Toward Efficient Reasoning of Large Language Models via Latent Concept-Pyramid Modeling

Sijia Chen **Ningxin Su**
The Hong Kong University of Science and Technology (Guangzhou)
sjiachen@hkust-gz.edu.cn ningxinsu@hkust-gz.edu.cn

Abstract

Large language models (LLMs) perform reasoning via lengthy token-by-token generation, incurring substantial inference cost. While recent methods compress this process by enabling LLMs to reason in a latent space, they still rely on next-vector generation for sequential logic and on flat representations. We therefore present Latent Concept-Pyramid Modeling (*L*CP), a new paradigm that reformulates LLM reasoning as hierarchical next-level concept generation in a coarse-to-fine manner. Specifically, *L*CP generates the pyramid level by level: a single apex concept encodes the most abstract semantics, and each subsequent level refines its predecessor into finer-grained concepts that together capture distinct CoT segments at one granularity. Such next-level concept prediction approximates the reasoning trace in a manner analogous to advancing from a skeletal outline, through broad structural forms, down to local details. Experimental evaluations on mathematical and coding reasoning benchmarks demonstrate that Qwen2.5 and Qwen3 models fine-tuned with *L*CP achieve significant reductions in both token cost and inference latency while maintaining competitive accuracy. *L*CP further transfers zero-shot to unseen datasets of the same task. Moreover, the pyramid's hierarchical, multi-grained representational capacity enables reconstruction of the original CoT in textual form, endowing *L*CP with interpretability. The source code and comprehensive training details are publicly available.[1]

1 Introduction

Large language models (LLMs) can perform reasoning, yet this performance comes at the cost of generating lengthy Chain-of-Thought (CoT) sequences that are often redundant [Jiang et al., 2026, Wu et al., 2026] and incur substantial inference overhead [Sui et al., 2025, Chen et al., 2025a]. Toward efficient reasoning, a promising direction is to shift reasoning from explicit language space to a latent space [Wang, 2026, Yu et al., 2026], where intermediate thoughts are represented by fewer compact vectors carrying denser, more generalized semantics.

A dominant line of latent reasoning preserves autoregressive generation by feeding hidden states directly back as subsequent inputs [Hao et al., 2025, Zhu and Sui, 2026], condensing traces into a few latent steps [Tan et al., 2025, Zeng et al., 2026], or mixing latent vectors with discrete language tokens [Goyal et al., 2024b, Yue et al., 2025, Su et al., 2025, Chu et al., 2026]. Despite these variations, all retain a *sequential* structure in which each reasoning state depends on its predecessor, thereby inheriting the latency of token-by-token generation. An orthogonal line instead compresses the *entire* CoT into a *compact, flat* representation that directly conditions the answer, either through encoding or distillation of full traces [Jiang et al., 2025, 2026] or through latent-memory optimization and soft thought tokens [Liu et al., 2025, Xu et al., 2025]. Although faster at inference, these methods

[1] https://github.com/AgenticFinLab/Latent-Concept-Reasoning

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

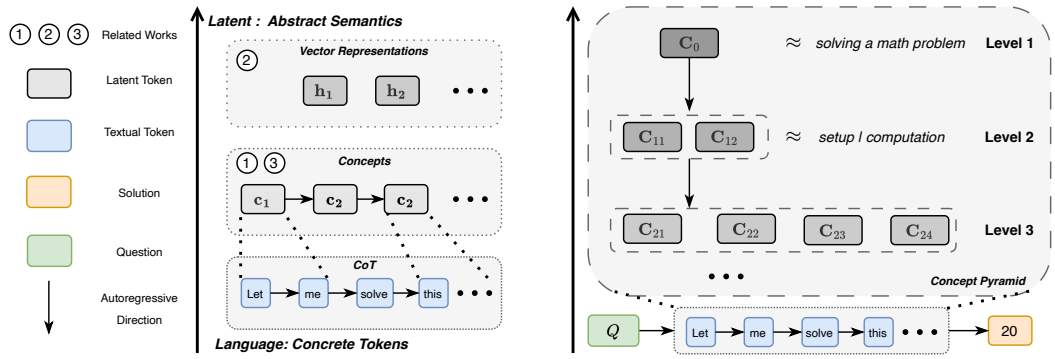


Figure 1: Overview of the reasoning process of related work and our *Latent Concept-Pyramid Modeling* (*L*CP). Instead of compressing the CoT at a single granularity via sequential latent tokens or flat representations, *L*CP builds a hierarchical concept pyramid and thus enables the LLM to reason level by level in a coarse-to-fine manner. See Section 3 for the details of related works.

