SUPERTHOUGHTS: REASONING TOKENS IN SUPERPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long Chain-of-Thought (CoT) reasoning improves LLM problem-solving but is computationally expensive due to sequential token generation. While recent works explore reasoning in continuous latent spaces to bypass discrete token generation, they often struggle with training stability and fail to scale to complex, long-horizon tasks due to lack of supervision signal. We propose SuperThoughts, which compresses pairs of consecutive CoT tokens into single latent representations and decodes two tokens per step via a lightweight Multi-Token Prediction (MTP) module. This preserves discrete token supervision at training time while increasing decoding throughput at inference time. We finetune Qwen2.5-Math-1.5B-Instruct, Qwen2.5-Math-7B-Instruct, Qwen2.5-Math-14B-Instruct, and evaluate on MATH500, AMC, OlympiadBench, and GPQA-Diamond. With a confidence-based adaptive mechanism that falls back to standard decoding when uncertain, SuperThoughts achieves $\sim$20–30% CoT length reduction while maintaining accuracy with minimal degradation (1-2 points accuracy drop on most tasks).

1 INTRODUCTION

Large language models (LLMs) solve complex problems by generating explicit Chain-of-Thought (CoT) sequences before answering (Wei et al., 2022). We can view each CoT token as a unit of compute (one forward pass). Recent successes such as OpenAI o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) demonstrate that spending more of this test-time compute on longer chains substantially improves performance (Snell et al., 2024). This raises a question: *why must the model reason in discrete token space?* The vocabulary is a finite, human-interpretable set of symbols, yet the model's internal representations live in a continuous, high-dimensional vector space. If reasoning could occur directly in this richer latent space, the model might express more intermediate computations per step, achieving the same quality with fewer steps, or better quality with the same compute.

Recent work on *latent reasoning* aims to bypass discrete token generation: COCONUT (Hao et al., 2024) trains models to reason with continuous latent thoughts that are never decoded into language, Cheng & Van Durme (2024) compress chain-of-thought into dense representations via knowledge distillation, and other approaches explore hybrid schemes that interleave latent and discrete tokens (Su et al., 2025; Shen et al., 2025b; Zhang et al., 2025a). However, these methods face a key challenge: *the lack of intermediate supervision*. Standard CoT training benefits from a dense, token-level cross-entropy signal at every reasoning step. In an unconstrained latent space, this supervision vanishes, and the model must learn useful intermediate representations without any direct feedback on what they should encode. This makes training unstable and prone to representational drift, particularly for long-horizon tasks where errors compound across many latent steps. As a result, prior methods have been demonstrated primarily in simple settings and often struggle to match explicit CoT on challenging benchmarks.

We propose **SuperThoughts** as a natural first step toward reasoning in a superposed space while keeping intermediate supervision. At each step, the model consumes *two* consecutive CoT tokens compressed into a single superposed embedding (Figure 1) and predicts two discrete tokens: one from the main model backbone and one from a lightweight Multi-Token Prediction (MTP) module (Gloeckle et al., 2024; Liu et al., 2024). This halves the number of forward passes while maintaining token-level cross-entropy supervision throughout training. Our contributions are:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

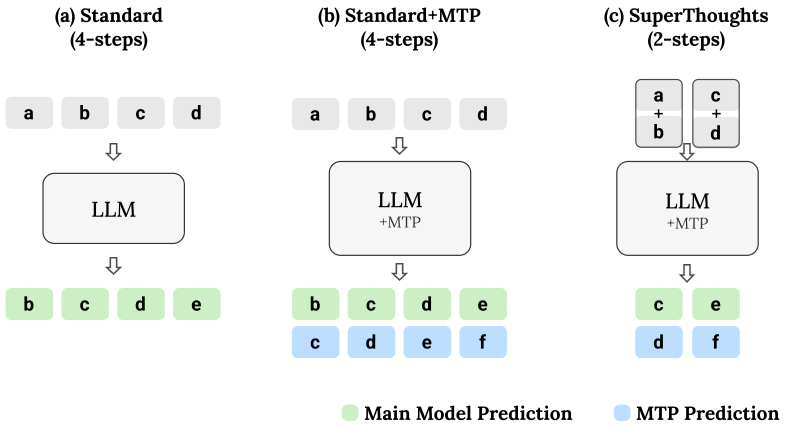


Figure 1: Comparison of three generation strategies for producing tokens "b" through "e". **(a) Standard:** Each forward pass consumes one token and predicts one token, requiring 4 steps. **(b) Standard + MTP:** A Multi-Token Prediction head predicts an additional token per step, but inputs remain single tokens, still requiring 4 steps. **(c) SuperThoughts:** Token pairs are fused into superposed embeddings as input, and two tokens are decoded per step via MTP, halving the required forward passes to 2 steps. Green denotes main model predictions; blue denotes MTP predictions.

1. We propose the SuperThoughts architecture (a Compressor, a Main Module and an MTP Module) and a two-stage training protocol that first aligns the compressed latent space via distillation (Berton et al., 2025), then jointly trains all components end-to-end with discrete token supervision.

2. We introduce a confidence-based adaptive inference mechanism that falls back to standard decoding when the MTP module is uncertain, trading throughput for accuracy on difficult reasoning steps.

3. We evaluate on MATH500 (Hendrycks et al., 2021), AMC23 (MAA, 2023), OlympiadBench (He et al., 2024) and GPQA-Diamond (Rein et al., 2024), achieving 20–35% CoT length reduction while maintaining accuracy within 1–2 points of the baseline.

2 Related Works

Latent Reasoning in LLMs. Recently, several works focus on using CoT states beyond discrete tokens. Hao et al. (2024); Yue et al. (2025); Shen et al. (2025b) directly feed the last continuous hidden state as input embedding for the next step, but these methods either require complicated training curriculum or only consider simple settings. Giannou et al. (2025); Zhang et al. (2025a); Deng et al. (2025); Shen et al. (2025a) generate first and then compress the newly generated tokens, but only save context length and involve attention mask manipulations that are not compatible with modern inference engines (Kwon et al., 2023; Zheng et al., 2024). Cheng & Van Durme (2024); Su et al. (2025); Tan et al. (2025) train the model to compress discrete CoT into latent tokens and during inference generate latent tokens directly. Several works explore composing multiple next token choices into a latent input token (Zhang et al., 2025b; Zhuang et al., 2025; Zhu et al., 2025; Jain & Rappazzo, 2025; Wu et al., 2025; Tang et al., 2026; Gozeten et al., 2026). Peng et al. (2026) pretrain LLMs with token superposition, which yields a pretraining time speedup.

Compressed Input Context. Many works compress more information into input embeddings. Prefix Tuning (Li & Liang, 2021) uses a learned soft embedding prefix to condition the LLM, and other works compress input context tokens to save context length (Jiang et al., 2023; Li et al., 2023; Mu et al., 2023; Berton et al., 2025; Feldman & Artzi, 2025).

