkpoint shows that 845 problems are solved by at least one budget. Of the 629 questions missed at $K = 6$, 138 are recovered by some smaller budget and 30 by some larger budget; these recovery sets overlap. Thus adaptive budget selection is not merely a rule of “more steps for harder questions.” It requires an instance-conditioned choice among a non-monotonic family of computations.

Our Dynamic K-Head attaches a linear head to the pre-rollout question representation and selects the number of latent states actually executed, jointly exposing the K-head and reasoner to variable computation. The contrastive objective implementing difficulty supervision pulls together budget representations of questions with similar difficulty and separates dissimilar ones. Training begins with a fixed-budget warm-up and gradually transfers budget selection to the K-head; inference chooses its highest-scoring budget. The interface requires only a controllable recurrent rollout and is instantiated with CoDI in our experiments.

Relative to CoDI, Dynamic K-Head improves accuracy while reducing reasoning/output token cost. With $\lambda_K = 0.01$, it reaches 54.51% test accuracy at 13.33 average tokens, a 2.20-point accuracy gain with 12.4% fewer tokens than CoDI’s 52.31% accuracy at 15.21 tokens. With $\lambda_K = 0.03$, it reaches 54.44% accuracy at 12.03 tokens, retaining a 2.12-point gain while reducing token cost by 20.9%.

Our contributions are fourfold:

- We show that fixed latent budgets conceal non-monotonic instance-level variation: problems missed at one budget can be recovered by either smaller or larger budgets, motivating adaptive budget selection beyond a “more computation for harder questions” rule.
- We show that latent-token count alone can provide an imprecise estimate of end-to-end cost: a shorter latent rollout can shift reasoning into a longer explicit continuation. Our end-to-end accounting includes latent updates, implementation-specific special tokens, and all subsequently decoded text, and is applied uniformly to enable fair token-cost comparisons across different experimental settings.
- We introduce Dynamic K-Head, a pre-rollout, question-conditioned budget selector for recurrent latent reasoners, jointly trained under selected budgets with difficulty supervision, without direct budget labels or a monotonic difficulty-to-budget mapping.
- Dynamic K-Head improves CoDI by 2.12 accuracy points while reducing reasoning/output token cost by 20.9%; without difficulty supervision, budget selection is less accurate and more expensive.

2 Related Work

2.1 Explicit and Latent Reasoning

Explicit CoT exposes intermediate natural-language steps and has become a standard tool for mathematical reasoning [1,4]. Zero-shot CoT broadens its use without exemplars [2], while self-consistency samples multiple explicit traces and aggregates their answers [3]. These techniques improve reasoning but retain, or multiply, autoregressive rationale cost. Pause-token methods insert learned computation slots before an answer [7], while Quiet-STaR trains language models to generate internal rationales before predicting future text [8]. Both lines suggest that inference-time computation can be separated from user-visible output.

Continuous latent reasoning removes the requirement that every intermediate state correspond to a vocabulary item. Earlier implicit-CoT work distills explicit reasoning into layerwise hidden computation [9] or progressively removes rationale steps during fine-tuning [10]. SoftCoT instead uses a lightweight assistant to produce continuous soft thoughts for an unmodified target LLM [11]. Coconut recurrently feeds the last hidden state back as the next input embedding and observes search-like behavior on logical tasks [5]. CoDI uses one shared model as teacher and student: an explicit CoT path supplies hidden-state targets to a latent path, aligning the two at answer positions [6]. PCCoT applies Jacobi-style

parallel updates to latent tokens to improve training and inference throughput [12]. These approaches motivate compressed reasoning, but do not learn a pre-rollout, question-conditioned budget selector for recurrent latent computation.

Recent analyses suggest that latent computation is not a sequence of interchangeable depth increments: causal interventions reveal staged, non-local functionality across latent steps, while mechanistic probes find complete intermediate-state sequences on shorter-hop tasks but partial, late-intermediate pathways on longer-hop tasks [13,14]. This perspective helps explain our non-monotonic K sweeps and motivates adaptive budget selection rather than monotonic depth allocation.

2.2 Adaptive Computation

Adaptive computation has a long history in recurrent and universal architectures. Adaptive Computation Time learns a halting probability with a ponder cost [15]; PonderNet places a probabilistic model over computation depth [16]; and the Universal Transformer repeatedly refines token representations with optional dynamic halting [17]. Depth-Adaptive Transformer and CALM exit from intermediate Transformer layers at sequence or generation-token granularity [18,19], whereas Mixture-of-Depths selectively processes token positions at each layer under a fixed total capacity [20]. These methods establish the value of conditional compute, but adapt layer execution rather than selecting the number of recurrent continuous thoughts before answer decoding.

For latent language-model reasoning, Adaptive Latent Reasoning learns on-line stop/continue decisions through post-SFT reinforcement learning, whereas AdaAnchor applies a hand-designed stability rule while training anchors and LoRA modules around a frozen dense backbone [21,22]. Dynamic K-Head instead selects a latent budget for each question before rollout. Two distinctions are central. First, the K-head and LoRA-adapted reasoner are jointly trained under the budgets actually executed. Second, K is not an ordered refinement axis: under non-monotonic dynamics, a halting mechanism can only stop or append another update, and continuing may irreversibly drift from a useful state. We therefore treat budgets as distinct choices and apply difficulty supervision without optimal- K labels or a monotonic mapping.