collapse the trajectory into a uniform granularity whose coarse semantics often fail to preserve the multi-step logical dependencies required for complex reasoning.

One natural way to address this dilemma is *concept-level autoregression* [Liu et al., 2026, Barrault et al., 2025, Qu et al., 2026], which represents the CoT as a compact sequence of abstract concepts. However, these concepts remain inherently flat rather than organized into the abstract-to-concrete hierarchies that human cognition [Koechlin and Summerfield, 2007] and coarse-to-fine representation learning [Kusupati et al., 2022, Tian et al., 2024] identify as essential for building and decoding complex structures.

Therefore, we present *Latent Concept-Pyramid Modeling* (*L*CP), a new paradigm that compresses the reasoning trace into hierarchical concepts in a coarse-to-fine manner and performs reasoning via *next-level concept prediction* (Figure 1). Specifically, *L*CP represents the complete reasoning trace at progressively finer granularities, from a single generalized apex concept down to multiple detail-oriented base concepts. Crucially, the concept pyramid, in which each level captures one granularity of the CoT, can be decoded back into textual form for *L*CP's interpretability.

To implement this, *L*CP introduces two modules: a *Concept Pyramid Builder* (*LCP-b*) and a *Concept Predictor* (*LCP-p*). *LCP-b* extracts the pyramid level by level through residual decomposition: per-level learnable queries attend to the residual left by coarser levels and produce the corresponding concepts, whose aggregated reconstruction is subtracted to form the next-level residual. Soft boundaries and an intra-level ordering constraint preserve the underlying reasoning logic without committing to hard CoT partitions. *LCP-p* reuses a causal decoder-only LLM backbone that takes the question as input and predicts concepts level by level until reaching the reasoning answer.

Extensive experiments on mathematical reasoning and code generation show that LLMs equipped with *L*CP achieve significant reductions in token cost and inference latency while maintaining competitive accuracy compared to strong latent-reasoning baselines. The compression rate is further adjustable by pre-specifying the pyramid depth. Crucially, unlike prior approaches that work purely on latent vectors, *L*CP makes the reasoning trajectory readable, as the concept pyramid can be decoded back into textual CoT by a simple decoder-only LLM. Additionally, *L*CP transfers zero-shot to unseen datasets of the same task.

2 Related Work

For **step-by-step latent reasoning**, a dominant paradigm compresses the reasoning trace by feeding hidden states directly back as autoregressive inputs, as in Coconut [Hao et al., 2025] and subsequent works [Zhu and Sui, 2026, Gozeten et al., 2026]. Alternative methods condense traces into latent steps via reinforcement learning [Tan et al., 2025], token importance [Zeng et al., 2026], or elastic-depth looped transformers [Chen et al., 2025b, Saunshi et al., 2025, Jeddi et al., 2026]. Other approaches integrate latent and textual reasoning by mixing discrete VQ-VAE codes with language [Su et al., 2025], inserting span-level latent pauses or implicit tokens [Goyal et al., 2024a, Pfau et al., 2024, Chu et al., 2026, He et al., 2025], or adaptively routing between latent and verbal-

ized steps [Yue et al., 2025, Wang et al., 2025]. Despite these variations, all retain a *flat sequential structure* in which each reasoning state depends on its predecessor, thereby limiting the reasoning efficiency.