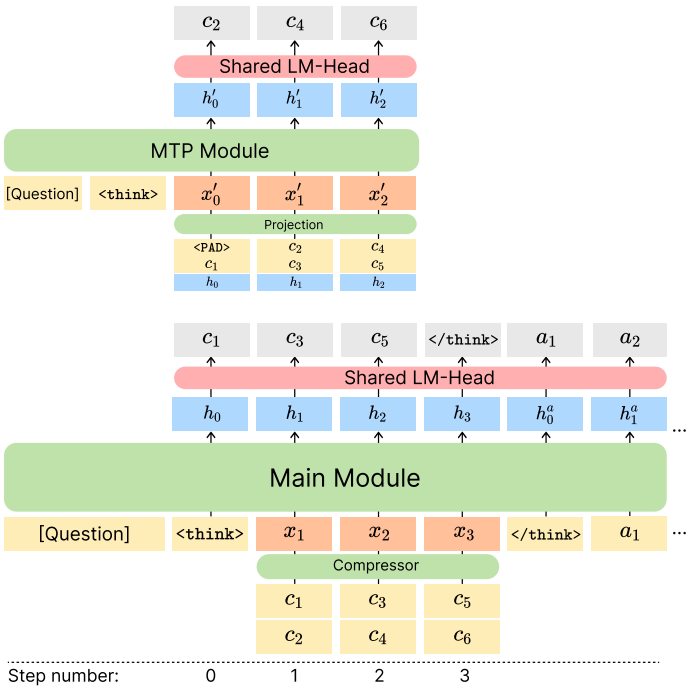


Figure 2: **Overview of SuperThoughts architecture.** At each step i, the Compressor encodes a CoT token pair (c_{2i-1}, c_{2i}) into a single latent vector $\boldsymbol{x}_i$ via a learned $2H \rightarrow H$ compressor, where H is the dimension of a single token embedding. The Main module processes $\boldsymbol{x}_i$ to produce hidden state $\boldsymbol{h}_i$ and predicts the next odd-indexed token c_{2i+1}. The MTP module then receives a projection $\boldsymbol{x}'_i$ that combines three inputs – the previous even token, the just-predicted odd token from Main, and the Main hidden state – via a learned $3H \rightarrow H$ projection, and predicts the corresponding even-indexed token c_{2i+2}. Both modules share the same output LM head. This design enables the model to consume two tokens and generate two tokens per step.

Reducing Discrete CoT Tokens. Many methods have also produced shorter discrete CoT sequences through Reinforcement Learning (Aggarwal & Welleck, 2025; Shrivastava et al., 2025) and fine-tuning (Xia et al., 2025). These methods are orthogonal to SuperThoughts.

Variable Compute Per Token. Recent work has explored adaptive compute allocation in language models by moving beyond uniform token-level processing, such as BLT (Pagnoni et al., 2025) and H-Net (Hwang et al., 2025) segmenting bytes into dynamically-sized patches, and DLCM (Qu et al., 2026) learning variable-length semantic concepts on top of tokens. Liu & Zhang (2025) propose HAMBURGER, which similarly fuses multiple tokens into a single input embedding via a compositional embedder and decodes several tokens per forward through a micro-step decoder.

Multi-token Prediction. Several works train LLMs to predict multiple future tokens at once (Bachmann & Nagarajan, 2024; Gloeckle et al., 2024; Liu et al., 2024; Ahn et al., 2025). However, at inference time current MTP architectures can only utilize the extra tokens for self-speculative decoding (Liu et al., 2024; Gloeckle et al., 2024; Cai et al., 2024), since the main model still needs to populate KV entries for tokens the MTP modules generate. This does not reduce the total FLOPs: self-speculative decoding with MTP targets latency reduction under low GPU utilization rather than computational efficiency (see Appendix A).

3 METHODS

Standard Chain-of-Thought (CoT) reasoning generates a sequence of discrete tokens $c_{1:L} = (c_1, \ldots, c_L)$ autoregressively, requiring L forward passes to produce a reasoning chain of length L. We introduce **SuperThoughts**, a framework that halves this computational cost by processing and generating tokens in pairs. At each reasoning step, the model consumes two tokens and predicts

two tokens, reducing the number of forward passes from L to $L/2$ while preserving discrete token supervision.

In this section, we detail: **(1) Architecture (Section 3.1):** The three components of our model that includes a Compressor, a Main module, and a lightweight Multi-Token Prediction (MTP) Module; **(2) Training (Section 3.2):** A two-stage protocol that first aligns the compressed latent space via distillation, then jointly trains all components; and **(3) Adaptive Inference (Section 3.3):** A decoding algorithm that dynamically falls back to standard single-token generation when model confidence is low.

3.1 SUPERTHOUGHTS ARCHITECTURE

Our model processes reasoning chains by operating on superposed token pairs during the thinking process rather than individual CoT tokens. We structure each example as a sequence

$$\underbrace{q_{1:L_q}\ \texttt{<think>}}_{\text{prompt tokens}}\ \underbrace{c_{1:L_c}\ \texttt{</think>}\ a_{1:L_a}}_{\text{response tokens}},$$

where $q_{1:L_q}$ denotes question tokens, $c_{1:L_c}$ denotes CoT tokens, and $a_{1:L_a}$ denotes answer tokens. The sequence includes special delimiter tokens <think> and </think>, with <think> appended to the prompt to initiate reasoning. We reorganize the CoT sequence $c_{1:L_c}$ into a sequence of pairs, reducing the effective reasoning length from L_c to $S = L_c/2$ steps, where we assume L_c is even and S represents the number of superposition steps; if L_c is odd, we pad with a special token to maintain the pair structure.

Our model consists of three components: (1) Compressor, (2) Main module and (3) Multi-Token Prediction (MTP) Module. At CoT phase, for each step i, the Compressor encodes a token pair (c_{2i-1}, c_{2i}) into a single latent vector, from which the Main module predicts the next token c_{2i+1} and the MTP module predicts the next-next token c_{2i+2}.

Compressor. Let $\texttt{Emb}(\cdot) \in \mathbb{R}^H$ denote the token embedding function. For each step $i = 1, \ldots, S$, the compressor $\texttt{Comp}(\cdot)$ maps the pair $(\texttt{Emb}(c_{2i-1}), \texttt{Emb}(c_{2i}))$ into a single compressed vector $\boldsymbol{x}_i \in \mathbb{R}^H$. We explore two implementations for the compressor: either a *Linear Projection*, where we concatenate the token embeddings and project them using a learnable matrix $P \in \mathbb{R}^{H \times 2H}$:

$$\boldsymbol{x}_i = P \begin{bmatrix} \texttt{Emb}(c_{2i-1}) \\ \texttt{Emb}(c_{2i}) \end{bmatrix},$$

or a *Transformer Block*, where we process the pair using a small Transformer layer and extract the hidden state corresponding to the second token (denoted by the subscript 2):

$$\boldsymbol{x}_i = \texttt{TF}\big(\texttt{Emb}(c_{2i-1}), \texttt{Emb}(c_{2i})\big)_2,$$

where $[\cdot, \cdot]$ denotes sequence concatenation. These compressed vectors $\boldsymbol{x}_{1:S}$ serve as the inputs to the Main module.