2.3 Difficulty Supervision

Example difficulty has long been used to order training curricula [23], and model training dynamics can expose easy, ambiguous, and hard data regions [24]. Difficulty is not directly observed in standard reasoning datasets, however, and need not vary monotonically with the best latent budget. Teacher-model judgments can provide weak supervision, but may reflect the annotator’s priors rather than the student’s computation. We therefore avoid regressing K onto an assumed target. Instead, a contrastive objective uses only relative neighborhoods in the scalar difficulty scores. This design borrows the principle of pulling related examples together and separating unrelated ones from self-supervised and supervised

contrastive learning [25,26], but uses neither augmentation identity nor discrete class or budget labels.

3 Background and Problem Formulation

Standard CoT externalizes intermediate reasoning as a sequence of natural-language tokens before producing the answer, so every rationale token incurs an autoregressive decoding step. Latent CoT instead carries out this intermediate computation in continuous hidden-state space [5,6]. Given a question q , the model first prefills the question. At each latent update, it takes the hidden state at the current final position, converts it into the continuous input representation for the next position, and feeds that representation back into the model without selecting a vocabulary token. The resulting final-position hidden state is then used for the next update, forming a recurrent latent rollout.

We denote the number of latent updates by K . After exactly K updates, the model leaves the latent rollout and generates the answer y as ordinary vocabulary tokens with the standard autoregressive decoder, conditioned on both the question and the accumulated latent states. A fixed-budget latent reasoner uses the same number of updates K for every question. Let $\mathcal{K}$ denote the allowed budgets and let k index one candidate. We instead select

$$\hat{K}(q) = \arg \max_{k \in \mathcal{K}} p_\phi(k | q) \quad (1)$$

before latent reasoning, and perform exactly $\hat{K}(q)$ of the latent updates described above.

4 Dynamic K-Head

4.1 Pre-Rollout Budget Selection

Let h_q be the question representation extracted before latent reasoning. A K-head g_ϕ maps h_q to logits over an allowed budget set $\mathcal{K}$:

$$a = g_\phi(h_q), \quad p_\phi(k | q; \tau) = \text{softmax}(a/\tau)_k. \quad (2)$$

Here τ controls the sharpness of the budget distribution.

Training begins with an anchor-budget warm-up. Let ρ denote the warm-up fraction and $K_a \in \mathcal{K}$ the anchor budget. During the first ρ fraction of optimization, every example uses K_a and the K-head objective is disabled. After warm-up, we define an anchor-mixture prior

$$p_0(k) = (1 - \alpha) \frac{1}{|\mathcal{K}|} + \alpha \mathbf{1}[k = K_a], \quad (3)$$

where α controls the extra mass assigned to the anchor. Each example then independently draws its training-time executed budget according to

$$\begin{aligned} \hat{K}_i &\sim p_{\text{train}}(\cdot | q_i), \\ p_{\text{train}}(k | q_i) &= (1 - \epsilon) p_\phi(k | q_i; \tau) + \epsilon p_0(k). \end{aligned} \quad (4)$$

Over the post-warm-up portion of training, τ is annealed from τ_{start} to τ_{end} , while ϵ is annealed from ϵ_{start} to ϵ_{end} . The concrete K-head architecture, budget set, and schedule values are specified in Section 5.

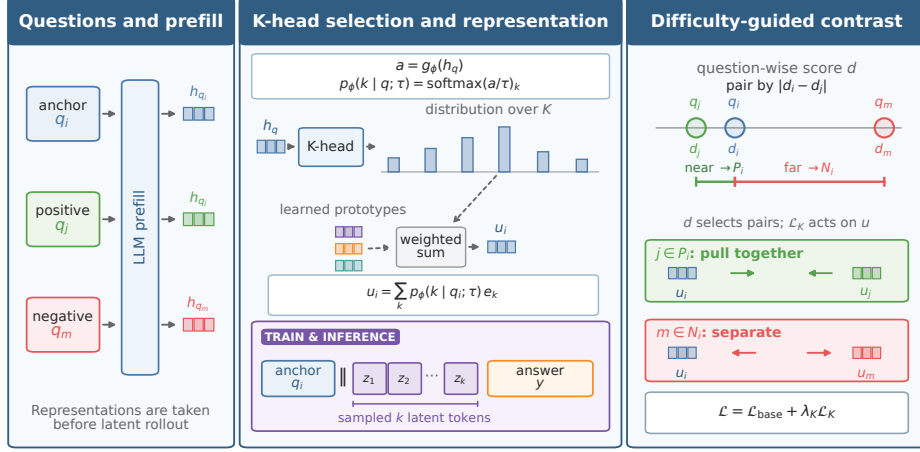


Fig. 2. Schematic of Dynamic K-Head training and execution. Anchor, positive, and negative questions are prefilled by the LLM, and the K-head produces a distribution over latent budgets. Learned prototypes map this distribution to u_i , while per-question difficulty scores make nearby questions positive pairs and distant questions negative pairs. A sampled K controls the illustrated shared train/inference path, where the anchor question is concatenated with exactly K latent tokens before answer decoding.