Other leading methods **compress the reasoning trace** into latent representations that directly condition the final answer. For instance, LLMs learn to perform non-autoregressive silent-thought [Jiang et al., 2025] or adopt compact representations [Xu et al., 2025, Jiang et al., 2026], while other frameworks compress sequential reasoning into jointly optimized latent vectors [Shen et al., 2025, Wu et al., 2025]. Alternative works reason with latent representations learned from KV cache trajectories [Liu et al., 2025, Kuzina et al., 2026]. Although more efficient, they present the full reasoning trajectory with *fixed, flat representations*, which may lack the granularity to preserve the reasoning logic and dependencies across intermediate steps needed for complex multi-step problems.

Another strategy to balance these paradigms is to reason in a higher-abstraction space, such as **concept-level reasoning for compression**. Next-concept prediction methods train transformers to autoregressively predict latent units spanning multiple tokens, grounded either in discrete concept vocabularies constructed via vector quantization [Liu et al., 2026] or in continuous sentence-level spaces [An et al., 2024, Barrault et al., 2025]. Although not explicitly designed for compression, Qu et al. [2026] demonstrates that CoTs can be represented by abstract concepts, each capturing variable-length token segments. Furthermore, the "next-scale prediction" mechanism in VAR [Tian et al., 2024] shows that complex structures can be effectively modeled through hierarchical representations. Building on concept-level reasoning, our method organizes these representations into a multi-granularity hierarchy (i.e., a concept pyramid), achieving inference efficiency while enriching semantic granularity via "next-level concept generation".

3 Preliminary

A decoder-only LLM $f_\theta(\cdot)$ performs reasoning by generating a chain-of-thought (CoT) $\mathbf{S} = [s_1, s_2, \ldots, s_T]$ to produce a solution $\mathbb{A}$ for a question $\mathbb{Q}$. Under autoregressive generation, $p(\mathbf{S}, \mathbb{A} \mid \mathbb{Q}) = \prod_{t=1}^{T} p(s_t \mid \mathbb{Q}, s_{<t}) \cdot p(\mathbb{A} \mid \mathbb{Q}, \mathbf{S})$, where each token s_t is conditioned on previous tokens $s_{<t}$. Reasoning in the latent space is an effective way to improve efficiency.

Sequential latent reasoning ① feeds hidden states back as autoregressive inputs [Goyal et al., 2024b, Hao et al., 2025, Zhu and Sui, 2026, Gozeten et al., 2026]. The model computes a latent state $\boldsymbol{h}_t = f(\boldsymbol{h}_{<t}, \mathbb{Q})$, yielding a sequential process $p(\mathbf{H}, \mathbb{A} \mid \mathbb{Q}) = \prod_{t=1}^{M} p(\boldsymbol{h}_t \mid \mathbb{Q}, \boldsymbol{h}_{<t}) \cdot p(\mathbb{A} \mid \mathbb{Q}, \mathbf{H})$, where $\mathbf{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_M]$ and typically $M < T$ since each latent step encodes multiple tokens. A more efficient alternative is the **compression of the reasoning chain** ②, which encodes the entire CoT into a compact representation $\mathbf{z} = f_{\theta_e}(\mathbf{S})$ [Jiang et al., 2025, 2026], directly predicting the solution via $p(\mathbb{A} \mid \mathbb{Q}, \mathbf{z})$. This avoids sequential generation but collapses the full trajectory into *fixed representations* with flat granularity.

Concept-level reasoning ③ aims to present $\mathbf{S}$ as $\mathbf{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_J]$ concepts, where each concept vector $\boldsymbol{c}_j$ represents one CoT segment [Qu et al., 2026], and concepts are predicted autoregressively in sequence [Liu et al., 2026, Barrault et al., 2025]: $p(\mathbf{C}, \mathbb{A} \mid \mathbb{Q}) = \prod_{j=1}^{J} p(\boldsymbol{c}_j \mid \mathbb{Q}, \boldsymbol{c}_{<j}) \cdot p(\mathbb{A} \mid \mathbb{Q}, \mathbf{C})$ with $J \ll T$.