Main module. The Main module (base LLM) is the primary reasoning backbone, responsible for evolving the latent reasoning state and predicting *odd-indexed* tokens. At each time step, it takes as input the compressed representation of two tokens, and outputs a latent reasoning state, which is fed to a language modeling head to predict the next token. In more detail, step $i = 0$, the Main module takes in <think>, produces $\boldsymbol{h}_0$ and predicts c_1; at steps $i \geq 1$, it takes in $\boldsymbol{x}_i$, produces the hidden state $\boldsymbol{h}_i$ and predicts c_{2i+1}. At the final step $i = S$, it predicts the closing delimiter </think> from $\boldsymbol{h}_S$. The Main and MTP modules share the same output language modeling head.

MTP module. The MTP module is responsible for predicting *even-indexed* tokens. At each step, it takes as input the hidden representation output from Main module, some of the past tokens, and predicts next to next token. In more detail, at step i, it takes in the previous even token c_{2i} (with $c_0 =$ <PAD>), the current odd token c_{2i+1} just predicted by Main and the hidden state $\boldsymbol{h}_i$ from Main. These are compressed into $\boldsymbol{x}'_i \in \mathbb{R}^H$ via a learnable projection $P' \in \mathbb{R}^{H \times 3H}$:

$$\boldsymbol{x}'_i = P' \begin{bmatrix} \texttt{RMSNorm}(\texttt{Emb}(c_{2i})) \\ \texttt{RMSNorm}(\texttt{Emb}(c_{2i+1})) \\ \texttt{RMSNorm}(\boldsymbol{h}_i) \end{bmatrix}.$$

A one-layer transformer then produces $\boldsymbol{h}'_i$ and predicts c_{2i+2}.

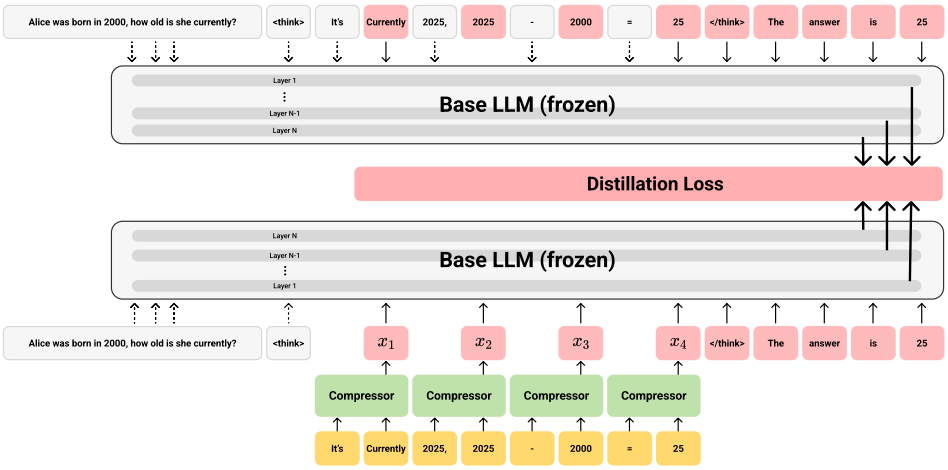


Figure 3: **Compressor training via latent distillation. Top (Teacher):** The frozen Base LLM processes the full discrete token sequence. **Bottom (Student):** The same frozen Base LLM receives compressed representations $\boldsymbol{x}_i$ from the Compressor, which fuses each CoT token pair (e.g., "It's" + "Currently" $\rightarrow \boldsymbol{x}_1$). The Compressor is trained to minimize the Smooth-L_1 distance between teacher and student hidden states across all layers at corresponding positions. Positions used to compute the distillation loss are marked in red.

3.2 Training Strategy

Training a model to reason in latent space can be unstable if the compressed representations drift significantly from the pre-trained language manifold. To mitigate this, we employ a two-stage training protocol: first, we warm-start the compressor via knowledge distillation to align the latent space; second, we jointly train the entire model using standard cross-entropy loss.

Stage 1: Training Compressor module via latent distillation. Before end-to-end training, we train the Compressor module via distillation, following Berton et al. (2025). Consider the Main module processing the discrete sequence $[c_1, c_2, \ldots, c_{2S}]$ one token at a time (the *teacher*), versus the Main module processing compressed pairs $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_S]$ where $\boldsymbol{x}_i = \texttt{Comp}(c_{2i-1}, c_{2i})$ (the *student*). We train the Compressor so that the student's hidden state after $\boldsymbol{x}_i$ matches the teacher's hidden state (layer-wise) after c_{2i}. In effect, the model should produce the same hidden states whether processing tokens discretely or in compressed form.

We define the set of distillation targets D as pairs of corresponding positions in the teacher (uncompressed) and student (compressed) sequences. Following Berton et al. (2025), we include all answer token positions. Crucially, we extend their approach by also including all *even* CoT token positions, enforcing alignment within the reasoning chain itself and not just the final answer.

The loss is computed as the Smooth-L_1 distance between the teacher's hidden states $H^{(\ell)}$ and the student's hidden states $\tilde{H}^{(\ell)}$ across all layers ℓ and target pairs $(t, t') \in D$:

$$\mathcal{L}_{\text{distill}} = \sum_{\ell} \frac{1}{\sigma^{(\ell)}|D|} \sum_{(t,t')\in D} \text{SmoothL1}_\beta\big(H_t^{(\ell)},\, \tilde{H}_{t'}^{(\ell)}\big),$$

where $\sigma^{(\ell)} = \text{Std}(H_D^{(\ell)})$ is layer-wise normalization and the Smooth-L_1 distance $\text{SmoothL1}_\beta(u, v)$ is defined as $\frac{1}{d}\sum_{i=0}^{d} \text{SmoothL1}_\beta(u, v)_i$ with

$$\text{SmoothL1}_\beta(u, v)_i = \begin{cases} \dfrac{1}{2}\dfrac{(u_i - v_i)^2}{\beta}, & |u_i - v_i| < \beta, \\ |u_i - v_i| - \dfrac{\beta}{2}, & \text{otherwise.} \end{cases}$$

Stage 2: Joint training with cross-entropy loss. Once the compressor is aligned, we train the full model (Main + MTP + Compressor) end-to-end. This essentially minimizes the cross entropy loss

for all tokens predicted (including those coming from the Main module and the MTP module). Let $\ell(y \mid \boldsymbol{h}) := \mathrm{CE}\big(y, \mathrm{head}(\boldsymbol{h})\big)$ denote token-level cross-entropy (CE) loss. Define the Main targets $y_i^{\mathrm{main}} = c_{2i+1}$ for $i = 0, 1, \ldots, S-1$, and $y_S^{\mathrm{main}} =$ </think>. The CoT losses are

$$\mathcal{L}_{\mathrm{NTP}}^{\mathrm{CoT}} = \frac{1}{S+1} \sum_{i=0}^{S} \ell\big(y_i^{\mathrm{main}} \mid \boldsymbol{h}_i\big)\,,$$

$$\mathcal{L}_{\mathrm{MTP}}^{\mathrm{CoT}} = \frac{1}{S} \sum_{i=0}^{S-1} \ell(c_{2i+2} \mid \boldsymbol{h}'_i)\,.$$

