THINK-AT-HARD: SELECTIVE LATENT ITERATIONS TO IMPROVE REASONING LANGUAGE MODELS

Tianyu Fu[*1,2], **Yichen You**[*1], **Zekai Chen**[1], **Guohao Dai**[3,2], **Huazhong Yang**[1], **Yu Wang**[† 1]

[1]Tsinghua University [2]Infinigence AI [3]Shanghai Jiao Tong University

ABSTRACT

Improving reasoning abilities of Large Language Models (LLMs), especially under parameter constraints, is crucial for real-world applications. Looped transformers address this by performing multiple latent iterations to refine each token beyond a single forward pass. However, we identify a *latent overthinking* phenomenon: most token predictions are already correct after the first pass, but are sometimes revised into errors in later iterations. In this work, we ask whether *selectively skipping* latent iterations may *improve accuracy*. We reveal significant potential with an oracle iteration policy that boosts model performance by up to 7.3%. Motivated by this, we propose Think-at-Hard (TaH), a looped transformer optimized for selective iteration. TaH employs a lightweight neural decider to trigger latent iteration only at tokens that are likely incorrect after the standard forward pass. During latent iterations, depth-aware Low-Rank Adaptation (LoRA) modules shift the LLM's objective from general next-token prediction to focused hard-token refinement. A duo-causal attention mechanism extends attention from the token sequence dimension to an additional iteration depth dimension, enabling cross-iteration information flow with full sequential parallelism. Experiments on nine benchmarks show consistent gains across math, QA, and coding tasks. With identical parameter counts, TaH outperforms always-iterate baselines by 3.8–4.4% while skipping iterations on 93% of tokens, and exceeds single-iteration Qwen3 baselines by 3.0–3.8%. When allowing $< 3\%$ more parameters from LoRA and decider modules, the gains further increase to 5.3–6.2% and 6.1–6.8%, respectively.

1 INTRODUCTION

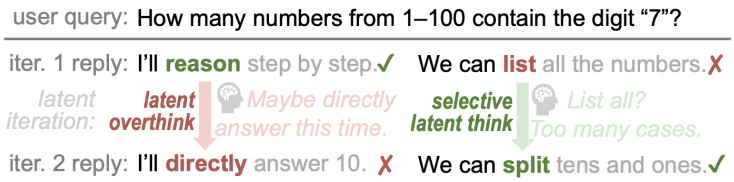


Figure 1: Latent iterations can fix wrong predictions, but can also *overthink* and flip correct ones. *Selective* iteration only when needed can improve reasoning with reduced computation.

Recent advances in Large Language Model (LLM) reasoning have enabled broad applications across diverse domains (Jaech et al., 2024; Guo et al., 2025; Yang et al., 2025). With hundreds of billions of parameters, LLMs can generate complex Chain-of-Thought (CoT) to solve challenging tasks. At the same time, smaller language models have also drawn increasing attention. With only a few billion parameters, they offer compelling alternatives: lower costs, faster inference, and suitability for edge computing (Abdin et al., 2024; Team et al., 2025; Wang et al., 2025a).

[*]Equal contribution.

[†]Corresponding author: Yu Wang (yu-wang@tsinghua.edu.cn).

Table 1: Performance comparison across iteration strategies, using Ouro-1.7B model except for the last row. Always1 means no latent iteration. Oracle policy iterates on 12–19% tokens. MMLU100 denotes the first 100 questions from MMLU-STEM.

Iter. Policy	NTP	AMC23	MMLU100	HE++
Always1	73.1	38.1	56.0	39.6
Always2	79.7	40.6	60.0	40.9
Orcl.	81.8/ + 2.1	47.9/ + 7.3	62.0/ + 2.0	43.3/ + 2.4
Orcl. w.TaH	89.3/ + 9.6	68.8/ + 28.2	85.0/ + 25.0	72.9/ + 32.0

At this crossroads, enhancing reasoning capabilities under parameter constraints becomes a central challenge. A common approach is to distill smaller models to mimic LLM CoT trajectories. However, not all tokens are equally predictable: certain tokens encode critical logic or reasoning directions that are fundamentally harder to predict (Lin et al., 2024; Fu et al., 2025; Wang et al., 2025b). With limited computation per output token, small models quickly hit a performance ceiling and mispredict some of these tokens. Once critical errors occur, the reasoning trajectory can irrecoverably diverge and produce drastically different outcomes.

Prior work proposes looped transformers to address this parameter–performance paradox (Hutchins et al., 2022; Saunshi et al., 2025; Zeng et al., 2025; Zhu et al., 2025). Instead of verbalizing the next token immediately after one forward pass, these models typically feed the last-layer hidden states back into the LLM for additional passes, refining representations in the latent space. After certain iterations, the final hidden states pass through the language modeling head to generate the next token. By uniformly scaling up iterations per token, these models can correct initially wrong token predictions, potentially increasing performance without increasing parameter count.

However, we identify a *latent overthinking* problem in looped transformers, where excessive iterations revise correct answers into wrong ones. As shown in Figure 3, while the second iteration corrects 8.7% of predictions, it also flips 2.1% of correct ones into errors. This occurs because most tokens, such as coherence or suffix tokens, are already predicted correctly after the first pass; further iterations may instead introduce harmful changes. This mirrors overthinking in explicit CoT reasoning (Wu et al., 2025), where additional reasoning steps degrade rather than improve answers. This reveals a surprising opportunity:

selectively skipping latent iterations on most tokens can further increase model performance.

We validate this with an oracle policy that iterates only on initially mispredicted tokens, as shown in Table 1. Compared to always iterating, this selective oracle can achieve up to 32% higher accuracy with optimized architecture.

Achieving selective latent iteration presents three main challenges. First, the model architecture should enable cross-depth attention, allowing each iteration to access full context. This is crucial because when early tokens skip deeper iterations, later tokens must still access their representations from shallower depths. Meanwhile, this cross-depth flow cannot compromise the sequence-level parallelism essential for efficient training and prefilling. Second, the model must adapt to distribution shifts across iterations, while maximizing parameter reuse. Third, training must remain stable despite tightly coupled dependencies: the iteration policy depends on prediction quality at each depth, while that quality depends on the depths to which the policy assigns previous tokens.

To address these challenges, we propose TaH, a looped transformer optimized for selective latent iteration. As shown in Figure 2, TaH employs a neural decider to determine whether to iterate or verbalize each token. We design a duo-causal attention mechanism to enable cross-depth attention with full sequence parallelism. To specialize deeper iterations for current-token refinement and preserve strong first-pass predictions, we apply LoRA adapters solely at iterations $d > 1$. We train TaH stably by aligning both the LLM backbone and iteration decider with a static oracle iteration policy. We summarize our contributions as follows.

- **Selective Latent Iteration**. We identify the latent overthinking phenomenon and quantify its influence on token prediction accuracy and downstream tasks. This insight motivates

new directions where latent iteration is applied selectively to a few tokens for both better reasoning quality and efficiency.

- **Specialized Model Architecture**. We develop a model architecture that natively supports selective iterations. The dedicated duo-causal attention mechanism, LoRA adapters, and iteration decider enable efficient cross-depth information flow, objective transitions, and dynamic depth selection.
- **Stable Training**. We introduce a stable training scheme that uses a static oracle policy to decouple model adaptation and policy learning. It overcomes the circular dependency between iteration decisions and prediction quality.

We fine-tune TaH from Qwen3-Base 0.6B and 1.7B on Open-R1, then tested across nine reasoning benchmarks spanning math, QA, and coding tasks. TaH achieves average accuracy gains of 3.0% and 3.8% over standard single-iteration baselines. Compared to Ouro looped-transformer trained on the same data, TaH achieves 3.8–4.4% gains while reducing latent iterations by 93%.

2 Related Work

Unlike standard LLMs that verbalize at every autoregressive step, latent thinking shifts part of generation away from explicit natural-language CoT in order to improve reasoning (Li et al., 2025). We provide additional discussions on signal-guided control and latent optimization in Appendix A.5.

Looped Transformers. These methods interleave latent and verbal reasoning, adding latent iterations before each token verbalization. Previous work focuses on scaling up iteration depths, with the main architectural focus on next-iteration inputs: reusing hidden states directly (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025) or using logit-weighted embeddings (Zeng et al., 2025). Looped transformers achieve deeper computation without parameter increases. However, uniform depth scaling burdens training and inference, and risks overthinking already-correct tokens.

Positioning. TaH belongs to the looped transformer family, but identifies *selective iteration* as a new design principle to improve performance. While concurrent works (Bae et al., 2025; Zhu et al., 2025) also enable dynamic recursion, they degrade performance at non-maximum depths and require full retraining. TaH instead leverages existing pre-trained models, adding depth-aware LoRA and duo-causal attention to improve reasoning with minimal fine-tuning overhead.

3 Preliminaries

Autoregressive LLMs. Modern LLMs generate text through an autoregressive next-token prediction process. It includes a *prefill* stage and a *decode* stage (Radford et al., 2018; 2019; Kwon et al., 2023). In the prefill stage, the model processes the entire input sequence in parallel; in the decode stage, it consumes one new token at a time along with cached history to predict the next token.

Formally, let t_i denote the token at position i and $x_i \in \mathbb{R}^h$ its embedding. Let $E \in \mathbb{R}^{v\times h}$ be the embedding matrix, so $x_i = E[t_i]$ when t_i is treated as an index. Here, v and h are the vocabulary size and hidden dimension. The output projection matrix is $W_{\text{out}} \in \mathbb{R}^{h\times v}$ (equal to $E^\top$ if tied). Given the context $T_{\le i} = [t_0, \dots, t_i]$ with embeddings $X_{\le i} = [x_0, \dots, x_i]$, the model θ produces a *last-layer hidden state* y_i for token t_i:

$$y_i \;=\; \mathcal{P}_\theta\big(x_i \mid X_{\le i}\big) \;\in\; \mathbb{R}^h. \tag{1}$$

The next-token distribution p_i and sample are:

$$p_i \;=\; \mathrm{softmax}\big(W_{\text{out}}^\top y_i\big) \in \mathbb{R}^v, \qquad t_{i+1} \;=\; \mathcal{S}(p_i), \tag{2}$$

where $\mathcal{S}$ is a sampling rule such as nucleus sampling. Decoding repeats until an end-of-sequence token is generated.

Causal Attention. Modern LLMs typically adopt *causal* attention mechanism. As shown in Figure 2(a), each position attends only to itself and earlier positions, consistent with Equation 1. This design brings two key benefits: (1) it enables parallel training with next-token prediction and shifted logits, avoiding the need for token-by-token generation; and (2) it allows efficient inference by caching Key/Value states of past tokens instead of recomputing them.