4.2 Difficulty Supervision

Each training question has a scalar difficulty score d_i assigned by a teacher given only the question. To obtain a differentiable representation of the K-head output, we learn one prototype e_k per candidate budget and form

$$u_i = \sum_{k \in \mathcal{K}} p_\phi(k | q_i; \tau) e_k. \quad (5)$$

Within a batch B , examples are ranked for anchor i by $|d_i - d_j|$. The n_+ nearest examples form the positive set P_i , and the n_- farthest form the negative set N_i . Let $A_i = P_i \cup N_i$ denote the comparison set. Adapting the standard supervised contrastive objective [26] to these difficulty-defined sets, with cosine similarity s and temperature T , we use

$$\mathcal{L}_K = -\frac{1}{|B|} \sum_{i \in B} \frac{1}{|P_i|} \sum_{p \in P_i} \log \frac{\exp(s(u_i, u_p)/T)}{\sum_{a \in A_i} \exp(s(u_i, u_a)/T)}. \quad (6)$$

The final objective is

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \lambda_K \mathcal{L}_K. \quad (7)$$

Here $\mathcal{L}_{\text{base}}$ is the original training objective of the corresponding fixed-budget latent reasoner. This difficulty supervision can therefore be added to different latent-reasoning methods without changing their method-specific base objectives. It encourages nearby difficulty scores to induce similar budget distributions without imposing a target K or a monotonic difficulty-to-budget mapping.

4.3 Token-Cost Accounting

Prior latent-reasoning studies, for example CoDI and Adaptive Latent Reasoning, measure token cost primarily by latent reasoning length, thereby using the number of latent tokens or updates as a proxy for cost [6,21]. This proxy can be imprecise because a model may continue reasoning in text after leaving latent mode: reducing K may merely shift computation from latent updates to explicit autoregressive decoding rather than reduce the total cost. Appendix B provides a real GSM8K test example. The model selects only one latent update, $\hat{K} = 1$, but then generates 28 text tokens containing three explicit calculation steps and the final answer; together with the two special tokens in this CoDI execution, its total cost is $1 + 2 + 28 = 31$ tokens. Thus, a smaller latent budget does not necessarily imply a lower end-to-end reasoning/output cost. We therefore count the latent computation, the special tokens required by the latent-reasoning implementation, and all subsequently generated text. For a question q executed with latent budget K , the cost is

$$C(q, K) = K + T_{\text{special}} + T_{\text{text}}(q, K), \quad (8)$$

where K counts the executed latent updates, T_{special} counts implementation-specific special tokens used for latent reasoning, such as BOT and EOT, and $T_{\text{text}}(q, K)$ counts every token decoded after the latent rollout, including any explicit reasoning and the final answer.

4.4 Deterministic Per-Example Inference

At inference, prior mixing and stochastic sampling are removed. The K-head selects

$$\hat{K}(q) = \arg \max_{k \in \mathcal{K}} p_\phi(k | q). \quad (9)$$

The reasoner then executes exactly $\hat{K}(q)$ latent updates before decoding the answer.

4.5 Active-Then-Repack Execution

Different examples in a batch may execute different budgets. The implementation maintains only rows that still require another latent update and removes a row from the active cache once it has reached its selected budget. It then concatenates

each question with exactly its executed latent states, repacks these variable-length prefixes, and processes them with explicit attention masks and position IDs before decoding the answer. This procedure avoids further latent updates for examples that have already reached their budgets, while allowing examples with heterogeneous budgets to proceed through answer decoding in a batch.

5 Experimental Setup

Data. We use `whynlp/gsm8k-aug` and `whynlp/gsm8k-aug-nl`, two aligned versions of GSM8K-Aug [10,12]. Each provides 385,620 training examples, 500 validation examples, and 1,319 test examples; the test split follows the official GSM8K test set [4]. For each run, we train on the corresponding training split, select the best checkpoint by validation exact-match accuracy, and report its performance on the test split.

For OOD evaluation, we use SVAMP [27], GSM-Hard [28], and MultiArith [29]. We concatenate the 700-example SVAMP training split and its 300-example test split, use all 1,319 GSM-Hard examples, and use the 180-example MultiArith test split. Dataset fields and representative examples are provided in Appendix A.

Difficulty scores. For the main experiments, we use difficulty scores assigned by the DeepSeek model family from the question alone [30], ranging from 1 (nearly one-step arithmetic) to 5 (unusually tricky). In the ablations, we also evaluate continuation negative log-likelihood (NLL) as an alternative signal for difficulty supervision. Other possible signals include the number of annotated reasoning steps and statistics of the arithmetic operators appearing in those steps. Each of these signals can define relative neighborhoods for the contrastive objective without specifying a target budget.

Instantiation. Dynamic K-Head is not tied to a particular latent-reasoning framework: it retains the framework’s original training objective as $\mathcal{L}_{\text{base}}$ and augments it with the difficulty-supervision loss $\mathcal{L}_K$ in Eq. 7. In the present experiments, we instantiate the method with CoDI [6], using GPT-2 [31] and Llama 3.2 1B Instruct [32] as base models. CoDI trains a shared model through an explicit-rationale path and a recurrent latent path: the explicit path supplies hidden-state targets, and the latent path learns to compress the reasoning trajectory through self-distillation. CoDI-specific settings otherwise follow the original method.