Let $\mathcal{L}_{\mathrm{answer}}$ denote the standard next-token CE loss on the answer tokens. We define

$$\mathcal{L}_{\mathrm{NTP}} := \mathcal{L}_{\mathrm{answer}} + \mathcal{L}_{\mathrm{NTP}}^{\mathrm{CoT}}, \mathcal{L}_{\mathrm{Training}} = \mathcal{L}_{\mathrm{NTP}} + \lambda\, \mathcal{L}_{\mathrm{MTP}}^{\mathrm{CoT}}.$$

Since we train the MTP module from scratch, we first freeze the Main and Compressor module to train the MTP module, after which we unfreeze all modules and jointly train them.

3.3 Confidence-based Adaptive Inference

While our model can process two CoT tokens within one step with superposed tokens, this sometimes can degrade the performance. For example, if the model needs to decode two "hard" tokens, a single superposition step may lack sufficient computational capacity to predict both correctly. Ideally, the model should allocate more compute to "hard tokens" via discrete reasoning, while processing "easy tokens" efficiently with superposition.

We implement this by looking at the confidence of the MTP module. At each step i, after the Main module predicts the odd token c_{2i+1}, the MTP module produces $\boldsymbol{h}'_i$ that will be used to predict c_{2i+2}. Let

$$p_i^{\mathrm{MTP}} = \max_{j \in V} \mathrm{softmax}(\mathrm{head}(\boldsymbol{h}'_i))_j$$

be the maximum probability of the MTP prediction at step i and τ be a threshold. If $p_i^{\mathrm{MTP}} < \tau$, the MTP module is not confident, and we reject its prediction. On the next step $i+1$, instead of feeding a compressed pair to the Main module, we input $\mathtt{Emb}(c_{2i+1})$ directly and re-predict c_{2i+2} using the more powerful Main module, after which MTP tries to predict c_{2i+3} followed by the same acceptance check. This fallback mechanism allows the model to self-regulate its speed, processing two tokens for easy text while slowing down for difficult reasoning steps. We analyze the inference cost in Appendix B.

4 Experiments

4.1 Experimental Setup

We use Qwen2.5-Math-1.5B-Instruct and Qwen2.5-Math-7B-Instruct as the model we start from and post-train it to reason in superposition. The Main module is initialized the same as Qwen2.5 and the MTP module is initialized using the weights of the last layer. The projection matrices are initialized as a map that averages the input embeddings.

To train the model, we curate a synthetic reasoning dataset. We collect questions from Albalak et al. (2025) and Moshkov et al. (2025). For each question, we let Qwen2.5 to generate 10 responses with CoT. We filter out incorrect responses and further filter down to $\sim$1.5M responses. For each sample, we consider the whole response as the CoT part and the "\boxed{...}" part as the answer part. The dataset preparation process is detailed in Appendix C.

The Main module is initialized the same as the pretrained Qwen2.5 model. In stage 1 training, we choose a projection matrix or a 1-layer transformer as two variants of the Compressor module and uses learning rate 1×10^{-4}.

In stage 2 training, we first freeze the Compressor and the Main module and only train MTP with learning rate 5×10^{-4}. Then for the joint training, we use learning rate 1×10^{-5} and train for 2

Table 1: Accuracy and average correct CoT length of Qwen-2.5-Math-Instruct-1.5B/7B models on three benchmarks. We trained two variants of SuperThoughts model, one with a projection matrix as the Compressor and another with a 1-layer Transformer as the Compressor. The baseline CoT is trained on the same dataset as we train SuperThoughts.

		MATH500		OlympiadBench		AMC23	
		Acc (%)	CoT Len.	Acc (%)	CoT Len.	Acc (%)	CoT Len.
Qwen2.5-Math-1.5B-Instruct	Standard CoT	72.4	494.1	34.72	620.5	59.53	660.5
	SuperThoughts *(Projection Compressor)*						
	w/o adaptive	55.6	229.6	20.03	286.9	38.28	351.0
	w/ adaptive, $\tau = 0.999$	73.0	314.8	33.83	436.3	57.97	471.6
	w/ adaptive, $\tau = 0.9999$	72.0	355.5	33.09	463.0	57.03	531.1
	w/ adaptive, $\tau = 0.99999$	71.4	408.5	34.42	537.3	58.13	590.9
	SuperThoughts *(Transformer Compressor)*						
	w/o adaptive	55.8	224.9	21.01	289.4	40.78	348.7
	w/ adaptive, $\tau = 0.999$	72.2	319.8	32.49	423.7	56.41	459.3
	w/ adaptive, $\tau = 0.9999$	71.0	354.2	33.68	463.6	57.66	524.9
	w/ adaptive, $\tau = 0.99999$	72.0	406.7	34.13	530.7	58.44	562.7
Qwen2.5-Math-7B-Instruct	Standard CoT	83.0	538.6	36.35	654.6	69.22	749.4
	SuperThoughts *(Projection Compressor)*						
	w/o adaptive	72.4	252.1	30.71	339.5	57.11	359.2
	w/ adaptive, $\tau = 0.999$	81.0	334.7	35.31	415.6	66.72	467.2
	w/ adaptive, $\tau = 0.9999$	80.8	357.3	35.46	456.8	67.89	509.9
	w/ adaptive, $\tau = 0.99999$	82.4	404.2	38.13	553.0	67.11	583.1
	SuperThoughts *(Transformer Compressor)*						
	w/o adaptive	72.0	251.1	27.89	336.5	57.97	363.0
	w/ adaptive, $\tau = 0.999$	80.4	323.6	37.24	425.6	67.19	467.1
	w/ adaptive, $\tau = 0.9999$	80.8	360.3	35.46	442.2	67.66	514.8
	w/ adaptive, $\tau = 0.99999$	82.0	411.2	37.24	541.1	66.02	539.3

epochs. For model trained without using adaptive inference, we use $\lambda = 1.0$; for model trained to perform adaptive inference, we use $\lambda = 0.02$. This is because since uncertain MTP predictions can be rejected and re-predicted by the Main module in the subsequent step, we prioritize the Main module's accuracy and thus do not need to optimize the MTP loss as aggressively. We train Qwen2.5 on the same dataset as a baseline and discuss our baseline choice in Appendix A.

4.2 Results

SuperThoughts saves CoT length by 20-30% with confidence-based adaptive inference while only having a small accuracy drops (1-2 points on most tasks).