Looped Transformers. Looped transformers introduce an inner loop that iterates in latent space before verbalizing each output token. Let $d \in \{1, 2, \dots\}$ denote the iteration depth (written as a superscript), and set $x_i^{(0)} = E[t_i]$. At each iteration, looped transformers update y_i with causal attention on the hidden states of *the current iteration*:

$$y_i^{(d)} \;=\; \mathcal{P}_\theta\big(x_i^{(d)} \,\big|\, X_{\le i}^{(d)}\big), \qquad X_{\le i}^{(d)} = [x_0^{(d)}, \dots, x_i^{(d)}]. \tag{3}$$

An inner transition then produces the next-depth embedding. For example, Loop (Saunshi et al., 2025) simply sets $x_i^{(d+1)} = y_i^{(d)}$, while Ponder (Zeng et al., 2025) uses a logit-weighted embeddings:

$$x_i^{(d+1)} = \mathrm{softmax}\big(W_{\text{out}}^\top y_i^{(d)}\big)\; E = p_i^{(d)} E. \tag{4}$$

In practice, it uses the top-100 logits instead of full logits for efficiency.

Verbalization occurs at a fixed *maximum depth* $d_{\max}$ shared by all tokens, where $y_i^{(d_{\max})}$ is transformed into the next token t_{i+1}, resembling Equation 2.

4 Selective Latent Iteration Oracles

Setup. We analyze the latent iteration behavior of the Ouro looped transformer (Zhu et al., 2025) and our proposed TaH model (architecture detailed in Section 5). All models are fine-tuned from Qwen3-1.7B-Base on a balanced 100K-sample subset of the Open-R1 dataset (Hugging Face, 2025), following setups in Section 6.1.

Oracle Iteration Policy. To investigate the potential of selective iteration, we establish an oracle policy π. It triggers additional iterations only when the reference LLM θ mispredicts the target token at the first forward pass. In this section, we set θ to the model under evaluation, so π acts as a greedy, locally optimal policy.

Formally, let $\hat{p}_i$ denote the reference model's first-pass next-token distribution at position i, and t_{i+1} the ground-truth token. The oracle iteration depth d_i^π is:

$$d_i^\pi \;=\; 1 + \mathcal{D}(\hat{p}_i^{(1)}, t_{i+1}), \tag{5}$$

where $\mathcal{D}$ is a binary discrepancy metric. We use top-1 mismatch here: $\mathcal{D} = \mathbf{1}[\hat{t}_{i+1} \neq t_{i+1}]$, with $\hat{t}_{i+1} = \arg\max_t \hat{p}_i^{(1)}$ ($\mathbf{1}[\cdot]$ denotes the indicator function). For simplicity, we assume $d_{\max} = 2$. The arbitrary-depth case is in Appendix A.2.5, and alternative discrepancy metrics are ablated in Table 13.

For brevity, we call a token *easy* if the reference model correctly predicts it at the first pass ($d_i^\pi = 1$), and *hard* otherwise. Prior work (Fu et al., 2025) shows that *hard* tokens typically occupy only a small proportion (e.g., 7%) of all tokens.

Next-Token Prediction. We evaluate next-token prediction (NTP) accuracy by comparing top-1 predictions against ground-truth tokens in the Open-R1 validation set. As shown in Table 1, always iterating twice improves NTP accuracy for Ouro by 6.6%, but at the cost of doubled depth. Surprisingly, the oracle policy, which skips iterations on 81-88% tokens, further improves accuracy by 2.1%. This gain comes from avoiding *latent overthinking*: without selective iteration, the model can revise correct predictions to wrong ones at the second pass, as shown in Figure 3.

Downstream Tasks. We next examine whether NTP gains translate to downstream task performance. Since ground-truth tokens are unavailable during generation, we use top-1 predictions from the stronger Qwen3-8B as proxy labels. Table 1 shows that the oracle policy improves downstream performance by 2.0–7.3%.

TaH Design Objectives. The oracle experiments reveal two key insights. First, models have significant untapped potential when iteration depth is selected correctly. Second, the oracle policy is effective enough to learn from, even though other globally optimal policies may exist. These findings motivate the design objectives of TaH: (1) its model architecture should natively support selective iteration depths; and (2) following the oracle policy, its training objective for deeper iterations is not to cover all tokens, but to selectively focus only on the few failing tokens. We later validate that TaH-1.7B can better utilize the oracle policy to achieve $> 25\%$ improvement, surpassing even Qwen3-4B. While the oracle requires ground-truth tokens unavailable at inference, approximating it with neural networks is a promising direction.

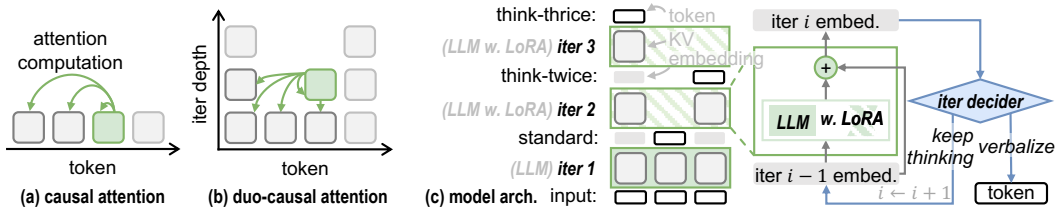


Figure 2: TaH Overview. (a) Regular causal attention: tokens attend only to previous positions. (b) Our duo-causal attention: tokens attend to both previous positions and shallower iteration depths, maintaining 2D causality. (c) Model architecture: TaH selectively iterates or verbalizes tokens. It uses LoRA at deeper iterations to shift from next-token prediction to hard-token refinement. A neural decider determines whether to continue iterating or output the token.

5 TaH Design

We expand the motivations and key designs of TaH in this section, including the duo-causal attention mechanism (Section 5.1), model architecture (Section 5.2), and training scheme (Section 5.3).

5.1 Duo-Causal Attention

Motivation. In looped transformers with fixed depth, standard causal attention on the current iteration's KV states incorporates all context (Equation 3). However, dynamic iteration depths create a challenge: tokens at deeper levels cannot access hidden states of previous tokens that verbalized at shallower depths. This poses a dilemma between requiring up-to-date context from all previous tokens and maintaining parallel training where depth-d computations cannot depend on uncomputed deeper states ($d' > d$). Existing approaches compromise on one of these aspects. Some sacrifice parallelism by allowing attention to deeper iterations (Hao et al., 2024); others preserve parallelism by restricting attention to only the initial iteration's KVs (Bae et al., 2025). To resolve this dilemma, we introduce a simple yet effective mechanism to maximize cross-depth information flow while maintaining high parallelism.

Duo-causal Attention Mechanism. As shown in Figure 2(b), duo-causal attention extends *causality* to two dimensions, letting tokens attend across both previous positions and shallower iteration depths. Formally, we extend the accessible set from Equation 3 to

$$X_{\leq i}^{(\leq d)} \;=\; \{\, x_j^{(k)} \mid j \leq i,\; k \leq d \,\}. \tag{6}$$

When all tokens iterate only once (as in standard transformers), this reduces to regular causal attention. The duo-causal design achieves both full parallel training and cross-depth information flow. At depth d, all tokens compute their depth-d representations simultaneously using *only and all* information from depths 1 through d.

5.2 Depth-adaptive Model Architecture

Motivation. Previous looped transformers typically use identical weights across all iterations. However, we find that over 73% of next-tokens are correctly predicted at the first iteration (Table 1). This suggests deeper iterations serve a different objective: they refine the first iteration's prediction rather than predicting further ahead to the next-next token. This mirrors deep LLMs, where shallow layers predict next tokens for deeper layers to refine (Belrose et al., 2023; Schuster et al., 2022; Bae et al., 2023). While deep LLMs naturally handle this shift through distinct parameters per depth, looped transformers must accommodate both objectives with shared weights, potentially limiting performance. Moreover, fixed iteration depths can cause *latent overthinking*, motivating our dynamic approach.

Backbone Model. To address the objective shift, we apply a LoRA adapter (Hu et al., 2022) to the shared LLM backbone only for iterations $d > 1$. As shown in Figure 2(c), this allows the base LLM to focus on latent embeddings, while the adapter handles the objective shift. It preserves strong next-token prediction at $d = 1$, alleviating interference from deeper iterations. We also add residual

connections across iterations to simplify the refinement and improve gradient flow. Formally, at depth d, we compute

$$y_i^{(d)} \;=\; \mathcal{P}_{\theta_d}\Big(x_i^{(d)} \;\Big|\; X_{\le i}^{(\le d)}\Big), \tag{7}$$

with depth-specific parameters

$$\theta_d = \theta \text{ for } d = 1, \qquad \theta_d = \theta + \Delta \text{ for } d > 1,$$

where θ and Δ denote the LLM and LoRA weights, respectively. The next-iteration inputs use logit-weighted embeddings (Equation 4); verbalization follows standard sampling (Equation 2). Each $y_i^{(d)}$ either continues iterating or verbalizes according to the decider $\mathcal{I}_\phi$.

Iteration Decider. We use a lightweight MLP as the iteration decider $\mathcal{I}_\phi$ to determine whether each token should continue iterating or verbalize. After each iteration, it processes concatenated hidden states from shallow, middle, and final layers of the backbone LLM to predict a continuation probability:

$$\hat{c}_i^{(d)} \;=\; \mathcal{I}_\phi\big(h_i^{(d)}\big) \in [0, 1].$$

During inference, token i verbalizes when $\hat{c}_i^{(d)}$ falls below threshold $c_{\text{threshold}}$ or reaches maximum depth $d_{\max}$.

5.3 Training Scheme

We adopt a two-stage training scheme. We first fine-tune the backbone model to support selective latent iteration, then train the iteration decider. Both stages are aligned to the same oracle iteration policy.

Motivation. As shown in Figure 2(c), the backbone LLM and the iteration decider are tightly coupled, making joint training unstable. Specifically, the backbone's prediction quality across iterations determines the optimal depth, while the decider controls the backbone's KV cache and output depth. To stabilize training, we train the two components sequentially under a fixed oracle policy π (validated in Section 4).

Oracle Iteration Policy π. Our goal is to iterate only on *hard* tokens that a standard supervised fine-tuned (SFT) model would mispredict on the first pass. Thus, we define π with SFT model as the reference: we trigger an additional iteration when its top-1 next-token prediction differs from the ground-truth token. In principle, one could instead define π using the current looped model itself (i.e., an on-policy oracle), but we find this to be empirically unstable (Section 6.3).

Stage 1: Backbone supervision under π. We optimize the backbone LLM (θ and LoRA adapter Δ) with π-guided iteration execution. The loss is standard next-token prediction at the oracle-determined depth:

$$\mathcal{L}_{\text{SFT}}(\theta, \Delta) \;=\; \sum_i \; -\log p_i^{(d_i^\pi)}\big(t_{i+1}\big),$$

where $p_i^{(d_i^\pi)}$ is the next-token distribution at position i, depth d_i^π. This preserves first-iteration accuracy for easy tokens while training deeper iterations to refine hard tokens.