To enable this, token-level CoTs, typically encoded as $\mathbf{e}_\mathbf{S} = [e_1, e_2, \ldots, e_T] \in \mathbb{R}^{T \times d_e}$, need to be compressed into concept sequences by first splitting CoT with non-uniform information distribution into *hard* segmentations $[a_j, b_j]$, where the j-th segment spans token indices a_j through b_j. DLCM [Qu et al., 2026] implements this by detecting boundaries through normalized dissimilarity between adjacent tokens. Positions where consecutive tokens exhibit high dissimilarity are classified as boundaries. The concept vector is obtained by mean-pooling within this hard interval, where every token belongs to exactly one segment, so $[a_j, b_j] \cap [a_{j'}, b_{j'}] = \emptyset$ for $j \neq j'$.

The general objective is to minimize $\mathcal{L} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{CE}}$, where $\mathcal{L}_{\text{recon}}$ penalizes the discrepancy between the CoT and its concept-based reconstruction [Qu et al., 2026], and $\mathcal{L}_{\text{CE}} = -\log p(\mathbb{A} \mid \mathbb{Q}, \mathbf{C})$ measures the likelihood of the correct solution given the concept sequence.

Limitations of Existing Paradigms mostly arise from their reliance on sequential generation with flat latent representations: whether operating on hidden tokens, latent steps, or concepts, existing paradigms optimize a next-step objective that enforces only local adjacency consistency, a regime

known to induce representation degeneration and anisotropic embedding geometry in autoregressive models [Gao et al., 2019, Ethayarajh, 2019, Su et al., 2022]. This makes representations reside at a single abstraction level while losing multi-level hierarchy. Empirically, the resulting flatness can render additional latent reasoning steps counterproductive: SemCoT [He et al., 2025], for instance, degrades from 9.83 to 2.50 on GSM8K as the number of latent steps grows from 1 to 5 (Table 2).

Hard segmentation imposes discrete boundaries b_j on a continuous reasoning process, yet **logical steps rarely align with token boundaries**. Within each segment, mean-pooling yields $\boldsymbol{c}_j = \alpha_j \bar{e}_j^{\text{core}} + (1-\alpha_j)\bar{e}_j^{\text{filler}}$, where $\alpha_j = |\mathcal{S}_j^{\text{core}}|/|\mathcal{S}_j|$ is the fraction of reasoning-bearing tokens in segment j, and $\bar{e}_j^{\text{core}}$, $\bar{e}_j^{\text{filler}}$ are the mean embeddings of the core and filler tokens, respectively. As $\alpha_j \to 0$, the concept degenerates to filler average, diluting the reasoning logic. Compounded by the flat latent geometry, such dilution erases intra-concept inference steps and leaves adjacent concepts without explicit logical bridges.

Sequential approaches preserve dependencies but suffer linear inference cost; direct-compression methods achieve constant-time inference but abandon structure entirely. No existing paradigm **simultaneously achieves** *compactness*, *dependency preservation*, and *multi-scale hierarchy*.

Since latent reasoning operates on compact vectors decoupled from the language space, intermediate states such as $\mathbf{H}$, $\boldsymbol{z}$, or $\mathbf{C}$ are **not directly human-readable**. Unlike explicit CoT, which provides a step-by-step audit trail, flat latent representations offer no mechanism to decode back into textual form for reasoning trajectory inspection.

4 Method

In this section, we propose *Latent Concept-Pyramid Modeling* (*L*CP), which enables *next-level concept prediction* of LLMs, shifting reasoning from sequential token generation or latent/concept generation [Hao et al., 2025, Barrault et al., 2025, Liu et al., 2026, Qu et al., 2026] to *multi-level concept generation* in the spirit of coarse-to-fine next-scale prediction [Tian et al., 2024]. That is, *L*CP builds a concept pyramid with a hierarchical structure $\mathbf{P} = \{\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_{K-1}\}$, where each $\mathbf{C}_k = [\boldsymbol{c}_{k,0}, \ldots, \boldsymbol{c}_{k,J_k-1}] \in \mathbb{R}^{J_k \times d_c}$ contains J_k concept vectors at progressively finer granularity, with the per-level width set to $J_k = 2^k$ so that pyramid capacity grows geometrically from the apex to the base. This yields:

$$p(\mathbf{P}, \mathbb{A} \mid \mathbb{Q}) = \prod_{k=0}^{K-1} p(\mathbf{C}_k \mid \mathbb{Q}, \mathbf{C}_{<k}) \cdot p(\mathbb{A} \mid \mathbb{Q}, \mathbf{P}),$$

where the CoT segment $[a_{k,j},\, b_{k,j}]$ assigned to each $\boldsymbol{c}_{k,j}$ satisfies $a_{k,j}, b_{k,j} \in [0, T]$ and adjacent segments may overlap, i.e., a *soft* boundary.