Dynamic K-Head instantiation. We implement g_ϕ as one linear layer with bias and use $\mathcal{K} = \{1, 2, 3, 4, 6, 8\}$. The warm-up fraction is $\rho = 0.1$, with anchor $K_a = 6$; the anchor-mixture weight is fixed at $\alpha = 0.3$. After warm-up, τ decreases linearly from 1.5 to 0.7, while ϵ decreases from 0.3 to 0.1. Difficulty pairs are constructed within each batch using $n_+ = 2$ and $n_- = 8$. Unless otherwise stated, we use $\lambda_K = 0.03$ and sweep its value in the later ablation (Section 6.4).

Implementation and hardware configuration. Both GPT-2 and Llama experiments use LoRA fine-tuning [33], and training and evaluation are implemented with PyTorch and Hugging Face Transformers [34,35]. All experiments run on NVIDIA H100 80GB HBM3 Tensor Core GPUs, with two GPUs allocated to each run. The eight-GPU server provides NVLink connectivity between every pair of GPUs.

6 Results

6.1 Overall Accuracy–Cost Results

Table 1 compares No-CoT SFT, CoT SFT, Coconut [5], CoDI, Adaptive Latent Reasoning [21], and Dynamic K-Head across GPT-2 and Llama 3.2 1B on GSM8K and GSM8K-NL. All Dynamic K-Head models use DeepSeek difficulty supervision. Relative to the fixed-budget CoDI baseline, Dynamic K-Head is consistently more accurate on Llama: accuracy increases from 52.31% to 54.44% on GSM8K and from 46.70% to 50.34% on GSM8K-NL. On GSM8K, this gain is accompanied by a 20.9% cost reduction, from 15.21 to 12.03 tokens. On GSM8K-NL, the higher accuracy costs 1.95 additional tokens. These results show that learned pre-rollout budget selection can improve the fixed-budget latent reasoner rather than merely shorten its rollout.

Among the latent methods evaluated here, Dynamic K-Head is also the closest to CoT SFT on Llama. On GSM8K, it is 7.88 accuracy points below CoT SFT while reducing average token cost from 33.96 to 12.03, a $2.82\times$ compression. On GSM8K-NL, the gap narrows to 2.73 points while cost falls from 73.95 to 17.17 tokens, a $4.31\times$ compression. Dynamic K-Head therefore recovers much of explicit CoT’s answer quality with substantially less autoregressive output. Figure 3 visualizes the same comparison in the accuracy–cost plane, combining both reasoning formats within each backbone panel. Dynamic K-Head contributes a point to the overall Pareto frontier for both backbones.

6.2 Out-of-Distribution Evaluation

We evaluate the Llama 3.2 1B models on the three arithmetic OOD datasets described in Section 5. Figure 4 compares Dynamic K-Head with CoT SFT, No-CoT SFT, and fixed- K CoDI checkpoints. Token cost follows Eq. 8.

Compared with the fixed- K CoDI baseline, Dynamic K-Head reduces average cost by 19.3–25.7% across the three datasets while improving accuracy by 0.30–2.22 points. This joint improvement indicates that instance-dependent budget selection transfers beyond the training distribution rather than merely shortening in-domain rollouts. Relative to CoT SFT, Dynamic K-Head reduces cost by 41.7–69.7%, corresponding to $1.71\times$ – $3.30\times$ compression. CoT SFT is 4.20 and 0.99 points more accurate on SVAMP and GSM-Hard, respectively; on MultiArith, Dynamic K-Head is 0.56 points more accurate while using half the cost.

Dynamic K-Head remains highly accurate on MultiArith but reaches only 13.12% on GSM-Hard. The budget selector averages 2.15–2.74 latent updates

Table 1. Accuracy and average token cost on the official GSM8K test set across two backbones and the GSM8K and GSM8K-NL training variants. AdaLR denotes Adaptive Latent Reasoning. All Dynamic K-Head rows use DeepSeek difficulty supervision. Token cost follows Eq. 8. Within each column, bold and underlined values indicate the best and second-best results, respectively; CoT SFT is excluded from the accuracy ranking, and No-CoT SFT is excluded from the cost ranking.

Method	GSM8K		GSM8K-NL	
	Accuracy (%)	Avg. token cost	Accuracy (%)	Avg. token cost
GPT-2				
No-CoT SFT	14.25	6.34	14.25	6.34
CoT SFT	48.90	32.27	33.59	67.60
Coconut	33.13	11.21	—	—
CoDI	39.80	14.24	37.38	<u>14.26</u>
Dynamic K-Head	<u>37.68</u>	<u>11.56</u>	<u>32.75</u>	12.57
Llama 3.2 1B				
No-CoT SFT	27.82	6.21	27.82	6.21
CoT SFT	62.32	33.96	53.07	73.95
Coconut	35.71	12.29	—	—
CoDI	<u>52.31</u>	15.21	<u>46.70</u>	15.22
AdaLR	48.14	11.09	—	—
Dynamic K-Head	54.44	<u>12.03</u>	50.34	<u>17.17</u>

across the three datasets and never selects $K = 8$, suggesting that the remaining errors reflect limited coverage of the shifted arithmetic demands rather than exhaustion of the available budget.