Stage 2: Decider imitation under frozen backbone. We freeze the backbone model (θ, Δ) and train the iteration decider ϕ to imitate the oracle policy's continuation decisions. We minimize weighted binary cross-entropy:

$$\mathcal{L}_{\text{dec}}(\phi) \;=\; -\sum_{i,d} w_d \Big[c_i^{(d)} \log \hat{c}_i^{(d)} + (1 - c_i^{(d)}) \log(1 - \hat{c}_i^{(d)})\Big],$$

where the sum ranges over tokens i and depths $d = 1, \ldots, \min\{d_{\max} - 1, d_i^\pi\}$. Here $c_i^{(d)}$ is the ground-truth continuation label, $\hat{c}_i^{(d)}$ is the predicted probability, and w_d is the class weight for label imbalance (ratio of stop to continue labels).

The two-stage scheme stabilizes training by decoupling backbone learning (conditioned on a fixed π) from iteration policy learning (imitation of π).

6 EXPERIMENT

6.1 SETUP

We present key experiment configurations here, with detailed setups in Appendix A.1.

Baselines. We compare diverse methods under equal parameter budgets, using Qwen3-{0.6B,1.7B,4B}-Base (Yang et al., 2025) as backbones. We compare TaH over the following baselines: (1) *Standard*, which always verbalizes directly and reduces to the standard Qwen model; (2) *SoftThink*, a latent optimization method, implemented following official design (Zhang et al., 2025) on top of the Standard model; (3) *AlwaysThink*, similar to TaH, but always iterates twice for all tokens during training and inference; (4) *Ouro*, a looped transformer that can scale iteration depths, implemented following official design (Zhu et al., 2025). Unless otherwise specified, TaH, Ouro, and AlwaysThink all use a maximum of two iterations, fine-tuned from the same Qwen3 backbone on the same training data and recipe. Ouro reaches its highest performance when set to maximum iterations, so we report results under this setup.

TaH Setup. To match the total parameter count of TaH (with LoRA and decider) with that of baselines, we prune one layer from the LLM backbone before training. The layer is chosen to minimize the increase in validation loss. We also report results for an unpruned variant, TaH+, which adds less than 3% extra parameters from LoRA and decider. The detailed parameter composition is shown in Table 4. Following (Fu et al., 2025), we set the continuation threshold $c_{\text{threshold}} = 0.9$ with about 7% of tokens being iterated twice. The oracle policy π uses Qwen3-0.6B, 1.7B, and 4B as reference models, respectively.

Training Scheme. All models are trained on the balanced Open-R1 (Hugging Face, 2025) mixture (math, QA, and code; 100K samples) using supervised fine-tuning. To fit memory and compute limits, we exclude responses longer than 8,192 tokens; 4B models additionally truncate at 4,096 tokens; all other training settings follow the official Open-R1 script. Each method is sufficiently trained for 5 epochs, and we select the checkpoint with the lowest validation loss as the final model. All backbones are initialized from the corresponding Qwen3-Base model.

Evaluation Setup. We evaluate across challenging reasoning benchmarks, including GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021b), AMC23 (American Mathematics Competitions), AIME25 (American Invitational Mathematics Examination), OlympiadBench (He et al., 2024), MBPP++ (Austin et al., 2021), HumanEval++ (Chen et al., 2021), GPQA-Diamond (denoted as GPQA) (Rein et al., 2023), and MMLU-STEM (Hendrycks et al., 2021a). The maximum generation length is set to 8,192 tokens for all benchmarks, except GSM8K and MMLU-STEM which use 4,096 due to their simpler problems and larger size. Performance is reported as pass@1 under a zero-shot CoT setting, using sampling temperature 0.6. We generate one sample per problem for large datasets (MATH500, OlympiadBench etc.), and eight samples per problem for small datasets (AMC23, AIME25).

6.2 PERFORMANCE

Benchmark Evaluation. We validate TaH's reasoning ability across all nine benchmarks. Table 5 presents performance results for models at 0.6B, 1.7B, and 4B parameter sizes. Compared with the strong Standard Qwen3 baselines, we observe that existing approaches (AlwaysThink, SoftThink, and Ouro) yield only marginal improvements when fine-tuned from base. In contrast, TaH achieves consistent gains across benchmarks. For 0.6B and 1.7B models, TaH delivers average improvements of 3.0% and 3.8% over Standard, respectively; TaH+, which adds less than 3% additional parameters, further pushes these gains to 5.3% and 6.2%. Compared to concurrent work Ouro, TaH and TaH+ achieve 3.8–4.4% and 6.1–6.8% gains, respectively. For 4B models, which are trained with 4K context length due to resource limits, TaH and TaH+ also achieve gains of 1.7% and 2.2%, with potential for further improvement at longer training contexts.

Training Dynamics. During stage 1 (LLM backbone training), TaH performs iterations according to the oracle policy. As shown in Figure 4a, it converges notably faster than the Standard baseline and also achieves much lower validation perplexity. During stage 2 (iteration-decider training), the neural decider successfully imitates the oracle strategy. It reaches about 83% accuracy at predicting the oracle's iteration decisions, as shown in Figure 9.

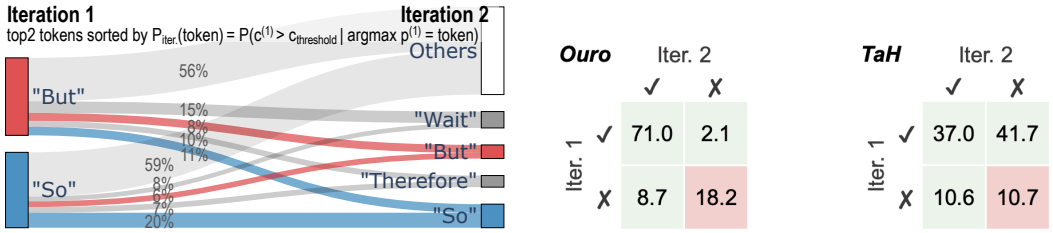
Table 2: Accuracy comparison across benchmarks. Best results are highlighted in bold. *4B models trained with ≤4K lengths due to resource constraints; AlwaysThink is excluded due to Out-Of-Memory (OOM) during training.

	Standard	SoftThink	Ouro	AlwaysThink	TaH	TaH+
			0.6B			
AIME25	1.9	2.9	2.1	1.3	2.1	**4.6**
OlympiadBench	15.4	14.0	14.2	12.6	19.1	**20.6**
AMC23	22.7	22.2	19.7	21.9	24.1	**24.7**
MATH500	39.9	39.6	37.4	37.8	46.2	**51.8**
GSM8K	58.2	55.9	56.6	52.6	63.6	**67.6**
GPQA	31.1	24.7	**35.4**	30.8	29.0	31.3
MMLU-STEM	54.2	53.0	54.0	51.4	56.4	**59.0**
HumanEval++	16.8	14.3	18.9	9.1	21.6	**22.0**
MBPP++	28.8	29.5	23.5	13.8	33.9	**35.1**
Average	29.9	28.5	29.1	25.7	32.9/ + 3.0	**35.2**/ + 5.3
			1.7B			
AIME25	10.8	5.4	10.8	7.5	13.8	**15.4**
OlympiadBench	33.8	30.7	31.3	31.0	37.2	**37.6**
AMC23	39.7	40.3	40.6	40.9	40.9	**48.4**
MATH500	67.8	64.8	68.2	63.2	71.4	**72.6**
GSM8K	80.2	80.0	79.8	74.2	**84.8**	84.5
GPQA	30.3	33.3	32.8	30.5	33.3	**39.4**
MMLU-STEM	74.1	73.5	72.4	69.6	74.8	**76.6**
HumanEval++	39.0	43.3	40.9	16.4	50.0	**51.5**
MBPP++	51.9	49.1	45.5	25.6	55.3	**57.5**
Average	47.5	46.7	46.9	39.9	51.3/ + 3.8	**53.7**/ + 6.2
			*4B**			
AIME25	24.2	25.8	25.0	–	27.1	**27.9**
OlympiadBench	50.4	50.2	51.4	–	50.5	**52.6**
AMC23	64.2	63.1	64.4	–	**69.7**	68.1
MATH500	84.2	85.0	83.8	–	**85.8**	85.6
GSM8K	91.7	**92.5**	90.7	–	91.0	91.7
GPQA	46.2	**50.3**	50.0	–	48.5	49.0
MMLU-STEM	85.4	86.0	85.9	–	86.2	**86.6**
HumanEval++	69.2	69.2	70.7	–	70.1	**72.0**
MBPP++	65.9	66.7	66.7	–	67.7	**68.1**
Average	64.6	65.4	65.4	–	66.3/ + 1.7	**66.8**/ + 2.2

6.3 Design Choice Exploration

We demonstrate the effectiveness and robustness of each design choice of TaH through ablation studies. All experiments in this section train TaH and its variants on the math subset of Open-R1 and evaluate on MATH500, AMC23, and OlympiadBench.

Model Architecture. (1) **Iteration Scheme**. As shown in Table 13, decider-based iteration outperforms the *Always-1* and *Always-2* alternatives, confirming the practical benefits of selective iteration, even with imperfect decisions from neural decider. Note that for *Always-1*, duo-causal attention degenerates to regular causal attention. (2) **Duo-Causal Attention**. Replacing duo-causal attention with standard causal attention variants causes significant drops: (a) attending only to the first iteration (Causal@iter1) drops by 5.4%; (b) attending only to the current iteration (Causal@current) drops even more, by 8.5%. They confirm duo-causal attention's essential role in cross-depth information flow. (3) **Depth Adapters**. Removing LoRA and residual connections leads to consistent drops, confirming their beneficial roles in objective transition across iterations.



(a) Next-token prediction changes across iterations. Top-2 tokens that think-twice most frequently are visualized.

(b) Distribution of token prediction accuracy across iterations for Ouro and TaH.

Figure 3: Behavior analysis visualizations.

Training Scheme. (1) **Supervision type.** Some previous work supervises all iteration depths with next-token labels, while TaH only supervises final output tokens. As shown in Table 13, such *token+latent* supervision underperforms TaH. This aligns with our intuition that different iterations should focus on their respective objectives. (2) **Iteration policy during LLM training.** We compare our static oracle strategy π with two alternatives. The *decider-based* approach trains the iteration decider first, then uses it during backbone training. It suffers from the coupling challenge discussed in Section 5.3. The *dynamic* approach recalculates the oracle using the evolving backbone in Equation 5, facing the same coupling challenge and causing training collapse. These results support our selection of the static oracle policy. (3) **Discrepancy metric in π.** We compare three discrepancy metrics to trigger latent iteration in π: top-1 mismatch, entropy, and cross-entropy (detailed definitions in Appendix A.3.1). As Table 13 shows, top-1 mismatch yields the best result, confirming its empirical effectiveness.

Continuation Threshold. As shown in Figure 4b, TaH maintains robust performance across different continuation thresholds and iteration ratios. We empirically set $c_{\text{threshold}} = 0.9$ for all evaluations.

Iteration Depth. We train a 1.7B TaH-3 with a maximum of three iterations. TaH-3 yields 5.8% average gain over Standard, and 0.8% over TaH-2, as detailed in Appendix A.2.5.