To this end, a *Concept Pyramid Builder* (*LCP-b*) extracts $\mathbf{P}$ from the ground-truth CoT, while a *Concept Predictor* (*LCP-p*) autoregressively generates $\mathbf{P}$ from $\mathbb{Q}$ alone: the apex $\mathbf{C}_0$ encodes the global outline, intermediate levels encode sub-goals, and the finest level $\mathbf{C}_{K-1}$ encodes local details, so that the solution $\mathbb{A}$ is derived from $\sum_k J_k$ concept vectors in only K sequential passes, rather than from T token-by-token steps.

4.1 Building Concept Pyramids toward Next-Level Prediction

LCP-b builds $\mathbf{P}$ from the CoT encodings $\mathbf{e}_{\mathbf{S}} \in \mathbb{R}^{T \times d_e}$ by first projecting the encodings into the concept space via $\mathbf{U}^{(0)} = \mathrm{LN}(\boldsymbol{W}_p\, \mathbf{e}_{\mathbf{S}}) \in \mathbb{R}^{T \times d_c}$, and then extracts concepts level by level through a residual decomposition, as summarized in Algorithm 1.

Soft boundaries. At each level k, learnable queries $\mathbf{Q}_k = [\boldsymbol{q}_{k,0}, \ldots, \boldsymbol{q}_{k,J_k-1}] \in \mathbb{R}^{J_k \times d_c}$ compete for residual information through scaled softmax attention with a learnable temperature τ: $\alpha_{k,j,t} = \frac{\exp\left(\boldsymbol{q}_{k,j}^\top \boldsymbol{u}_t^{(k)} / (\sqrt{d_c}\,\tau)\right)}{\sum_{t'=1}^{T} \exp\left(\boldsymbol{q}_{k,j}^\top \boldsymbol{u}_{t'}^{(k)} / (\sqrt{d_c}\,\tau)\right)}$, where $\boldsymbol{u}_t^{(k)}$ denotes the t-th row of $\mathbf{U}^{(k)}$. Unlike hard segmentation [Qu et al., 2026] that assigns each token to a single segment, the continuous weights $\alpha_{k,j,t}$ induce *soft boundaries*: a token may contribute to multiple concepts with varying strength, and the transition across adjacent concepts is smooth. Each query aggregates its attended context into a concept vector $\boldsymbol{c}_{k,j} = \boldsymbol{W}_k^{(u)} \left(\sum_{t=1}^{T} \alpha_{k,j,t}\, \boldsymbol{u}_t^{(k)} \right)$, with per-level projection $\boldsymbol{W}_k^{(u)} \in \mathbb{R}^{d_c \times d_c}$.

Algorithm 1 Concept Pyramid Builder (*LCP-b*)

Require: CoT encodings $\mathbf{e_S}$, Pyramid Level K

// project CoT to concept space
$\mathbf{U}^{(0)} \leftarrow \mathrm{LN}\big(\boldsymbol{W}_p\,\mathbf{e_S}\big)$
for $k = 0$ **to** $K-1$ **do**
  // soft boundaries
  $\alpha_{k,j,t} \leftarrow \mathrm{softmax}_t\big(\boldsymbol{q}_{k,j}^{\top}\boldsymbol{u}_t^{(k)}/(\sqrt{d_c}\,\tau)\big)$
  $\boldsymbol{c}_{k,j} \leftarrow \boldsymbol{W}_k^{(u)}\sum_{t=1}^{T}\alpha_{k,j,t}\,\boldsymbol{u}_t^{(k)},\ \forall j \in J_k$
  // rank-2^k residual update
  $\boldsymbol{R}_k \leftarrow \boldsymbol{A}_k^{\top}\mathbf{C}_k;\quad \mathbf{U}^{(k+1)} \leftarrow \mathbf{U}^{(k)} - \boldsymbol{R}_k$
  // expected positions for $\mathcal{L}_{order}$
  $\bar{t}_{k,j} \leftarrow \sum_{t=1}^{T}\alpha_{k,j,t}\,t$
end for
return $\mathbf{P} = \{\mathbf{C}_k\}_{k=0}^{K-1}$, soft boundaries $\{\boldsymbol{A}_k\}_{k=0}^{K-1}$