Table 1 presents our main results on three mathematical reasoning benchmarks: MATH500, OlympiadBench, and AMC23. We evaluate both compressor variants (Projection and Transformer) across different inference settings.

Uniform superposition reduces length but hurts accuracy. Without adaptive inference, SuperThoughts reduces CoT length by approximately half but incurs substantial accuracy drops. On **1.5B** (Projection), CoT length decreases by 47–54% across benchmarks, but accuracy drops by 14.7–21.3 points. On **7B** (Projection), compression rates are similar (48–53%), but accuracy drops are notably smaller at 5.6–12.1 points. This *scale effect* suggests that larger models better tolerate aggressive token compression. Both compressor variants exhibit similar behavior, indicating the bottleneck is per-step compute capacity rather than the compression mechanism.

Table 2: Accuracy and average correct CoT length of Qwen-2.5-Instruct-14B trained models[1] on four benchmarks.

	MATH500		OlympiadBench		AMC23		GPQA-Diamond	
	Acc (%)	CoT Len.	Acc (%)	CoT Len.	Acc (%)	CoT Len.	Acc (%)	CoT Len.
Standard CoT	78.0	506.1	38.0	651.1	65.5	723.6	41.5	675.3
SuperThoughts *(Projection Compressor)*								
w/ adaptive, $\tau = 0.999$	77.4	315.7	35.5	441.9	63.0	465.3	41.5	510.2
w/ adaptive, $\tau = 0.9999$	75.8	354.5	35.2	481.8	62.0	504.9	42.9	554.0
w/ adaptive, $\tau = 0.99999$	77.6	408.0	36.6	519.4	66.4	581.2	40.0	615.2
SuperThoughts *(Transformer Compressor)*								
w/ adaptive, $\tau = 0.999$	77.2	315.1	36.7	433.7	63.1	462.5	39.1	502.1
w/ adaptive, $\tau = 0.9999$	78.6	356.9	38.3	486.1	63.9	507.1	43.0	575.3
w/ adaptive, $\tau = 0.99999$	77.0	398.1	37.2	548.0	63.3	562.5	41.2	607.7

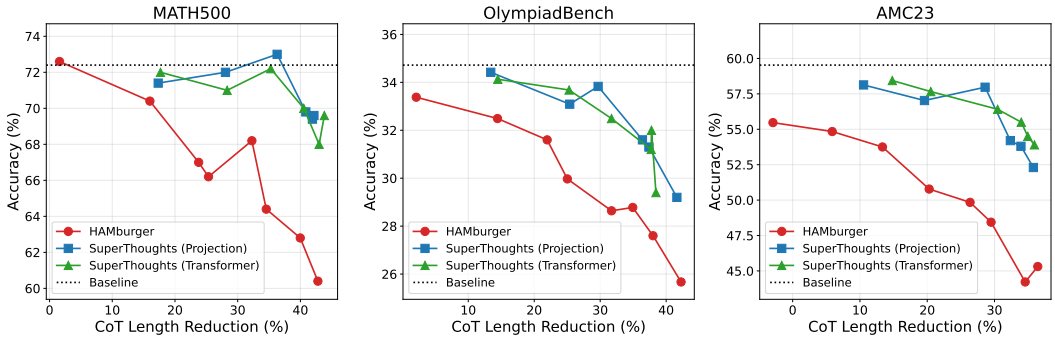


Figure 4: Comparison between SuperThoughts and HAMburger (Liu & Zhang, 2025) on trained Qwen2.5-1.5B-Math-Instruct.

Adaptive inference recovers accuracy. Confidence-based adaptive inference substantially closes the accuracy gap while retaining meaningful CoT length reductions. On **1.5B** (Projection) with $\tau = 0.999$, MATH500 accuracy matches the baseline (73.0% vs. 72.4%) with a 36% CoT reduction. OlympiadBench and AMC23 remain within 0.9–1.6 points of baseline while achieving 29–30% reductions. On **7B** (Projection), $\tau = 0.9999$ offers a balanced trade-off: 30–34% CoT reduction with accuracy within 0.9–2.2 points across all benchmarks.

Notably, while larger τ sometimes achieves best accuracy, increasing τ *does not always improve accuracy*. On **1.5B** (Projection) MATH500, accuracy is the highest at $\tau = 0.999$. Similarly, on **7B** (Projection) AMC23, the best adaptive accuracy occurs at $\tau = 0.9999$. We hypothesize this reflects noise or that $\tau = 0.999$ already provides a sufficiently high confidence threshold.

Compressor comparison. The Projection and Transformer compressors perform similarly. Without adaptive inference, Projection shows a slight edge (e.g., 30.7% vs. 27.9% on 7B OlympiadBench), but this gap disappears with adaptive decoding. Given its simplicity and lower computational cost, the linear projection is the preferred choice.

Comparison with HAMburger. We train HAMburger (Liu & Zhang, 2025) using the same data on Qwen2.5-1.5B and compare against SuperThoughts. We choose confidence $\in \{0.93, 0.95, 0.99, 0.995, 0.999, 0.9995, 0.9999, 1.0\}$ and for SuperThoughts we choose $\tau \in \{0.99, 0.993, 0.995, 0.999, 0.9999, 0.99999\}$. For each configuration we record the CoT length and the accuracy in Figure 4. SuperThoughts achieves higher accuracy at every compression level, and shorter CoT at every accuracy level, than HAMburger.

Beyond mathematical reasoning. To test whether **SuperThoughts** applies to reasoning capabilities beyond Math, we add additional science questions from Guha et al. (2025) and train Qwen2.5-14B-

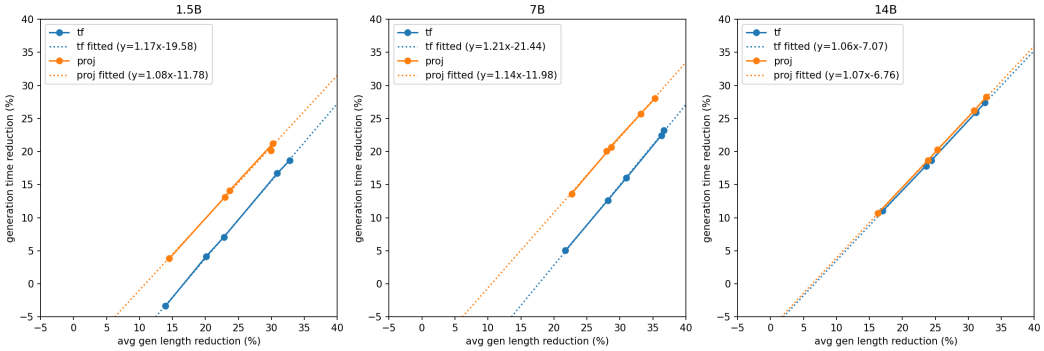


Figure 5: Generation length reduction vs. wall-clock time inference speedup on 1.5B, 7B and 14B models.

Instruct (a non-Math Instruct model) following the same training paradigm. Table 2 shows that **SuperThoughts** works on domains other than Math.