6.4 Behavior Analysis

Token Alternation Patterns. We analyze which tokens TaH selects for deeper iteration. On the validation set, *But* and *So* emerge as top candidates, with iteration probabilities of 34% and 18%, respectively. These tokens signal critical contrasting or causal relationships, confirming that models benefit from additional processing at logically complex junctures. Figure 3a illustrates how TaH alternates predictions after iteration at these key tokens, suggesting logic refinement behavior. See Appendix A.3.4 for detailed analysis.

Attention Pattern. We visualize the attention pattern of TaH in Figure 10 and Appendix A.3.6. The duo-causal attention learns to focus on different iterations in different heads, extracting broader contexts from multiple depths.

Accuracy Landscape. Figure 3 shows NTP accuracy across iterations. Because Ouro trains all iterations to predict all tokens, predictable tokens across depths largely overlap, leaving more tokens unpredictable by any iteration. TaH instead specializes deeper iterations for hard tokens, reducing overlap and improving coverage under selective iteration.

7 Conclusion

We present TaH, a selective latent iteration method that simultaneously improves model performance and efficiency. Architecturally, TaH introduces duo-causal attention, depth-specific LoRA, and a neural iteration decider, enabling effective selective iteration. An oracle policy guides the stable two-stage training of the tightly coupled LLM backbone and decider. Across nine benchmarks, TaH improves accuracy by 5.3–6.2% over strong baselines with minimal overhead, establishing a new direction for improved reasoning under parameter constraints.

ETHICS STATEMENT

This study raises no ethical issues, it did not involve human subjects or sensitive personal data.

REPRODUCIBILITY STATEMENT

This paper provides sufficient information to reproduce the reported results. All experiments were conducted using publicly available datasets together with open-source models and code. Appendix A details implementation aspects, including data selection, hyperparameters, and training procedures. To facilitate full reproducibility, we will release the code, configuration files, and model checkpoints upon publication.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024. URL https://arxiv.org/abs/2404.14219.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. *arXiv preprint arXiv:2310.05424*, 2023.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *arXiv preprint arXiv:2312.04916*, 2023.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*, 2023.

Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. R2r: Efficiently navigating divergent reasoning paths with small-large model token routing. *arXiv preprint arXiv:2505.21600*, 2025.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *URL https://arxiv. org/abs/2310.02226*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL `https://github.com/huggingface/open-r1`.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *Advances in neural information processing systems*, 35:33248–33261, 2022.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Eunki Kim, Sangryul Kim, and James Thorne. Learning to insert [pause] tokens for better reasoning. *arXiv preprint arXiv:2506.03616*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Jindong Li, Yali Fu, Li Fan, Jiahong Liu, Yao Shu, Chengwei Qin, Menglin Yang, Irwin King, and Rex Ying. Implicit reasoning in large language models: A comprehensive survey. *arXiv preprint arXiv:2509.02350*, 2025.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

Tianqiao Liu, Zui Chen, Zitao Liu, Mi Tian, and Weiqi Luo. Expediting and elevating large language model reasoning via hidden chain-of-thought decoding. *arXiv preprint arXiv:2409.08561*, 2024.

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained llms. *arXiv preprint arXiv:2503.23798*, 2025.

Wenheng Ma, Xinhao Yang, Shulin Zeng, Tengxuan Liu, Libo Shen, Hongyi Wang, Shiyao Li, Ke Hong, Zhenhua Zhu, Xuefei Ning, Tsung-Yi Ho, Guohao Dai, and Yu Wang. Cd-llm: A heterogeneous multi-fpga system for batched decoding of 70b+ llms using a compute-dedicated architecture. *ACM Trans. Reconfigurable Technol. Syst.*, October 2025. ISSN 1936-7406. doi: 10.1145/3771288. URL https://doi.org/10.1145/3771288. Just Accepted.

Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *URL https://arxiv. org/abs/2404.15758*, 2404, 2024.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. Token assorted: Mixing latent and text tokens for improved language model reasoning. *arXiv preprint arXiv:2502.03275*, 2025.

MiniCPM Team, Chaojun Xiao, Yuxuan Li, Xu Han, Yuzhuo Bai, Jie Cai, Haotian Chen, Wentong Chen, Xin Cong, Ganqu Cui, Ning Ding, Shengdan Fan, Yewei Fang, Zixuan Fu, Wenyu Guan, Yitong Guan, Junshao Guo, Yufeng Han, Bingxiang He, Yuxiang Huang, Cunliang Kong, Qiuzuo Li, Zhen Li, Dan Liu, Biyuan Lin, Yankai Lin, Xiang Long, Quanyu Lu, Yaxi Lu, Peiyan Luo, Hongya Lyu, Litu Ou, Yinxu Pan, Zekai Qu, Qundong Shi, Zijun Song, Jiayuan Su, Zhou Su, Ao Sun, Xianghui Sun, Peijun Tang, Fangzheng Wang, Feng Wang, Shuo Wang, Yudong Wang, Yesai Wu, Zhenyu Xiao, Jie Xie, Zihao Xie, Yukun Yan, Jiarui Yuan, Kaihuo Zhang, Lei Zhang, Linyue Zhang, Xueren Zhang, Yudi Zhang, Hengyu Zhao, Weilin Zhao, Weilun Zhao, Yuanqian Zhao, Zhi Zheng, Ge Zhou, Jie Zhou, Wei Zhou, Zihan Zhou, Zixuan Zhou, Zhiyuan Liu, Guoyang Zeng, Chao Jia, Dahai Li, and Maosong Sun. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025. URL https://arxiv.org/abs/2506.07900.

Chenyu Wang, Zishen Wan, Hao Kang, Emma Chen, Zhiqiang Xie, Tushar Krishna, Vijay Janapa Reddi, and Yilun Du. Slm-mux: Orchestrating small language models for reasoning. *arXiv preprint arXiv:2510.05077*, 2025a.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025b.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*, 2025.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Fan Yang, Xinhao Yang, Hongyi Wang, Zehao Wang, Zhenhua Zhu, Shulin Zeng, and Yu Wang. Glitches: Gpu-fpga llm inference through a collaborative heterogeneous system. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7. IEEE, 2024.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining language models to ponder in continuous space. *arXiv preprint arXiv:2505.20674*, 2025.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Appendix

A.1 Additional Experiment Setups

A.1.1 Training Recipe

We follow the official training setup of Open-R1 (Hugging Face, 2025) and use the Mixture-of-Thoughts dataset. For our main performance experiments (Section 6.2), we filter samples with output length exceeding 8K tokens from each category (math, code, and QA), then randomly sample 33K examples from each to form a balanced 100K training set. The filtered dataset contains 480M tokens, with 1% reserved for validation. For design choice exploration (Section 6.3), we use only the math subset filtered by 8K output length, resulting in 75K training samples. For 0.6B and 1.7B models, we use a maximum sequence length of 8192 tokens for all methods. For 4B models, we reduce the maximum sequence length to 4096 for Standard, TaH, TaH+, and 3072 for Ouro due to memory constraints. Detailed training hyperparameters are listed in Table 3.

A.1.2 Baseline Setups

AlwaysThink. AlwaysThink uses the same architecture as TaH, except that it iterates twice at every token position and uses standard causal attention (attending only to the current iteration depth) instead of duo-causal attention.

Ouro. We implement Ouro architecture following its official design (Zhu et al., 2025), but fine-tune it on the Open-R1 dataset from Qwen-Base initialization, to align with all other methods. During fine-tuning, we adopt the entropy-regularized loss of Ouro. Since the original Ouro is trained from scratch on different data distributions, performance may differ from the original paper.

A.1.3 Parameter Breakdown

Table 4 reports the parameter breakdown of the Standard, TaH, and TaH+ methods. To offset the additional parameters introduced by TaH through LoRA and the iteration decider, we remove one layer from the LLM backbone, ensuring a fair comparison. In practical deployments, we recommend TaH+, which adds only about 3% additional parameters.

A.1.4 Latent Overthinking Analysis Setup

We set up the oracle experiment in Table 14 to estimate the performance upper bound of our method. The oracle employs the DeepSeek-R1-Distill-Qwen-32B model as a dynamic label generator, replacing the MLP-based iteration decider. During each iteration, we compare the token predictions from the label generator with those from the TaH model. The model continues to iterate only when the top-1 predictions of these two models differ. Due to resource constraints and computational overhead, we evaluated the accuracy only on the first 100 samples from the MATH500 dataset, denoted as MATH100 throughout the paper.

Table 3: Training hyperparameters.

Hyperparameter	**Value**
learning rate	4e-5
max grad norm	0.2
training epochs	5
global batch size	128
warmup ratio	0.03
lr scheduler	cosine (min-lr ratio 0.1)
precision	bfloat16

Table 4: Parameter breakdown of Standard, TaH, and TaH+. Counts are reported using M (million) and B (billion).

Param.	Method	Backbone	LoRA	Iter. Decider	Total
0.6B	Standard	596M	–	–	596M
	TaH	580M	10M	5M	595M
	TaH+	596M	10M	5M	611M
1.7B	Standard	1.72B	–	–	1.72B
	TaH	1.67B	17M	18M	1.71B
	TaH+	1.72B	17M	18M	1.76B
4B	Standard	4.02B	–	–	4.02B
	TaH	3.92B	32M	70M	4.02B
	TaH+	4.02B	33M	70M	4.12B

Table 5: Accuracy comparison of different baselines across five benchmarks and two model sizes. Subscripts indicate improvement over Standard. The top two scores for each task and model size are highlighted in bold.

		Method					
Param.	Benchmark	Standard	Routing	SoftThink	AlwaysThink	TaH	TaH+
0.6B	AIME25	**4.2**	1.0	2.5	1.5	**4.2**	**5.0**
	OlympiadBench	18.8	7.4	19.4	10.2	**23.9**	**24.0**
	AMC23	23.4	10.9	24.1	15.6	**32.5**	**30.6**
	MATH500	47.2	27.3	48.8	32.8	**51.2**	**54.2**
	GSM8K	62.5	45.6	61.3	54.6	**64.4**	**68.8**
	Average	31.2	18.5	31.2	22.9	**35.2**$_{/+4.0}$	**36.5**$_{/+5.3}$
1.7B	AIME25	13.3	10.2	12.9	10.0	**17.9**	**14.6**
	OlympiadBench	33.0	30.6	33.4	30.0	**38.8**	**41.2**
	AMC23	42.2	42.2	43.1	42.5	**48.4**	**51.2**
	MATH500	68.4	60.0	68.8	61.8	**74.4**	**73.0**
	GSM8K	82.1	71.2	79.6	79.3	**84.5**	**85.8**
	Average	47.8	36.8	47.6	44.7	**52.8**$_{/+5.0}$	**53.2**$_{/+5.4}$
4B	AIME25	23.3	22.5	22.5		**30.4**	**28.3**
	OlympiadBench	47.7	45.0	50.1		**50.5**	**52.0**
	AMC23	62.8	60.9	64.1	OOM	**70.3**	**70.6**
	MATH500	82.8	76.1	83.2		**84.4**	**85.6**
	GSM8K	90.5	85.3	**90.9**		90.4	**91.5**
	Average	61.4	58.0	62.2	–	**65.2**$_{/+3.8}$	**65.6**$_{/+4.2}$

A.2 Additional Experimental Results

A.2.1 Performance on Math Benchmarks

Math Benchmark Evaluation. In this section, all models are trained on the math subset of Open-R1 and evaluated on math benchmarks (see Section A.1.1 for training details). We also add a *Routing* baseline, which selects a model from a candidate pair for each question. In our experiments, we use two pairs: (1) MobileLLM-R1-360M, Qwen3-1.7B, and (2) Qwen3-0.6B, Qwen3-4B. All candidate models are SFT-trained under the same settings. For each pair, the routing ratio is calibrated so that the average active parameter count matches our 0.6B and 1.7B targets, respectively. Table 5 reports results for 0.6B, 1.7B, and 4B backbones across five challenging math benchmarks. Even with strong Qwen3-Base initialization, existing approaches show limited effectiveness: AlwaysThink and routing methods fail to consistently outperform the standard baseline, while SoftThink yields only marginal gains. In contrast, TaH achieves stable improvements, with average gains of 4.0% (0.6B),

Table 6: Performance of math-only trained models (0.6B and 1.7B) on in-domain math benchmarks and the out-of-domain STEM benchmark (MMLU-STEM).