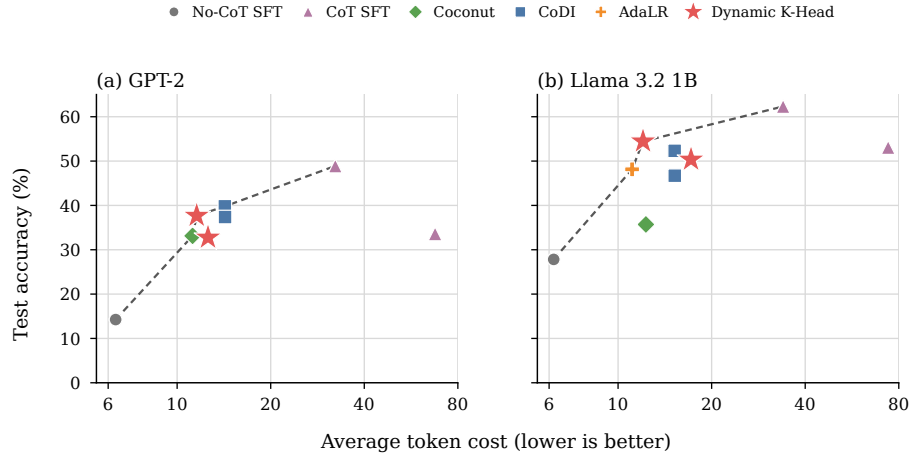


Fig. 3. Accuracy versus average token cost for the Table 1 results, with GPT-2 on the left and Llama 3.2 1B on the right. GSM8K and GSM8K-NL are overlaid using the same method-specific markers, and the horizontal axis uses a logarithmic scale. Dashed lines indicate the Pareto frontiers.

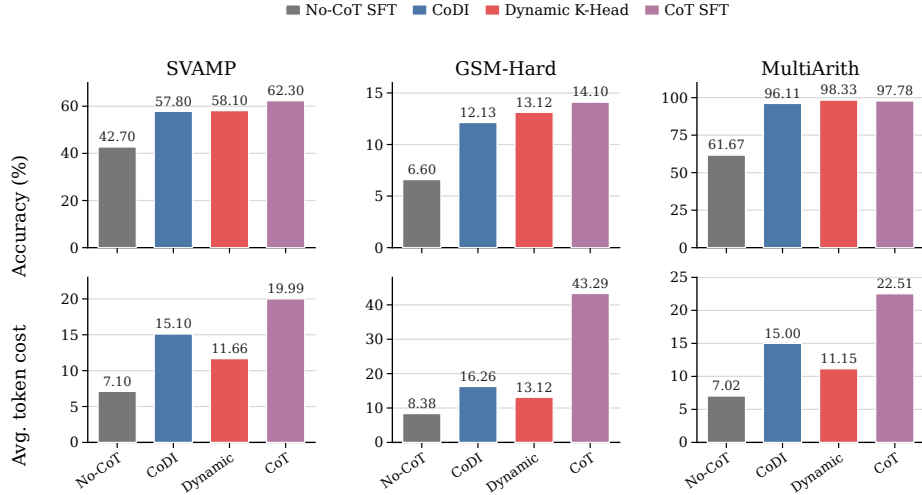


Fig. 4. Accuracy (top) and average token cost (bottom) on three OOD datasets. Within each column, methods are ordered as No-CoT SFT, CoDI ($K = 6$), Dynamic K-Head, and CoT SFT. Each panel uses a dataset-specific vertical scale.

6.3 Online vs. External Difficulty Supervision

The key question is whether budget selection benefits more from the student’s evolving notion of difficulty or from a fixed external ordering. Table 2 compares fixed-budget CoDI with Dynamic K-Head trained using continuation NLL or DeepSeek difficulty scores computed from questions alone.

On Llama, DeepSeek difficulty supervision reaches 54.44% at 12.03 tokens and improves both metrics over CoDI and continuation NLL. On GPT-2, it lowers cost to 11.56 tokens but also lowers accuracy to 37.68%. This weaker result may partly reflect GPT-2’s substantially smaller capacity, which can limit both the quality of h_q for budget selection and the reasoner’s ability to preserve accuracy under shorter computation.

The signals also differ in how they evolve. Continuation NLL is an online, student-dependent signal recomputed with the current model at every training epoch; it therefore tracks the student’s changing uncertainty. DeepSeek difficulty is a fixed, external question-level signal that provides a stable ordering throughout training. Since $\mathcal{L}_K$ uses these values to construct relative neighborhoods rather than target budgets, the choice between an adaptive internal signal and a stable external signal changes the representations learned by the K-head. The results indicate that neither notion of difficulty defines a universal compute target: its value depends on the capacity and representations of the student model.

6.4 Difficulty Supervision Controls Budget Selection

Table 3 reveals distinct patterns of budget selection. At $\lambda_K = 0$, $H(K | Q) = 1.715$ nearly equals $H(K) = 1.781$ and $I(Q; K) = 0.066$, indicating diffuse, nearly question-independent budget distributions.

Moderate difficulty supervision produces decisive, question-dependent budget selection. Although $\lambda_K = 0.01$ is most accurate (54.51%), 0.03 nearly matches it (54.44%) at lower cost (12.03 versus 13.33 tokens) and pairs low predictive entropy (0.097) with the highest mutual information (0.443), giving the strongest observed balance.