> The gap between theoretical speedup (CoT length reduction) and the actual speedup (wall-clock time reduction) is smaller as the model gets larger.

Inference wall-clock time analysis. We run an additional experiment to measure the theoretical speedup (generation length reduction) vs. actual speedup (generation time reduction). We implement SuperThoughts using nano-vLLM (GeeeekExplorer, 2025) for fast inference. We run MATH500 (500 questions) on the 1.5B, 7B and 14B models, and for each model we choose $\tau \in \{0.999, 0.9995, 0.9999, 0.99995, 0.99999\}$. For each generation configuration, we record the generation length reduction $R_{\text{len}} = (L_{\text{baseline}} - L_{\text{SuperThoughts}})/L_{\text{baseline}}$ and the wall-clock generation time reduction $R_{\text{time}} = (T_{\text{baseline}} - T_{\text{SuperThoughts}})/T_{\text{baseline}}$, where L is the average generation length and T is the generation time. We plot R_{len} versus R_{time} in Figure 5. We can see that the additional overhead from SuperThoughts (compressor, MTP and adaptive fallback) has less influence as the model gets larger. For example, with the projection compressor, a 30.25% reduction in generation length gives a 21.25% reduction in generation time on the 1.5B model, 33.17% gives 25.72% on 7B, and 32.75% gives 28.3% on 14B. This is because such overhead (e.g., additional kernel launches and other CPU activities) takes a smaller percentage of the total generation time as the model gets larger.

5 Discussion

The adaptive mechanism speaks to a broader question: *how should models allocate compute across reasoning steps?* Standard decoding spends the same compute on every token, yet intuitively some steps can be harder than other steps. Our confidence-based adaptive mechanism can be viewed as a simple compute scheduler: superpose when the MTP module is confident, fall back to discrete tokens when it is not fully confident.

We also observe that larger models tolerate aggressive compression better. Under uniform superposition (no adaptive fallback), the 7B model drops $5-12$ points while the 1.5B model drops $14-21$ points. This suggests larger models have enough spare capacity per step to fit two tokens more reliably. If the trend holds at greater scale, even larger models might tolerate superposing three or more tokens, or need fewer fallbacks to discrete decoding.

Finally, we train the MTP module from scratch because Qwen2.5 lacks a native MTP module. Recent models like Qwen3-Next (Team, 2025) and and MiMo (Xiaomi et al., 2025) are pre-trained with native MTP modules. Starting from a native MTP module would simplify training and likely improve results, since the MTP module is already aligned with the backbone. As native MTP becomes standard, SuperThoughts becomes easier to apply.

[1]Note that this is a non-Math model so the accuracies on Math benchmarks are lower than the 7B-Math model in Table 1; for the Projection compressor we add an additional RMSNorm after the compressor.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the implementation of the method and baselines. We used Qwen2.5 to generate CoT traces as described in Section 4. We used generative AI tools to polish writing at the sentence level.

We have not used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, clean and reformat dataset, support qualitative and thematic data analysis, interpret results; and help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs and assist with translation are not applicable to this work.

We reviewed all AI-assisted work. We review the code written by AI and write additional tests to ensure code correctness. We review all edits on the draft made by AI and make sure that the readability is improved. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=4jdIxXBNve.

Kwangjun Ahn, Alex Lamb, and John Langford. Efficient joint prediction of multiple future tokens. *arXiv preprint arXiv:2503.21801*, 2025.

Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and Nick Haber. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models, 2025. URL https://arxiv.org/abs/2502.17387.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2296–2318. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bachmann24a.html.

Gabriele Berton, Jayakrishnan Unnikrishnan, Son Tran, and Mubarak Shah. Compllm: Compression for long context q&a, 2025. URL https://arxiv.org/abs/2509.19228.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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

Jingcheng Deng, Liang Pang, Zihao Wei, Shichen Xu, Zenghao Duan, Kun Xu, Yang Song, Huawei Shen, and Xueqi Cheng. Latent reasoning in llms as a vocabulary-space superposition, 2025. URL https://arxiv.org/abs/2510.15522.

Yair Feldman and Yoav Artzi. Simple context compression: Mean-pooling and multi-ratio training. *arXiv preprint arXiv:2510.20797*, 2025.

GeeeekExplorer. Nano-vllm. https://github.com/GeeeekExplorer/nano-vllm, 2025.

Angeliki Giannou, Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. Stoic reasoner: Dual-mode transformers that compress to think and decompress to speak. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=2RTdyYfa0v.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Halil Alperen Gozeten, Muhammed Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=sTPKDKn5ig.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

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, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024.

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. *NeurIPS*, 2021.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL https://arxiv.org/abs/2507.07955.

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.

Adit Jain and Brendan Rappazzo. Learning to reason with mixture of tokens, 2025. URL https://arxiv.org/abs/2509.21482.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models, 2023. URL https://arxiv.org/abs/2310.05736.

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*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jingyu Liu and Ce Zhang. Hamburger: Accelerating llm inference via token smashing, 2025. URL https://arxiv.org/abs/2505.20438.

MAA. AMC 2023 problems, 2023. URL https://artofproblemsolving.com/wiki/index.php/2023_AMC_12A_Problems.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=2DtxPCL3T5.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

Bowen Peng, Théo Gigant, and Jeffrey Quesnelle. Efficient pre-training with token superposition. *arXiv preprint arXiv:2605.06546*, 2026.

Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, Tianyu Zhang, He Xing, Zheng Zhang, Yuxuan Song, Tianyu Zheng, Zhiyuan Zeng, Chenghua Lin, Ge Zhang, and Wenhao Huang. Dynamic large concept models: Latent reasoning in an adaptive semantic space, 2026. URL https://arxiv.org/abs/2512.24617.

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 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. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Shannon Zejiang Shen, Rulin Shao, Chenyu Wang, Songlin Yang, Vincent-Pierre Berges, Gargi Ghosh, Pang Wei Koh, Luke Zettlemoyer, Yoon Kim, Jason E Weston, David Sontag, and Wen tau Yih. Hybridcot: Interleaving latent and text chain-of-thought for efficient reasoning. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025a. URL https://openreview.net/forum?id=NRGRrHmq1H.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.36. URL https://aclanthology.org/2025.emnlp-main.36/.

Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

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. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=hYfOPXrbUr.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Jian Luan, and Ruihua Song. Think silently, think fast: Dynamic latent compression of llm reasoning chains, 2025. URL https://arxiv.org/abs/2505.16552.

Yao Tang, Li Dong, Yaru Hao, Qingxiu Dong, Furu Wei, and Jiatao Gu. Multiplex thinking: Reasoning via token-wise branch-and-merge. *arXiv preprint arXiv:2601.08808*, 2026.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Junhong Wu, Jinliang Lu, Zixuan Ren, Gangqiang Hu, Zhi Wu, Dai Dai, and Hua Wu. Llms are single-threaded reasoners: Demystifying the working mechanism of soft thinking, 2025. URL https://arxiv.org/abs/2508.03440.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3351–3363, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.165. URL https://aclanthology.org/2025.emnlp-main.165/.