Param.	Benchmark	Standard	SoftThink	AlwaysThink	TaH+
0.6	MATH500	47.2	48.8	32.8	**54.2**
	AMC23	23.4	24.1	15.6	**30.6**
	MMLU-STEM	51.6	51.4	42.6	**56.3**
	Average	40.7	41.4	30.3	**47.0**
1.7	MATH500	68.4	68.8	61.8	**73.0**
	AMC23	42.2	43.1	42.5	**51.2**
	MMLU-STEM	70.8	70.6	63.8	**73.7**
	Average	60.5	60.8	56.0	**66.0**

Table 7: Per-component latency breakdown on a single A800 GPU.

Component	Standard		TaH		AlwaysThink	
	Latency (s)	Ratio(%)	Latency (s)	Ratio(%)	Latency (s)	Ratio(%)
Iter-1 Forward	210.6	100.0	229.8	76.2	224.1	30.0
Iter-2 Forward	–	–	29.6	9.8	384.7	51.5
Iter. Decider	–	–	10.5	3.5	–	–
LoRA Switching	–	–	7.5	2.5	91.1	12.2
Other	–	–	24.1	8.0	47.4	6.3

5.0% (1.7B), and 3.8% (4B) over Standard. TaH+, which adds less than 3% parameters, further improves to 5.3%, 5.4%, and 4.2%, respectively. For 0.6B and 1.7B, TaH achieves 8.1–11.3% gains over AlwaysThink, and TaH+ achieves 8.5–12.6% gains. AlwaysThink-4B is not evaluated due to out-of-memory during training.

Out-Of-Distribution (OOD) Performance. We further evaluated the zero-shot generalization capability of models trained solely on math datasets from the main paper. As shown in Table 6, TaH+ demonstrates consistent improvements not only on in-domain math benchmarks (MATH500, AMC23) but also on out-of-domain tasks like MMLU-STEM. This indicates that the thinking patterns learned by TaH+ on math problems are robust and transferrable to broader scientific reasoning tasks.

A.2.2 Real-world Efficiency

Setup. We investigate the real-world efficiency of different 1.7B models under our current implementation. All measurements were obtained on a single A800 GPU with a batch size of 1 and a maximum output length of 8192 tokens, using a challenging AIME25 problem where all three methods reached the token limit. Memory usage was profiled using `torch.cuda.memory._record_memory_history`.

Latency Breakdown. We report the decoding latency, throughput, and a detailed time breakdown for Standard, AlwaysThink, and TaH in Table 7. Here, *Iter-1 forward* and *Iter-2 forward* denote the total forward-pass time spent on the first and second latent iterations, respectively; *Iter decider* is the time for the iteration decider network to judge whether to continue iterating or verbalize; *LoRA switching* is the overhead of switching LoRA adapters; and *Other* includes tensor initialization, concatenation, and related bookkeeping.

Table 9: Input tokens (shared across methods) and output token / iteration statistics for Standard, AlwaysThink, TaH, and TaH+ (Math-trained version).

			Standard		**AlwaysThink**		**TaH**		**TaH+**	
Param.	**Dataset**	**In.**	Out.	Iter.	Out.	Iter.	Out.	Iter.	Out.	Iter.
0.6B	AIME25	159	7450	1.00	7316	2.00	7648	1.05	7486	1.06
	OlympiadBench	100	6599	1.00	6622	2.00	6631	1.09	6513	1.06
	AMC23	85	6377	1.00	6368	2.00	6242	1.05	6145	1.05
	MATH500	71	4823	1.00	5350	2.00	4877	1.05	4793	1.06
	GSM8K	61	1955	1.00	2844	2.00	1923	1.07	1791	1.07
	Average ratio	–	1.00×	1.00×	1.02×	2.00×	1.00×	1.06×	0.97×	1.06×
1.7B	AIME25	159	7195	1.00	7173	2.00	7496	1.06	7498	1.06
	OlympiadBench	100	6008	1.00	6484	2.00	6387	1.06	6258	1.06
	AMC23	85	5681	1.00	7543	2.00	6122	1.04	5852	1.06
	MATH500	71	4004	1.00	4414	2.00	4233	1.06	4286	1.06
	GSM8K	61	1451	1.00	1644	2.00	1721	1.08	1686	1.08
	Average ratio	–	1.00×	1.00×	1.13×	2.00×	1.09×	1.06×	1.07×	1.06×

Efficiency Comparsion. We compare efficiency of 1.7B models at 8K length on a single NVIDIA A800 GPU. As shown in Table 8, TaH iterates twice on only 6% of tokens on AIME25, with 1.48× lower memory overhead and 2.48× faster decoding than AlwaysThink, while achieving higher accuracy. More detailed efficiency experiment setup and runtime breakdown are shown in Section A.2.2.

Table 8: Real-world decoding performance on a single A800 GPU, tested on AIME25 with 8K max token length. TPS denotes tokens per second.

	Standard	AlwaysThink	TaH
Avg. Depth	1.00	2.00	1.06
Memory (GB)	4.3	6.8	4.6
Latency (s)	210.6	747.2	301.4
TPS	38.9	11.0	27.2

Discussion. We note that our current implementation is not yet optimized at the system level, so there remains room for further efficiency improvements. For example, the *LoRA Switching* and *Other* overheads (bookkeeping) are relatively high due to the Python-level implementation of dynamic control flow. These engineering optimizations are largely orthogonal to the algorithmic design of TaH, and we plan to continue refining the implementation to further reduce latency and memory overhead. The theoretical FLOPs and memory access analysis of TaH are provided in Appendix A.2.3.

A.2.3 Theoretical Efficiency Analysis

Following prior work Hoffmann et al. (2022); Yang et al. (2024); Ma et al. (2025), we analyze the computational and memory access overhead of TaH relative to the Standard and AlwaysThink baselines. Table 9 presents the average number of input/output tokens and latent iterations per token across five benchmarks. We use these statistics to calculate the theoretical computation and memory access costs for each method.

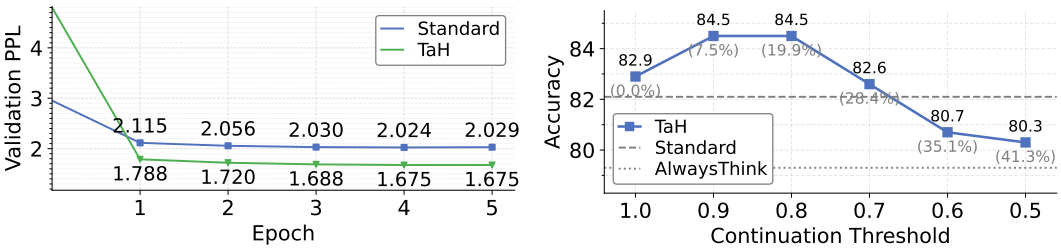
As shown in Table 10, TaH incurs only a marginal increase in cost per token (1.04 to 1.05×) compared to the Standard baseline. In comparison, *AlwaysThink* is prohibitively expensive, requiring 2.19 to 2.27× more computation and memory access. These theoretical results confirm that TaH exceeds the reasoning benefits of fixed-depth looped transformers without the substantial efficiency penalty.

A.2.4 Training Dynamics and Continuation Threshold

Figure 4a shows the stage 1 training dynamics, and Figure 4b reports GSM8K accuracy under different continuation thresholds (with iterate-twice ratios).

Table 10: Decoding computation (GFLOPs) and memory access (GB) per output token for Standard, AlwaysThink, TaH and TaH+ methods.

		Standard		**AlwaysThink**		**TaH**		**TaH+**	
Param.	**Dataset**	Comp.	Mem.	Comp.	Mem.	Comp.	Mem.	Comp.	Mem.
0.6B	AIME25	1.47	1.38	3.35	3.14	1.52	1.43	1.57	1.47
	OlympiadBench	1.41	1.32	3.21	3.02	1.51	1.42	1.50	1.41
	AMC23	1.40	1.31	3.17	2.97	1.43	1.34	1.46	1.37
	MATH500	1.31	1.22	2.98	2.80	1.35	1.26	1.39	1.31
	GSM8K	1.14	1.06	2.54	2.38	1.19	1.12	1.22	1.14
	Average ratio	1.00×	1.00×	2.27×	2.27×	1.04×	1.04×	1.06×	1.06×
1.7B	AIME25	4.31	4.03	9.45	8.83	4.51	4.21	4.64	4.34
	OlympiadBench	4.16	3.88	9.18	8.58	4.36	4.07	4.48	4.18
	AMC23	4.12	3.85	9.54	8.91	4.24	3.96	4.43	4.13
	MATH500	3.92	3.66	8.45	7.89	4.10	3.83	4.23	3.95
	GSM8K	3.62	3.38	7.48	6.98	3.87	3.61	3.98	3.72
	Average ratio	1.00×	1.00×	2.19×	2.19×	1.05×	1.05×	1.08×	1.08×



(a) Training dynamics of the LLM backbone on Qwen3-0.6B-Base. TaH converges rapidly and achieves lower perplexity.

(b) GSM8K accuracy with respect to continuation threshold. Numbers in brackets show the percentage of iterate-twice tokens.

Figure 4: Training dynamics and continuation threshold analysis.

A.2.5 Iteration Depth Beyond Two

Hard Token Labeling. Previous works have proposed many methods to evaluate the hardness of each tokens in the training data, like through excess loss (Lin et al., 2024; Xie et al., 2023), entropy (Wang et al., 2025b; Chen et al., 2023) and prediction difference (Fu et al., 2025).