Higher weights separate budget concentration from utility. At 0.05, $H(K | Q) = 0.006$ and $H(K) = 0.122$ indicate near-global concentration; its 52.62% accuracy at the lowest cost (10.27 tokens) makes it cheaper but less adaptive. At 0.10, mutual information is 0.424 but accuracy falls to 49.05%.

The low-score train–evaluation shift (≤ 2 : 91.3% versus 71.8%/71.9%; Appendix C) may contribute to high-weight concentration.

Table 2. Fixed-budget CoDI and Dynamic K-Head with two signals for difficulty supervision on GSM8K. NLL denotes online continuation NLL recomputed at each training epoch; difficulty denotes fixed DeepSeek scores computed from questions alone. Within each backbone block, bold and underlined values indicate the best and second-best results, respectively.

Method	Accuracy (%)	Avg. token cost
GPT-2		
CoDI	39.80	<u>14.24</u>
Dynamic K-Head (NLL)	<u>38.29</u>	14.37
Dynamic K-Head (difficulty)	37.68	11.56
Llama 3.2 1B		
CoDI	52.31	15.21
Dynamic K-Head (NLL)	<u>53.15</u>	<u>14.65</u>
Dynamic K-Head (difficulty)	54.44	12.03

Table 3. Accuracy, average token cost, marginal entropy $H(K)$, predictive entropy $H(K | Q)$, and question–budget mutual information $I(Q; K)$ for the loss-weight ablation on GSM8K. All information quantities are measured in nats.

λ_K	Accuracy (%)	Avg. cost	$H(K)$	$H(K Q)$	$I(Q; K)$
0.00	50.87	13.11	1.781	1.715	0.066
0.01	54.51	13.33	0.441	0.131	0.310
0.03	54.44	12.03	0.540	0.097	0.443
0.05	52.62	10.27	0.122	0.006	0.115
0.10	49.05	10.89	0.724	0.300	0.424

7 Limitations and Broader Considerations

Small models. Our experiments are limited to GPT-2 and Llama, so whether the findings generalize to other model families and larger scales remains unknown.

External difficulty scores. Teacher-assigned scores may contain systematic biases and are not guaranteed to accurately reflect the task difficulty experienced by the student model. Future work should compare multiple scoring models.

Weak supervision. The contrastive objective provides only weak, question-level signals for budget selection. Because the K-head reads a representation produced by the shared backbone, it may indirectly inherit some outcome-related information while the backbone is optimized by $\mathcal{L}_{\text{base}}$ under sampled budgets. This pathway nevertheless provides no explicit per-example credit assignment identifying which candidate K values would yield a correct answer, nor counterfactual evidence for budgets that were not executed. The learned K-head representations may therefore align with difficulty-score neighborhoods without matching the reasoner’s actual budget-specific success pattern. Stronger supervision could use outcomes collected across multiple budgets for each question, at the cost of additional rollouts.

8 Conclusion

We introduced Dynamic K-Head, a pre-rollout budget selector that treats latent budgets as non-monotonic, instance-conditioned choices and learns them with difficulty supervision. We also proposed end-to-end cost accounting that includes latent updates, special tokens, and decoded text. With CoDI on Llama 3.2 1B, Dynamic K-Head reaches 54.44% accuracy on GSM8K at 12.03 average tokens, improving fixed- $K = 6$ CoDI by 2.12 points while reducing cost by 20.9%, and it retains joint accuracy-cost gains on three out-of-distribution datasets. Budget-selection quality depends on the granularity and balance of difficulty coverage and on how directly supervision reflects budget utility. Future work should therefore combine finer-grained, better-balanced training data with more direct outcome-based supervision.

Code, Data, and Model Availability. We will open-source the code and publicly release the difficulty-annotated datasets and model weights. These resources are currently being organized and prepared for release.

Declaration of Generative AI Use. During manuscript preparation, the authors used OpenAI GPT-5.6-sol solely to improve readability, clarity, grammar, and style. The authors reviewed and approved all edits and take responsibility for the final content.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems. vol. 35, pp. 24824–24837 (2022)
2. Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. In: Advances in Neural Information Processing Systems. vol. 35 (2022)
3. Wang, X., Wei, J., Schuurmans, D., Le, Q.V., Chi, E.H., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. In: International Conference on Learning Representations (2023)
4. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., Schulman, J.: Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168 (2021)
5. Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., Tian, Y.: Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769 (2024)
6. Shen, Z., Yan, H., Zhang, L., Hu, Z., Du, Y., He, Y.: CODI: Compressing chain-of-thought into continuous space via self-distillation. arXiv preprint arXiv:2502.21074 (2025)