LLM Xiaomi, Bingquan Xia, Bowen Shen, Dawei Zhu, Di Zhang, Gang Wang, Hailin Zhang, Huaqiu Liu, Jiebao Xiao, Jinhao Dong, et al. Mimo: Unlocking the reasoning potential of language model–from pretraining to posttraining. *arXiv preprint arXiv:2505.07608*, 2025.

Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. Hybrid latent reasoning via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=LjtgTpWH71.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. LightThinker: Thinking step-by-step compression. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13307–13328, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.673. URL https://aclanthology.org/2025.emnlp-main.673/.

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*, 2025b.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=UdOEZgWJLc.

Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Text generation beyond discrete token sampling. *arXiv preprint arXiv:2505.14827*, 2025.

A Discussion on Baseline

We report the standard discrete CoT as our baseline choice. In this section we discuss other related methods and why we didn't choose them as baselines.

Methods that reduce the discrete tokens. Methods like TokenSkip (Xia et al., 2025) cuts the number of discrete CoT tokens by finetuning the model on a more concise reasoning trace. TokenSkip operates entirely within the discrete token space – it reduces the number of tokens generated but still decodes one token per forward pass. **SuperThoughts**, by contrast, reduces the number of forward passes themselves by decoding multiple tokens per step, targeting a different axis of efficiency. The two approaches are complementary and could in principle be combined.

MTP for self-speculative decoding. MTP module can be used as a draft model to speculatively predict future tokens and verify with the target model. However, speculative decoding does not reduce total FLOPs – the main model must still perform full forward passes to populate KV cache entries for every accepted token. It can speed up inference in low-utilization regimes with small batch sizes by increasing GPU utilization, but when serving large batches at high throughput, the GPU is already compute-bound and no speed-up is achieved. On the other hand, **SuperThoughts** reduces total FLOPs by adaptively generating multiple tokens per forward pass, in contrast to standard autoregressive decoding which produces a single token per step. Therefore, speculative decoding is not a suitable baseline for **SuperThoughts**, as it targets a fundamentally different bottleneck – latency under low utilization – rather than reducing total FLOPs, which is the focus of our work.

Latent reasoning. Latent reasoning methods like COCONUT (Hao et al., 2024), CODI (Shen et al., 2025b) and CoLaR (Tan et al., 2025) directly use the latent from the last layer as the input to the next reasoning step. However, these methods are designed for tasks and training regimes that involve short reasoning sequences (e.g., 20–60 tokens), and we find that they do not translate effectively to our settin. For example we tried to train CODI in our setting and the trained model can hardly produce a correct answer (accuracy 0.1%). We conduct experiment with Soft Thinking (Zhang et al., 2025b) that aggregates token embeddings from multiple candidates within a single step. However, Table 3 shows that Soft Thinking is not significantly better than Standard CoT (except 1.5B on AMC23) and the CoT length is longer.

Table 3: Accuracy and average correct CoT length of Qwen-2.5-Math-Instruct-1.5B/7B models on three benchmarks, comparing the Standard CoT baseline against Soft Thinking.

		MATH500		OlympiadBench		AMC23	
		Acc (%)	CoT Len.	Acc (%)	CoT Len.	Acc (%)	CoT Len.
1.5B	Standard CoT	72.4	494.1	34.72	620.5	59.53	660.5
	Soft Thinking	72.8	570.4	33.68	780.6	66.84	838.6
7B	Standard CoT	83.0	538.6	36.35	654.6	69.22	749.4
	Soft Thinking	82.0	622.0	37.39	868.7	64.61	812.7

B Inference Cost Analysis

In this subsection we analyze how much compute can **SuperThoughts** save compared to standard discrete CoT decoding. For simplicity we assume batch size $B = 1$; the analysis extends directly to $B > 1$ since B cancels in the ratio $C_{\text{ST}}/C_{\text{base}}$, and **SuperThoughts** supports batched generation as token superposition operates independently within each sequence. Let L_q be the question (prompt) length, L_c be the CoT length and L_a be the response length. The inference cost can be split into prefilling cost and decoding cost, while at prefilling stage the model process the question part one time and at the decoding stage the model generates CoT and answer autoregressively. For reasoning intensive tasks like Math, usually $L_c + L_a \gg L_q$, meaning that the decoding cost dominates. Also since $L_c \gg L_a$, for simplicity, ignore prompt and answer: let $L_q = 0, L_a = 0$ and $L = L_c$ be the context length.

We decompose compute into (1) *core attention* (the $QK^\top$ and Attn $\cdot\, V$ matmuls), which scales as $O(L^2H)$, and (2) *linear layers* (FFN + QKV/output projections), which scale as $O(LH^2)$. Let H be the embedding dimension and T be the number of layers. For GPT-2[2] (Radford et al., 2019) Transformer block, the core attention cost (number of multiplications) is $2TL^2H$ and the linear layers cost is $12TLH^2$.

For SuperThoughts, let S be the number of superposition steps and let L' be the CoT length, which includes both (1) steps when we superpose tokens and (2) steps when we use discrete tokens during adaptive inference. Note that without adaptive inference, $L' = S = L/2$ because we superpose at each CoT step. we break down the inference cost for each module:

- **(Main module)** Core attention cost: $2TL'^2H$. Linear layers cost: $12TL'H^2$.
- **(MLP module)** Core attention cost: $2L'^2H$. Linear layers cost: $12L'H^2$. Additional projection cost: $3L'H^2$.
- **(Projection Compressor)** Projection cost: $2SH^2$.
- **(Transformer Compressor)** Transformer cost: $8SH + 24SH^2$.

Note that for the Projection Compressor, the cost is $2SH^2$ not $2L'H^2$ because if we use adaptive inference and when we reject the MTP token, we do not compress tokens at the next CoT step. Ignoring the additional minor projection cost, a simple mental model could be that SuperThoughts has $\frac{L'}{L}$ CoT steps as the standard discrete model but each step costs $(T+1)/(T)$ compute. Note that since the core attention cost is quadratic in L and L', this can underestimate the compute saved depending on how large is L.

For SuperThoughts with Projection Compressor, ignoring these small projection-style overheads, a simple mental model is that SuperThoughts runs L' decoding steps instead of L, but each step executes a T-layer backbone plus an extra 1-layer MTP, i.e., a per-step multiplier of $(T+1)/T$. This multiplier applies to both the dense-matmul term ($\propto LH^2$) and the core attention-mixing term ($\propto L^2H$). Approximating with the linear term gives

$$\frac{C_{\text{ST}}}{C_{\text{base}}} \approx \frac{L'}{L} \cdot \frac{T+1}{T},$$

which typically overestimates this ratio (and thus underestimates compute savings), since the core attention term scales as $\frac{T+1}{T}\left(\frac{L'}{L}\right)^2$.