For shallow iteration budgets within two ($D_{\max} \leq 2$), we adopt the prediction difference policy. It simply labels the tokens that do not yield top-1 in next-token prediction at the first iteration as hard tokens. Formally, we use a binary halting rule:

$$H_i^{\pi} = \begin{cases} 0, & \text{if } h_i = 0 \quad \text{(easy token)} \\ D_{\max}, & \text{if } h_i = 1 \quad \text{(hard token)} \end{cases} \tag{8}$$

If the iteration depth goes beyond 2 ($D_{\max} > 2$), we use the reference model's cross-entropy as a non-binary indicator of difficulty. Define

$$\ell_i^{\text{ref}} = -\log p_{i,\text{ref}}^{(0)}(t_{i+1}).$$

We then map difficulty to halting depth via monotone quantile binning:

$$H_i^{\pi} = \left\lfloor \text{QuantileRank}\left(\ell_i^{\text{ref}}\right) \cdot D_{\max} \right\rfloor, \tag{9}$$

where $\text{QuantileRank}(\cdot) \in [0, 1]$ is the empirical CDF over the training set (higher loss $\Rightarrow$ deeper halting). This induces per-depth continuation labels $c_i^{(d)} = \mathbb{1}[d < H_i^{\pi}]$ for $d \in \{0, 1, \ldots, D_{\max}\}$.

Table 11: Performance comparison between TaH-2 and TaH-3 (maximum per-token iterations of 2 and 3, respectively). Iter.2 and Iter.3 denote the per-token percentages executing two and three iterations, respectively.

		Standard	**TaH-2**		**TaH-3**		
Param.	**Dataset**	Acc.	Acc.	Iter.2	Acc.	Iter.2	Iter.3
1.7B	MATH500	68.4	**74.4**	5.6	72.6	5.3	0.2
	GSM8K	82.1	84.5	7.5	**84.8**	7.6	0.3
	AMC23	42.2	48.4	4.2	**48.7**	5.1	0.1
	OlympiadBench	33.0	38.8	5.7	**41.6**	5.4	0.2
	AIME25	13.3	17.9	6.0	**20.4**	5.3	0.1
	Average	47.8	52.8	5.8	**53.6**	5.7	0.2

Table 12: Performance on MATH500 and GSM8K-500 (first 500 GSM8K samples)

	Method	
Dataset	Standard-0.6B	Ponder-1.4B
MATH500	47.2	2.0
GSM8K-500	62.8	1.8
Avg.	55	1.9

Experiment Result. Specifically, we train a 1.7B TaH with a maximum per-token iteration count of 3, using oracle labels generated by the method described above. As shown in Table 11, TaH-3 achieves a further improvement of **0.8%** on average over TaH-2.

A.2.6 Additional Latent Thinking Methods

Some latent thinking methods require pre-training and use base models other than Qwen3. We also compare with these methods, including Ponder (Zeng et al., 2025). Specifically, we adopt the released pretrained PonderingPythia-1.4B as the base model and perform SFT on the same training data. We observe that the fine-tuned model learns the stylistic patterns of the training data, but still underperforms substantially, which may be attributable to the limited capability of the PonderingPythia-1.4B backbone.

A.2.7 Design Choice Exploration

We conduct controlled ablations where each variant changes a single component from the default TaH configuration to isolate its effect. Table 13 reports the final downstream results, and Figure 12 expands it by showing validation perplexity dynamics across different supervision signals and iteration policies. The naming convention matches Table 13. TaH with token-only supervision and the oracle policy yields lower perplexity than *iter. decider* and *token+latent*. Although the *dynamic* policy achieves the lowest perplexity, it fails on downstream tasks and often produces infinite-loop generations.

A.2.8 Latent Overthinking

To analyze latent thinking patterns, we verbalize tokens from all iteration depths using their last-layer hidden states. The oracle method uses the oracle policy π from Section 5.3 for iteration decision. (1) **Generation.** Since ground-truth tokens are unavailable during generation, we use predictions from the stronger DeepSeek-R1-Distill-Qwen-32B model (Guo et al., 2025) as proxy labels. Table 14 shows that the oracle policy substantially improves performance by verbalizing correct predictions immediately while iterating only on incorrect ones. With our trained iteration decider approximating the oracle, TaH outperforms both Standard and AlwaysThink baselines. However, the ideal oracle policy achieves even higher gains, indicating future potential. (2) **Next-token prediction**. We evaluate next-token prediction accuracy on the Open-R1 dataset, using the test model itself

Table 13: Ablation study on design choices of TaH-0.6B. Each row varies one aspect from TaH configuration (marked with gray).

Variant	MATH500	AMC23	Olympiad	Average
TaH	**51.2**	**32.5**	**23.9**	**35.9**
Model Architecture				
Iteration Depth (TaH: Neural decider)				
Always-1	47.2	23.4	18.8	29.8/−6.1
Always-2	32.8	15.6	10.2	19.5/−16.4
Attention (TaH: Duo-causal)				
Causal@iter1	47.8	24.4	19.4	30.5/−5.4
Causal@current	42.0	23.8	16.4	27.4/−8.5
Depth Adapters (TaH: LoRA + Residual)				
w/o LoRA	51.6	29.7	22.4	34.6/−1.3
w/o LoRA & Res.	49.2	22.5	21.2	31.0/−4.9
Training Scheme				
Superv. (TaH: Token-only)				
Token+latent	49.4	29.6	15.9	31.6/−4.3
Iter. Policy (TaH: Oracle policy)				
Decider-based	44.8	24.1	17.3	28.7/−7.2
Dynamic	11.0	2.8	2.7	5.5/−30.4
Discrepancy metric in π (TaH: Top1 mismatch)				
Cross-entropy	47.4	21.2	20.4	29.7/−6.2
Entropy	42.0	21.9	16.9	26.9/−9.0

Table 14: Impact of iteration schemes on Qwen3-0.6B (first 100 MATH500 samples).

Training	Inference	Accuracy
Standard	Standard	52
AlwaysThink	AlwaysThink	38/−14
AlwaysThink	TaH-Oracle	77/+25
TaH-Oracle	TaH-Decider	54/+ 2
TaH-Oracle	TaH-Oracle	80/+28

as the reference model in π. Figure 1 reveals that AlwaysThink produces more incorrect than correct revisions, demonstrating latent overthinking. In contrast, oracle-controlled iterations substantially increase correct revisions by selectively targeting hard tokens.

A.3 Additional Analysis

A.3.1 Oracle Policy and Hard Token Analysis

Metrics for Hard Token Labeling. We investigate different metrics for defining hard tokens to validate our choice of top-1 prediction mismatch. We compare three labeling strategies:

1. Top-1 Mismatch (TaH Default): Labels a token as hard if the reference model's greedy prediction differs from the ground truth.
2. Entropy (TaH-Entropy): Labels a token as hard if the reference model's prediction entropy exceeds a threshold.
3. Cross-Entropy (TaH-CE): Labels a token as hard if the reference model's cross-entropy loss exceeds a threshold.

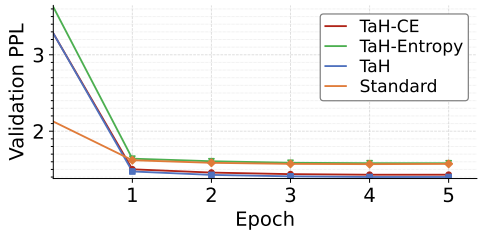


Figure 5: Validation loss curves of 0.6B models trained with different oracle labeling metrics. The default Top-1 Mismatch policy yields the lowest validation loss.

Table 15: Performance comparison of different difficulty metrics (Entropy, Cross-Entropy, and Top-1 Accuracy) on 0.6B models. All methods mark the same total number of tokens as "hard."

Method	MATH500	AMC23	OlympiadBench	Average
TaH-Entropy	42.0	21.9	16.9	26.9
TaH-CE	47.4	21.2	20.4	29.7
TaH	**51.2**	**32.5**	**23.9**	**35.9**

To ensure a fair comparison, for *TaH-Entropy* and *TaH-CE*, we set the thresholds such that the number of hard tokens in each sample matches the total ratio from the default Top-1 Mismatch policy. This isolates the impact of *which* tokens are selected, rather than *how many*.

Figure 5 compares the validation loss, and Table 15 reports downstream accuracy on 0.6B models. While cross-entropy (*TaH-CE*) improves over entropy labeling (*TaH-Entropy*), the Top-1 Mismatch policy (*TaH*) achieves superior performance across all benchmarks. This empirically verifies that directly targeting tokens where the model's top-1 prediction is wrong is the most effective way to identify hard tokens for TaH.

Labeling Robustness. We investigate the robustness of hard-token labels with respect to the choice of reference model. We do so by analyzing the consistency of hard-token identification across different model scales (e.g., Qwen3-0.6B, 1.7B, and 4B).

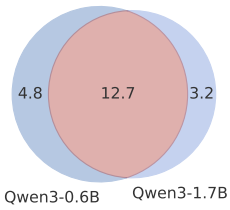
First, we quantify the agreement between models. As shown in Figure 6, hard tokens exhibit high consistency across scales. Notably, even a smaller, less accurate reference model (1.7B) successfully identifies 81% of the hard tokens for a larger model (4B).

Second, to understand the quality of this overlap, we partition tokens into an *overlap set* (marked as hard by both models) and a *non-overlap set* (marked as hard by only one model). We plot the cross-entropy loss under each reference model in Figure 7. We observe that overlap tokens have substantially higher average cross-entropy ($\approx 2.0\times$) than non-overlap tokens for *all* reference models. This indicates that either reference model can identify this core set of "hard" tokens, which corresponds to positions of genuine, high uncertainty. It reveals a consensus on hardness among models even of different sizes.

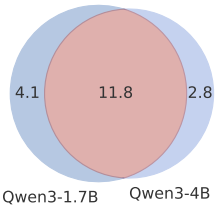
A.3.2 Iteration Decider Robustness

We evaluate the iteration decider, trained on the general OpenR1 corpus, across three validation subsets (Math, Code, and QA) to quantify its robustness and cross-domain generalizability. As summarized in Table 16, the decider maintains high decision accuracy across all domains without any retraining.

Despite being invoked on only 7.8-26.6% of tokens, the decider consistently yields 5.8-7.9% absolute accuracy gains over the standard single-pass baseline on all three domains. Moreover, the decider automatically adjusts its iteration rate according to task difficulty: it iterates more frequently on QA (26.6%) than on Math (7.8%), even under a fixed threshold $c_{\text{threshold}} = 0.9$. This behavior indicates that the decider responds to intrinsic uncertainty signals in the model's predictive distri-

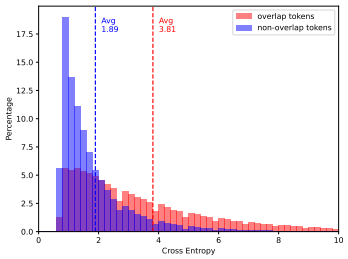


(a) Overlap between Qwen3-0.6B and 1.7B

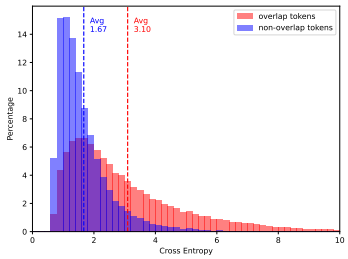


(b) Overlap between Qwen3-1.7B and 4B

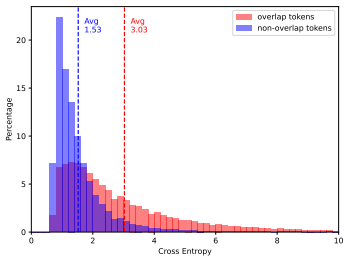
Figure 6: Venn diagrams illustrating the overlap of hard-token labels between different reference models. The high overlap proportions indicate that "hard" tokens are largely consistent across model scales.