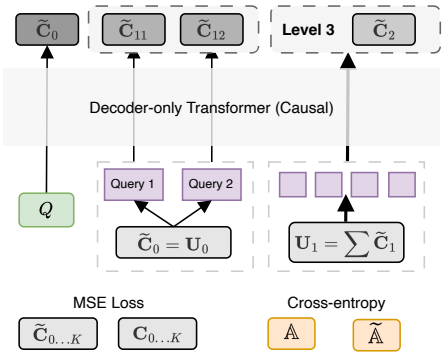


Figure 2: Illustration of *LCP-p*: per-level queries $\boldsymbol{\Phi}_k$ attend to the question and all coarser levels $<k$ under a block-wise causal mask to predict $\hat{\mathbf{C}}_k$ in parallel.

Residual decomposition and rank bottleneck. Collecting the attention weights into $\boldsymbol{A}_k = [\alpha_{k,j,t}] \in \mathbb{R}^{J_k \times T}$, the level-$k$ reconstruction $\boldsymbol{R}_k = \boldsymbol{A}_k^{\top}\mathbf{C}_k \in \mathbb{R}^{T \times d_c}$ updates the residual as $\mathbf{U}^{(k+1)} = \mathbf{U}^{(k)} - \boldsymbol{R}_k$, which preserves the invariant $\sum_{i \le k}\boldsymbol{R}_i + \mathbf{U}^{(k+1)} = \mathbf{U}^{(0)}$. Since $\boldsymbol{R}_k = \boldsymbol{A}_k^{\top}\mathbf{C}_k$ with $\boldsymbol{A}_k^{\top} \in \mathbb{R}^{T \times J_k}$ and $\mathbf{C}_k \in \mathbb{R}^{J_k \times d_c}$, the rank budget of level k is $\mathrm{rank}(\boldsymbol{R}_k) \le \min(J_k, T, d_c)$. Level 0 can therefore recover at most a rank-1 global direction of $\mathbf{U}^{(0)}$, whereas each finer level spans progressively more independent directions as $J_k = 2^k$ grows, until the budget saturates at $\min(T, d_c)$. This progressively increasing rank budget constitutes a *rank bottleneck* that imposes a coarse-to-fine decomposition by construction, analogous to how dimensional nesting induces hierarchical representations in Kusupati et al. [2022], and prevents any single level from monopolizing the representational capacity.

Intra-level ordering constraint. To preserve the sequential structure of reasoning, concepts within each level are constrained to attend to progressively later CoT positions. Let $\bar{t}_{k,j} = \sum_{t=1}^{T}\alpha_{k,j,t}\cdot t$ denote the expected position of $\boldsymbol{c}_{k,j}$; the margin-based ordering loss is $\mathcal{L}_{\text{order}} = \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}}\sum_{j=0}^{J_k-2}\mathrm{ReLU}\big(\bar{t}_{k,j} - \bar{t}_{k,j+1} + \gamma\big)$, where $\mathcal{K} = \{k : J_k > 1\}$ indexes levels containing more than one concept and $\gamma > 0$ is a margin. This softly enforces $\bar{t}_{k,j} < \bar{t}_{k,j+1}$, yielding positional monotonicity without imposing hard segment boundaries.