7. Goyal, S., Ji, Z., Rawat, A.S., Menon, A.K., Kumar, S., Nagarajan, V.: Think before you speak: Training language models with pause tokens. In: International Conference on Learning Representations (2024)
8. Zelikman, E., Harik, G., Shao, Y., Jayasiri, V., Haber, N., Goodman, N.D.: Quiet-STaR: Language models can teach themselves to think before speaking. arXiv preprint arXiv:2403.09629 (2024)
9. Deng, Y., Prasad, K., Fernandez, R., Smolensky, P., Chaudhary, V., Shieber, S.: Implicit chain of thought reasoning via knowledge distillation. arXiv preprint arXiv:2311.01460 (2023)
10. Deng, Y., Choi, Y., Shieber, S.: From explicit CoT to implicit CoT: Learning to internalize CoT step by step. arXiv preprint arXiv:2405.14838 (2024)
11. Xu, Y., Guo, X., Zeng, Z., Miao, C.: SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. arXiv preprint arXiv:2502.12134 (2025)
12. Wu, H., Teng, Z., Tu, K.: Parallel continuous chain-of-thought with jacobi iteration. arXiv preprint arXiv:2506.18582 (2025)
13. Li, Z., Bai, X., Chen, K., Li, Y., Yang, J., Lin, C., Zhang, M.: Dynamics within latent chain-of-thought: An empirical study of causal structure. arXiv preprint arXiv:2602.08783 (2026)
14. Liang, J., Pan, L.: Do latent-CoT models think step-by-step? a mechanistic study on sequential reasoning tasks. arXiv preprint arXiv:2602.00449 (2026)
15. Graves, A.: Adaptive computation time for recurrent neural networks. arXiv preprint arXiv:1603.08983 (2016)
16. Banino, A., Balaguer, J., Blundell, C.: PonderNet: Learning to ponder. In: 8th ICML Workshop on Automated Machine Learning (2021)
17. Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., Kaiser, L.: Universal transformers. In: International Conference on Learning Representations (2019)
18. Elbayad, M., Gu, J., Grave, E., Auli, M.: Depth-adaptive transformer. In: International Conference on Learning Representations (2020)
19. Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V.Q., Tay, Y., Metzler, D.: Confident adaptive language modeling. In: Advances in Neural Information Processing Systems. vol. 35 (2022)
20. Raposo, D., Ritter, S., Richards, B., Lillicrap, T., Humphreys, P.C., Santoro, A.: Mixture-of-depths: Dynamically allocating compute in transformer-based language models. arXiv preprint arXiv:2404.02258 (2024)
21. Ning, A., Kuo, Y.L., Gomes, G.: Learning when to stop: Adaptive latent reasoning via reinforcement learning. arXiv preprint arXiv:2511.21581 (2025)
22. Sheshanarayana, D., Pal, R.S., Sinha, M., Dasgupta, T.: Thinking in latents: Adaptive anchor refinement for implicit reasoning in LLMs. arXiv preprint arXiv:2603.15051 (2026)
23. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th International Conference on Machine Learning. pp. 41–48. ACM (2009). <https://doi.org/10.1145/1553374.1553380>
24. Swayamdipta, S., Schwartz, R., Lourie, N., Wang, Y., Hajishirzi, H., Smith, N.A., Choi, Y.: Dataset cartography: Mapping and diagnosing datasets with training dynamics. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. pp. 9275–9293 (2020). <https://doi.org/10.18653/v1/2020.emnlp-main.746>
25. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: Proceedings of the 37th International Conference on Machine Learning. pp. 1597–1607 (2020)

26. Khosla, P., Teterwak, P., Wang, C., et al.: Supervised contrastive learning. In: *Advances in Neural Information Processing Systems*. vol. 33 (2020)
27. Patel, A., Bhattamishra, S., Goyal, N.: Are NLP models really able to solve simple math word problems? In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pp. 2080–2094 (2021). <https://doi.org/10.18653/v1/2021.naacl-main.168>
28. Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., Neubig, G.: PAL: Program-aided language models. In: *Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research*, vol. 202, pp. 10764–10799. PMLR (2023)
29. Roy, S., Roth, D.: Solving general arithmetic word problems. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pp. 1743–1752 (2015). <https://doi.org/10.18653/v1/D15-1202>
30. DeepSeek-AI: DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437 (2024)
31. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. Tech. rep., OpenAI (2019)
32. Meta: Llama 3.2 1B Instruct model card. <https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct> (2024)
33. Hu, E.J., Shen, Y., Wallis, P., et al.: LoRA: Low-rank adaptation of large language models. In: *International Conference on Learning Representations* (2022)
34. Paszke, A., Gross, S., Massa, F., et al.: PyTorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems*. vol. 32, pp. 8024–8035 (2019)
35. Wolf, T., Debut, L., Sanh, V., et al.: Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. pp. 38–45 (2020). <https://doi.org/10.18653/v1/2020.emnlp-demos.6>

A Dataset Formats and Examples

A.1 GSM8K-Aug Training Data

Both `whynlp/gsm8k-aug` and `whynlp/gsm8k-aug-nl` store each record with three fields: `question` is a string, `steps` is a list of strings, and `answer` is a string. The datasets align questions and answers but serialize `steps` differently. In `whynlp/gsm8k-aug`, each step is a compact calculator-style expression enclosed by double angle brackets. In `whynlp/gsm8k-aug-nl`, each step is a natural-language sentence.

Aligned example. The following record occurs in both formats.

Question: Out of 600 employees in a company, 30% got promoted while 10% received a bonus. How many employees did not get either a promotion or a bonus?

whynlp/gsm8k-aug steps:

1. <<600*30/100=180>>
2. <<600*10/100=60>>
3. <<180+60=240>>
4. <<600-240=360>>

whynlp/gsm8k-aug-nl steps:

1. $600 \times 30/100 = 180$ employees were promoted.
2. $600 \times 10/100 = 60$ employees received a bonus.
3. A total of $180 + 60 = 240$ employees received a promotion or a bonus.
4. Therefore, $600 - 240 = 360$ employees did not get either a promotion or a bonus.