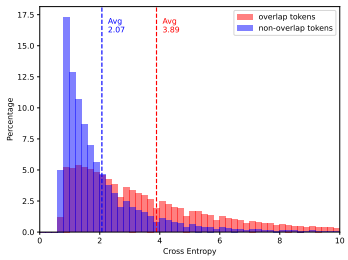
(a) Qwen3-1.7B: cross-entropy of overlap vs. non-overlap hard tokens (w.r.t. Qwen3-0.6B).



(b) Qwen3-0.6B: cross-entropy of overlap vs. non-overlap hard tokens (w.r.t. Qwen3-1.7B).



(c) Qwen3-4B: cross-entropy of overlap vs. non-overlap hard tokens (w.r.t. Qwen3-1.7B).



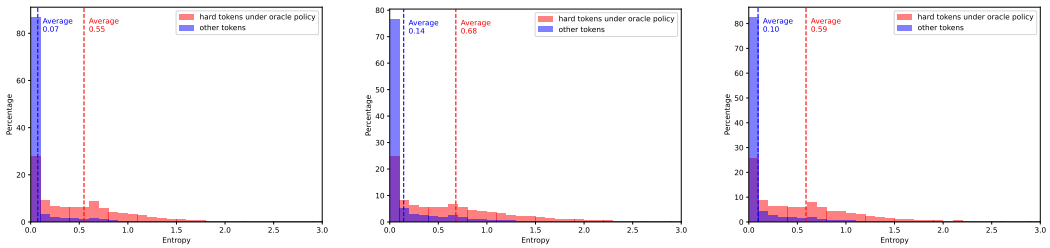
(d) Qwen3-1.7B: cross-entropy of overlap vs. non-overlap hard tokens (w.r.t. Qwen3-4B).

Figure 7: Token-level cross-entropy distributions of overlap and non-overlap hard tokens across different reference model pairs. For each pair of reference models (e.g., Qwen3-1.7B and Qwen3-0.6B), we plot the cross-entropy of tokens labeled as hard by both models (overlap) and by only one model (non-overlap) on both reference models.

bution rather than memorizing domain-specific patterns, consistent with the token-level difficulty analysis in Appendix A.3.3.

Table 16: Iteration decider behavior and downstream gains on different validation subsets. The decider is trained once on general OpenR1 and evaluated without retraining.

Metric	**Math**	**Code**	**QA**
Iteration Percentage	7.8%	10.7%	26.6%
Iteration Accuracy	86.7%	82.3%	76.6%
Benchmark Gain over Standard	**+6.8%**	**+7.9%**	**+5.8%**



(a) Math (b) Science (c) Code

Figure 8: Output Logit entropy distribution at the first iteration of TaH, categorized by oracle policy's difficulty labels (hard token) on the OpenR1 validation set (Math, QA, Code). The distinct separation between distributions confirms that TaH's internal logits provide a strong, task-agnostic signal for identifying hard tokens.

Table 17: Conditional probabilities of continuation confidence and next-token distribution.

Token T_1	$P(c^{(1)} > c_{\text{threshold}} \mid t^{(1)} = T_1)$	**Token** T_2	$P(t^{(2)} = T_2 \mid t^{(1)} = T_1)$
But	34.3%	So	13.63%
		Wait	12.17%
		Therefore	8.95%
So	17.7%	So	28.17%
		Therefore	13.67%
		But	4.89%

A.3.3 Hard Token Identifiability

Why is the iteration decider robust and generalizable across tasks? We investigate this by analyzing the intrinsic properties of "hard" tokens.

We compute the token entropy of hard and easy tokens across three diverse subsets of the OpenR1 dataset (Math, Science, and Code). As shown in Figure 8, hard tokens exhibit a universal signature of significantly higher entropy ($> 5\times$) compared to easy tokens. This distinct separation confirms that "hardness" is an intrinsic, robustly identifiable property of the model's predictive state, rather than a complex, task-specific pattern. Given this clear signal, the neural iteration decider can easily learn reliable classification strategies that generalize well across different domains.

A.3.4 Token Alternation Pattern

We analyze tokens that most frequently trigger a second iteration ("think-twice" tokens). For each token type t, we compute the continuation rate

$$\Pr\left(c_i^{(1)} > c_{\text{threshold}} \mid t_i = t\right),$$

using the inference threshold $c_{\text{threshold}} = 0.9$ (Section 5.3). We estimate this quantity on the OpenR1 validation set and, for diagnostics, randomly sample 10K token positions (≈0.4% of tokens) to track whether the next-token prediction switches between depth 1 and depth 2. This setting quantifies which token types most often trigger an additional iteration and how often iteration alters the predicted next token.

A.3.5 Iteration Decision Error

We analyze how iteration decision accuracy affects TaH's end-to-end response quality, since iteration decider will not be perfect as shown in Figure 9. To this end, we randomly inject errors into the oracle iteration-decider predictions at different rates. Formally, we denote the original oracle prediction as the *label* $l \in \{0, 1\}$ and the altered prediction as the *output* $o \in \{0, 1\}$. We define the *iter. error* as

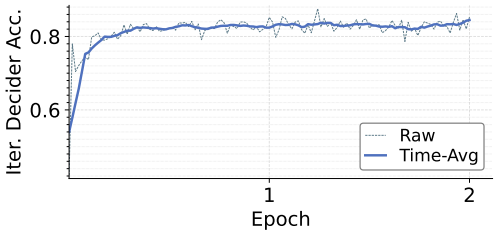


Figure 9: Iteration-decider accuracy vs. epoch (Qwen3-0.6B).

Table 18: TaH performance under different iteration-decider error rates. All values are reported in percentages.

Iter. Error (%)	Underthink (%)	Overthink (%)	MATH100 Accuracy (%)
0.0	0.0	0.0	80.0
2.8	2.8	0.0	78.0
10.0	1.5	8.5	55.4
15.0	2.1	12.9	45.2
20.0	2.5	17.5	27.1
22.1	0.0	22.1	21.6

the total proportion of deliberately introduced errors:

$$\text{iter. error} = P(l \neq o) = \underbrace{P(l = 1, o = 0)}_{\textit{underthink rate}} + \underbrace{P(l = 0, o = 1)}_{\textit{overthink rate}}. \tag{10}$$

We further distinguish the impacts of overthinking and underthinking. Here, overthinking refers to cases where the decider incorrectly signals *continue*, while underthinking corresponds to cases where it incorrectly signals *stop*. Table 18 shows how TaH's MATH100 accuracy varies with different iteration error rates. We quantify these effects by fitting a linear model to the data:

$$\text{accuracy} = -1.41 \times \text{underthink rate} - 2.73 \times \text{overthink rate} + 0.81.$$

This analysis indicates that inaccurate iteration decisions are the main factor behind the performance gap between TaH and its oracle variant, with overthinking being the dominant source of performance gaps.

A.3.6 Duo-causal Attention Pattern

We perform forward computation on 100 samples, each with a length of 128 tokens. Figure 10 shows the average attention weights of three representative attention heads in the second iteration of the TaH model. The left panel illustrates a head that mainly attends to keys from the first iteration. The middle panel shows a head focusing on keys from the second iteration. The right panel displays a head with a balanced attention distribution. These results suggest that the TaH model, under the duo-causal attention mechanism, can automatically learn diverse attention patterns across layers and heads.

Figure 11 further presents the total attention scores assigned to keys in the first iteration. It can be seen that the first layer tends to focus more on keys from the second iteration. Different layers also exhibit varying attention behaviors.

A.4 Implementation Details

A.4.1 Duo-causal attention implementation

Figure 13 illustrates the implementation of duo-causal attention, with the formal definitions provided below.

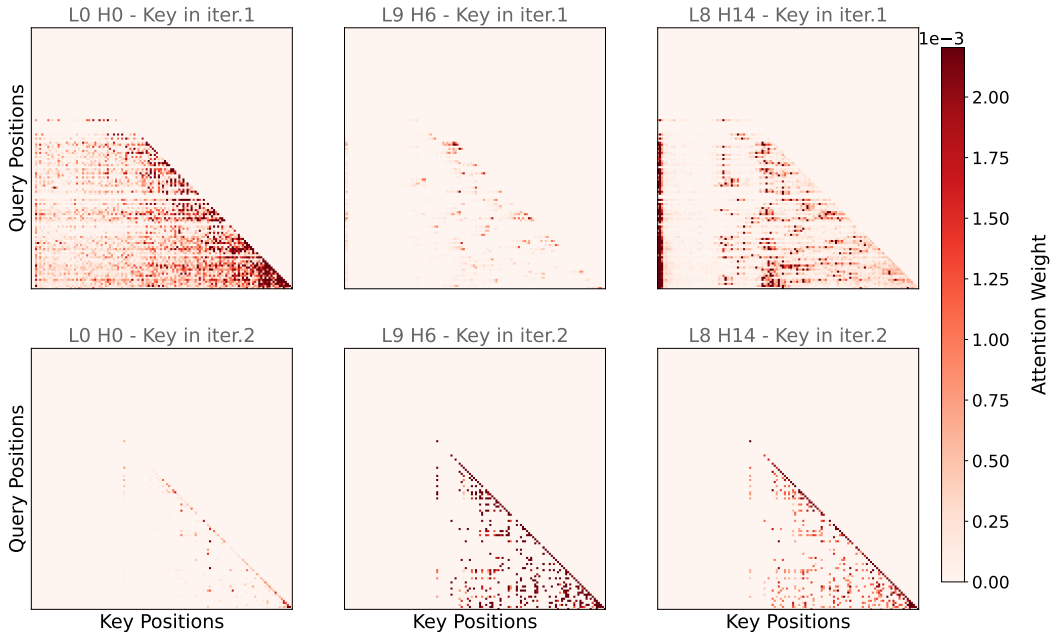


Figure 10: TaH duo-causal attention pattern.

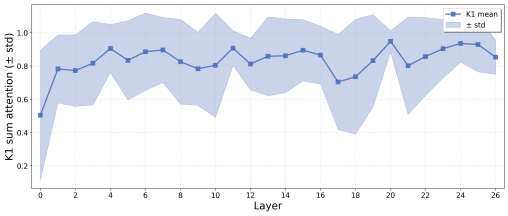


Figure 11: TaH mean and standard deviation of attention weights (key from iteration 1) across layers in iteration 2.