[2]Here we use GPT-2 architecture for simplicity. For other architecture like Qwen, only the constant term changes and the analysis still holds.

C ALGORITHMS FOR DATASET PREPARATION

Algorithm 1: SuperThoughts Dataset Preparation

Input: Question q, answer a, chain token ids $\mathbf{c} = (c_1, \ldots, c_N)$, token probabilities $\mathbf{p} = (p_1, \ldots, p_N)$, window size k, special treatment mode $\mathcal{M} \in \{\texttt{prob}, \texttt{none}\}$
Output: NTP tensors $(\mathbf{x}, \mathbf{G}, \mathbf{V}, \mathbf{y})$ and MTP tensors $(\mathbf{x}_{\text{mtp}}, \mathbf{G}_{\text{mtp}}, \mathbf{V}_{\text{mtp}}, \mathbf{y}_{\text{mtp}})$

// Step 1: Adaptive token selection (mode-specific, see Algorithms 2--3)
$\mathbf{s} \leftarrow \texttt{AdaptiveSelect}_{\mathcal{M}}(\mathbf{c}, \mathbf{p})$;
// Step 2: Window-aligned padding
$\tilde{\mathbf{c}}, \mathbf{m} \leftarrow \texttt{WindowAlignedPad}(\mathbf{c}, \mathbf{s}, k, \mathcal{M})$;
// Step 3: Build NTP sequence
$\mathbf{x}, \mathbf{G}, \mathbf{V}, \mathbf{y} \leftarrow \texttt{BuildSequence}(q, a, \tilde{\mathbf{c}}, \mathbf{m}, k, \texttt{false})$;
// Step 4: Build MTP sequence (shift chain right by $k-1$)
$\tilde{\mathbf{c}}_{\text{mtp}} \leftarrow (\underbrace{\text{COT_PAD}, \ldots, \text{COT_PAD}}_{k-1}, \tilde{c}_1, \ldots, \tilde{c}_{|\tilde{\mathbf{c}}|})$;
$\mathbf{m}_{\text{mtp}} \leftarrow (\underbrace{\texttt{true}, \ldots, \texttt{true}}_{k-1}, m_1, \ldots, m_{|\mathbf{m}|})$;
$\mathbf{x}_{\text{mtp}}, \mathbf{G}_{\text{mtp}}, \mathbf{V}_{\text{mtp}}, \mathbf{y}_{\text{mtp}} \leftarrow \texttt{BuildSequence}(q, a, \tilde{\mathbf{c}}_{\text{mtp}}, \mathbf{m}_{\text{mtp}}, k, \texttt{true})$;
return $(\mathbf{x}, \mathbf{G}, \mathbf{V}, \mathbf{y})$, $(\mathbf{x}_{\text{mtp}}, \mathbf{G}_{\text{mtp}}, \mathbf{V}_{\text{mtp}}, \mathbf{y}_{\text{mtp}})$;

Algorithm 2: ADAPTIVESELECT$_{\texttt{prob}}$: Fraction-Based Probability Selection

Input: Token ids $\mathbf{c} = (c_1, \ldots, c_N)$, token probabilities $\mathbf{p} = (p_1, \ldots, p_N)$
Output: Boolean mask $\mathbf{s} = (s_1, \ldots, s_N)$

Sample $\alpha \sim \mathcal{U}[\alpha_{\min}, \alpha_{\max}]$; // Here $0 \le \alpha_{\min} < \alpha_{\max} \le 1$ is a fraction
$m \leftarrow \lfloor \alpha \cdot N \rfloor$;
$\mathcal{I} \leftarrow$ indices of the m smallest values in $\mathbf{p}$;
$s_i \leftarrow \mathbb{1}[i \in \mathcal{I}] \quad \forall i \in \{1, \ldots, N\}$;
return $\mathbf{s}$;

Algorithm 3: ADAPTIVESELECT$_{\texttt{none}}$: No Selection (Baseline)

$s_i \coloneqq \texttt{false} \quad \forall i$;

Algorithm 4: WINDOWALIGNEDPAD: Window-Aligned Padding

Input: Token ids $\mathbf{c}$, special mask $\mathbf{s}$, window size k, mode $\mathcal{M}$
Output: Padded ids $\tilde{\mathbf{c}}$, padding mask $\mathbf{m}$ ($m_i = \texttt{true} \Rightarrow$ inserted pad)

```
pad_right ← (M ≠ prob-frac);    // isolate specials in own window?
c̃ ← [], m ← [], j ← 0;          // j: position within current window
for i ← 1 to |c| do
    if s_i then
        if j ≠ 0 then                          // pad to finish current window
            c̃ += [COT_PAD]^(k−j), m += [true]^(k−j);
        c̃ += c_i, m += false;
        j ← 1;
        if pad_right then                      // fill rest of window
            c̃ += [COT_PAD]^(k−1), m += [true]^(k−1);
            j ← 0;
    else
        c̃ += c_i, m += false;
        j ← (j + 1) mod k;
return c̃, m;
```

Algorithm 5: BUILDSEQUENCE: Assemble Input and Target Tensors

Input: Question q, answer a, padded chain $\tilde{\mathbf{c}}$, padding mask $\mathbf{m}$, window size k, flag `is_mtp`
Output: Input ids $\mathbf{x}$, grouped CoT inputs $\mathbf{G} \in \mathbb{Z}^{L \times k}$, valid masks $\mathbf{V} \in \{0,1\}^{L \times k}$, targets $\mathbf{y}$

```
// Tokenize with chat template
t ← CHATTEMPLATE(q, a);                 // contains <think> ... </think>
b ← index of <think> in t;
e ← index of </think> in t;
// Group padded chain into k-windows
G_l ← (c̃_{(l−1)k+1}, ..., c̃_{lk}) for l = 1, ..., L_cot;
V_l ← (¬m_{(l−1)k+1}, ..., ¬m_{lk}) for l = 1, ..., L_cot;
// Construct input ids: replace reasoning span with
   placeholders
x ← [t_{1:b}, PAD, ..., PAD (L_cot times), t_{e:|t|}];
r_start ← b + 1,  r_end ← b + L_cot;
G[r_start : r_end] ← (G_1, ..., G_{L_cot});     // elsewhere filled with PAD
V[r_start : r_end] ← (V_1, ..., V_{L_cot});     // elsewhere false
cot_mask_i ← ⋁_{j=1}^{k} V_{i,j};  // true if position i has any valid CoT token
// Construct targets
for i ← 1 to L do
    if cot_mask_i then
        y_i ← first G_{l(i)+1}[j] such that G_{l(i)+1}[j] ≠ COT_PAD; // next window's first
          valid token
    else if is_mtp or i ≤ r_end then
        y_i ← IGNORE;    // mask out question region; MTP masks answer
          too
    else
        y_i ← x_{i+1};       // standard next-token prediction for answer
if ¬ is_mtp then
    y_b ← G_1[1];                // at <think>, predict first CoT token
return x, G, V, y;
```