Answer: 360

A.2 OOD Evaluation Data

SVAMP. We concatenate the 700-example training split and the 300-example test split from `ChilleD/SVAMP`; the model input is `question_concat`, and the gold answer is `Answer`.

GSM-Hard. We use all 1,319 examples from `juyoung-trl/gsm-hard`, with `instruction` as the input and `response` as the answer.

MultiArith. We use the 180-example test split from `ChilleD/MultiArith`, with fields `question` and `final_ans`. Representative records from the evaluated splits follow; surrounding whitespace is omitted.

SVAMP example (test split).

Question (question_concat): There were 106 dollars in Olivia’s wallet. After she visited a supermarket and a showroom there were 26 dollars left. If she spent 49 dollars at the showroom How much did she spend at the supermarket?

Answer (Answer): 31

GSM-Hard example.

Question (instruction): A robe takes 2287720 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?
Answer (response): 3431580.0

MultiArith example (test split).

Question (question): Paige had 11 songs on her mp3 player. If she deleted 9 old songs from it and then added 8 new songs, how many songs does she have on her mp3 player?
Answer (final_ans): 10

B A Small Latent Budget Can Still Produce a Long Continuation

The following is a real, correctly answered example from an official GSM8K test evaluation of a GPT-2 Dynamic K-Head model. The selector chose only one latent update, $\hat{K} = 1$, but the model then continued the computation in explicit text.

Question: Tom plants 10 trees a year. Every year he also chops down 2 trees a year. He starts with 50 trees. After 10 years 30% of the trees die. How many trees does he have left?
Selected latent budget: $\hat{K} = 1$
Decoded continuation:
 8*10=80>> <<50+80=130>>
 <<130*30/100=39>>The answer is: 91
Gold answer: 91

The continuation contains 28 generated tokens, including three explicit calculation steps and the final answer. Counting the BOT and EOT tokens used by this CoDI execution, the example costs $1 + 2 + 28 = 31$ tokens despite using only one latent update. This illustrates why K alone is not a reliable proxy for end-to-end reasoning/output cost.

C Difficulty-Score Distributions

The DeepSeek scores, computed from questions alone, are more concentrated in the augmented training set than in the evaluation sets. Scores at most 2 account for 91.3% of training questions, compared with 71.8% of validation and 71.9% of test questions. Validation and test have closely matched profiles, while both place more mass than training at scores 2.5 and 3.5.

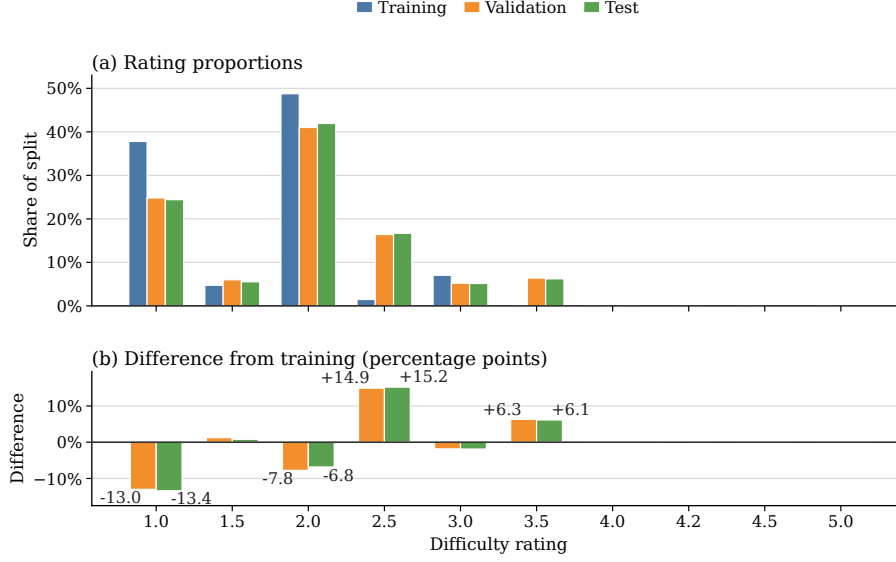


Fig. 5. DeepSeek difficulty scores, computed from questions alone, across training, validation, and test. The lower panel reports validation-minus-training and test-minus-training differences in percentage points.

D Information Quantities for Budget Selection

Let Q be uniformly distributed over the $N = 1,319$ GSM8K test questions, and let $p_i(k) = p_\phi(k \mid q_i; \tau = 1)$ be the soft K-head distribution for question q_i . We compute

$$\begin{aligned}
 H(K \mid Q) &= \frac{1}{N} \sum_{i=1}^N - \sum_{k \in \mathcal{K}} p_i(k) \ln p_i(k), \\
 \bar{p}(k) &= \frac{1}{N} \sum_{i=1}^N p_i(k), \\
 H(K) &= - \sum_{k \in \mathcal{K}} \bar{p}(k) \ln \bar{p}(k), \\
 I(Q; K) &= H(K) - H(K \mid Q).
 \end{aligned}$$

Here $H(K \mid Q)$ is the mean per-question predictive entropy, $H(K)$ is the entropy of the aggregate soft-budget distribution, and $I(Q; K)$ measures how strongly the soft budget distribution varies across questions. All quantities use natural logarithms and are reported in nats.