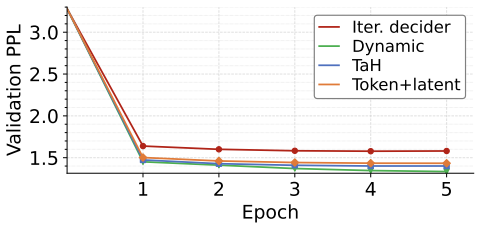


Figure 12: Validation perplexity for different training schemes.

(1) KV cache concatenation. At depth d, we form the visible K/V sequence by concatenating all shallower-to-current depths along the sequence dimension:

$$\mathrm{KV}^{(\le d)} = [\mathrm{KV}^{(1)}; \mathrm{KV}^{(2)}; \cdots; \mathrm{KV}^{(d)}].$$

This realizes the accessible set in Equation 6, allowing deeper iterations to access all shallower iterations while preserving positional causality. The KV cache is managed by iteration depth during decoding, as shown in Figure 13(b). The fragmented KV-cache management strategy is standard in existing LLM serving systems (Kwon et al., 2023; Zheng et al., 2024).

(2) Two-dimensional causal mask. For a query (i, d), a key (j, k) is attendable iff $j \le i$ and $k \le d$. We implement this as an additive attention mask with 0 for allowed entries and $-\infty$ otherwise, enforcing positional and iteration causality jointly. Figure 13(c) visualize the landscape of the duo-causal attention mask. When $d = 1$ for all tokens, the rule reduces to standard causal attention.

(3) Compatibility with efficient attention. The mask is provided in the standard additive form and the concatenated K/V remain contiguous along the sequence dimension, matching the usual scaled dot-product attention interface. As a result, duo-causal attention is directly compatible with optimized kernels such as FlashAttention, without kernel modifications.

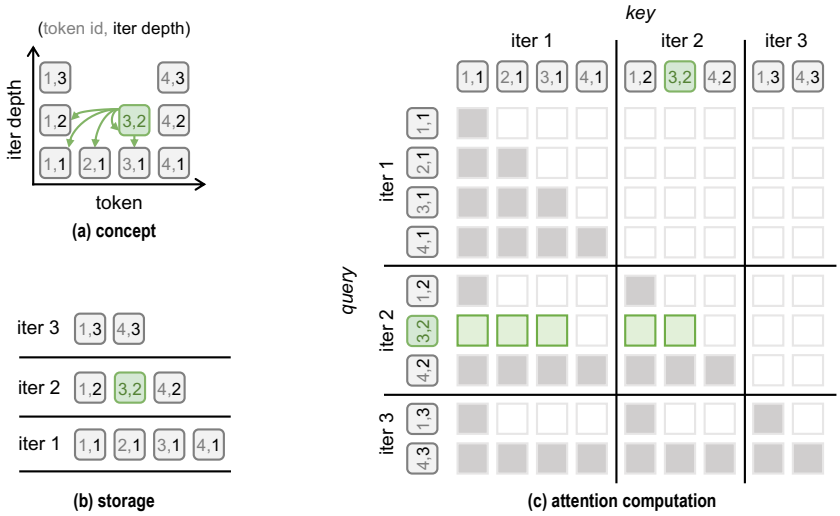


Figure 13: Duo-causal attention implementation. (a) Conceptual TaH example with dynamic iteration depths. Each cell denotes a token–depth pair (token id, iter depth). (b) Each iteration maintains its own KV cache. (c) KV caches from all iterations are concatenated into a 1D sequence and processed with standard attention under a duo-causal mask. The duo-causal mask is conceptually partitioned into blocks by iteration depth. The diagonal blocks use a standard causal mask, while off-diagonal blocks use reduced causal masks that enforce the duo-causal rules.

A.5 Additional Related Work

Signal-guided Control. These methods keep reasoning in token space but steer computation by inserting control tokens. They add filler tokens (e.g., dots) (Pfau et al., 2024), learnable `[PAUSE]` tokens (Goyal et al., 2024; Kim et al., 2025) for extra compute during decoding. They are lightweight but constrained to discrete-token interventions with limited latent control.

Latent Optimization. These methods perform autoregressive reasoning directly in internal representations, emitting little or no intermediate text. They distill CoT into continuous embeddings via progressive replacement (Hao et al., 2024; Cheng & Van Durme, 2024), hidden-state alignment (Su et al., 2025; Liu et al., 2024), or logit-weighted embeddings (Zhang et al., 2025). While efficient, these methods sacrifice interpretability, with training-based ones requiring heavy mitigation from verbal LLMs.

Instead of using the shared model parameter multiple times through latent iteration, previous work also proposes layer skipping methods for dynamic computing allocation.

Layer Skipping. Layer skipping aims to accelerate LLM inference by dynamically bypassing certain layers for specific tokens. Some methods use a learnable module to make real-time skipping decisions. MoD (Raposo et al., 2024) uses a top-k router to select a subset of tokens for processing, while FlexiDepth (Luo et al., 2025) uses a plug-in router to determine whether a layer should be bypassed. Others use a fixed strategy to skip layers. SkipDecode (Del Corro et al., 2023) enforces a monotonically decreasing number of active layers during generation. However, these methods still require loading the entire model's parameters, resulting in a large memory access overhead. Instead of skipping some layers, TaH adds computational depth by allowing core tokens to undergo multiple refinement iterations. This approach provides greater computational depth without increasing the model's parameter count.

A.6 Limitations and Future Work

Comparison with Official Qwen3 Models. Official Qwen3 models are trained on different data distributions and scales, and use different training procedures, including on-policy distillation (Yang et al., 2025). By contrast, our models use SFT only on limited, publicly accessible data. Consequently, performance may differ between the two.

Future Work. We consider advanced training techniques largely orthogonal to TaH. Promising directions include: (1) integrating TaH with online distillation to provide stronger supervision; (2) applying reinforcement learning to optimize the compute–accuracy trade-off and to learn continuation policies beyond imitation. Due to resource constraints, this work focuses on SFT with public data; we leave these combinations for future work.

A.7 Use of Large Language Models (LLMs)

Large language models (e.g., GPT-5, Gemini) were used exclusively for proofreading and minor copyediting. They were not employed to generate scientific content, contribute to research ideation, or design methods. All research ideas, methods, and conclusions are solely the responsibility of the authors.

A.8 Examples

We use an example question from AIME25 and responses from Standard, TaH and AlwaysThink to provide an intuitive understanding of our method. Only TaH can successfully answer this question. Blue-highlighted tokens indicate positions that undergo two iterations.

Text 1. **Question**

On $\triangle ABC$ points A, D, E, and B lie that order on side $\overline{AB}$ with $AD = 4, DE = 16$, and $EB = 8$. Points A, F, G, and C lie in that order on side $\overline{AC}$ with $AF = 13, FG = 52$, and $GC = 26$. Let M be the reflection of D through F, and let N be the reflection of G through E. Quadrilateral $DEGF$ has area 288. Find the area of heptagon $AFNBCEM$.

Text 2. Standard Response

Okay, let me try to figure out this problem step by step. It's about a heptagon formed by reflecting points and finding its area. ... We have triangle ABC. On side AB, points A, D, E, and B are in that order. The lengths are given as $AD = 4$, $DE = 16$, and $EB = 8$. So $AB = 28$ ... On side AC, points A, F, G, and C are in that order with $AF = 13$, $FG = 52$, and $GC = 26$. So $AC = 91$ ... Let me place point A at $(0, 0)$ and B at $(28, 0)$ ... Let C be at $(91, h)$ ... Then $D = (4, 0)$, $E = (20, 0)$, $F = \left(13, \frac{13h}{91}\right)$, $G = \left(65, \frac{65h}{91}\right)$ ...
Area of quadrilateral $DEGF$ can be calculated ... Applying the shoelace formula ... Area $= \frac{1}{2} \cdot \frac{1248h}{91} = \frac{624h}{91} = 6h$
So with 288: $6h = 288 \Rightarrow h = \frac{288}{6} = 48$
Reflections: M is the reflection of D through F and N is the reflection of G through E ... $M = (22, 0)$, $N = (-39, 0)$, $B = (28, 0)$, $C = (91, 48)$, $E = (20, 0)$ ...
Using the shoelace formula for $AFNBCEM$ in order ... Sum $= 0 + 0+0+1344+(-960)+0+0 = 384$
Take absolute value and multiply by $1/2$: Area $= \frac{1}{2} \cdot |384| = 192$
...
Final Answer $\boxed{192}$ ✗

Text 3. TaH Response

Okay, let's try to tackle this geometry problem. It seems a bit complex with all these points and reflections, ... First, let me visualize triangle ABC. Points D, E, and B are on side AB with AD = 4, DE = 16, and EB = 8. ... Let me set up a coordinate system. Let's place point A at the origin $(0, 0)$ ... Then, since AB is a line segment, let me assume AB is along the x-axis. So point B would be at $(28, 0)$ ... Now, points F, G, C are on AC ... Let me define coordinates for C as (c_x, c_y) ...
...
Compute the sum of $x_i y_{i+1} - x_{i+1} y_i$ for each i ...
Sum of all terms: $0 + \frac{1300c_y}{91} + 0 - > \frac{52c_y}{91} = \frac{(1300-52)c_y}{91} = \frac{1248c_y}{91}$
Then, the area is $\frac{1}{2} \cdot \left|\frac{1248c_y}{91}\right| = \frac{624c_y}{91}$
...
Compute the area of this heptagon ...
Using shoelace formula ...
Term2: –240 Term3: 840 Term4: 1176 Term5: –840 Term6: 240 ...
Sum: $0 - 240 + 840 + 1176 - 840 + 240 + 0 = 1176$
Take absolute value and multiply by $1/2$: Area $= \frac{1}{2} \cdot |1176| = 588$
...
Final Answer $\boxed{588}$ ✓

Text 4. AlwaysThink Response

Okay, so I need to find the area of the heptagon AFNBCEM. Let me first try to visualize the problem. There's triangle ABC. On side AB, there are points A, D, E, and B. The order is A, D, E, B. The distances given are AD=4, DE=16, and EB=8. So, starting at A, moving 4 units to D, then 16 units to E, then 8 units to B. So, the total length of AB is AD + DE + EB = 4 + 16 + 8 = 28 units. Got that.
...
Points in order: D(4,0), E(20,0), F(13,h), G(65,h), back to D(4,0).
Compute the terms:
...
Sum these terms: $0 + 20h - 52h - 4h = -36h$
Take absolute value and multiply by 1/2: $Area = (1/2) * | - 36h| = (1/2) * 36|h| = 18|h|$
But the area is given as 288, so $18|h| = 288 \Rightarrow |h| = 16$.
...
Now, apply the shoelace formula ... Therefore, total sum is $8640/17$
Take absolute value and multiply by 1/2: $Area = (1/2) * |8640/17| = 8640/34 = 4320/17 \approx 254.1176$
...
Therefore, the answer is $4320/17$.
Final Answer $\boxed{\frac{4320}{117}}$ ✗