Combining these, the training objective is to produce pyramids that are reconstructible, ordered, residual-free, and semantically sufficient for reasoning

$$\mathcal{L}_{\text{b}} = \lambda_{\text{r}}\underbrace{\big\|\boldsymbol{W}_b\big(\textstyle\sum_k \boldsymbol{R}_k\big) - \mathbf{e_S}\big\|_2^2}_{\mathcal{L}_{\text{recon}}} + \lambda_{\text{o}}\,\mathcal{L}_{\text{order}} + \lambda_{\alpha}\underbrace{\big\|\mathbf{U}^{(K)}\big\|_1}_{\mathcal{L}_{\text{res}}} - \lambda_{\beta}\underbrace{\log p(\mathbb{A} \mid \mathbb{Q}, \mathbf{P})}_{\mathcal{L}_{\text{CE}}^{(\text{b})}},$$

where $\mathcal{L}_{\text{res}}$ penalizes any uncaptured residual in concept space, and $\mathcal{L}_{\text{CE}}^{(\text{b})}$ ensures that $\mathbf{P}$ retains sufficient reasoning information [Xu et al., 2025].

4.2 Learning Concept Pyramids Prediction

LCP-p uses a causal decoder-only LLM f_{θ_p} as its backbone and predicts the pyramid *level by level in parallel* (Figure 2): rather than emitting one concept per step, all J_k concepts of level k are produced in a single readout. The core idea is that, given the contextual features accumulated so far (the question and all coarser levels $<k$), *LCP-p* predicts which concepts at level k are needed to better approximate the target CoT. To this end, we extend the input with per-level *level queries* $\boldsymbol{\Phi}_k = [\boldsymbol{\phi}_{k,0}, \ldots, \boldsymbol{\phi}_{k,J_k-1}] \in \mathbb{R}^{J_k \times d_e}$, learnable read-out slots.

Block-wise causal mask. To enable parallel intra-level prediction while preserving inter-level dependence, we adopt a *block-wise causal mask* inspired by Tian et al. [2024]: slots within the same level attend to one another, while a slot at level k attends only to $\mathbb{Q}$ and all levels $<k$. Thus inter-level causality (k depends on $<k$) is preserved, whereas intra-level concepts share information freely and

are read out simultaneously through $\mathbf{\Phi}_k$. At inference, the pyramid is therefore generated in only K passes with KV caching, replacing the $\sum_k J_k$ token-by-token steps of flat autoregression.

Training objective. Under teacher forcing, the builder's ground-truth concepts $\mathbf{C}$ are lifted through $\boldsymbol{W}_b$ and tagged with slot markers $\boldsymbol{m}_{k,j}$, serving as the causal context $\mathbf{X}_c = \boldsymbol{W}_b\mathbf{C} + \mathbf{M} \in \mathbb{R}^{(\sum_k J_k)\times d_e}$, so that one forward pass of f_{θ_p} on $[\mathbf{q};\, \mathbf{X}_c;\, \mathbf{\Phi}]$ under the block-wise causal mask yields $\hat{\mathbf{C}}_k$. We then match $\hat{\mathbf{C}}_k$ to the frozen builder's pyramid via: $\mathcal{L}_{\text{p}} = \frac{1}{K}\sum_{k=0}^{K-1}\left\|\hat{\mathbf{C}}_k - \text{sg}[\mathbf{C}_k]\right\|_2^2 + \lambda_{\text{CE}}\mathcal{L}_{\text{CE}}^{(\text{p})}$, where $\text{sg}[\cdot]$ detaches gradients from the builder [Liu et al., 2026].

5 Experiments

5.1 Experimental Setup

Dataset. We train *L*CP on mathematical reasoning benchmarks `GSM8K` [Cobbe et al., 2021] (7,470) and `MATH` [Hendrycks et al., 2021] (7,500), and evaluate on their test splits. The `MATH`-trained checkpoint is evaluated on the held-out test split `MATH-500` [Lightman et al., 2023] and on the separate competition set `AIME2024`, and the `GSM8K`-trained checkpoint at its default K=4 is evaluated zero-shot on `MultiArith`. For code generation, `codeforces-cots` ('solutions_py_decontaminated' subset with 9,556 training samples, with the content between '¡think¿' of 'generation' serving as the CoT) is used for training and `HumanEval` for testing. Full dataset details are provided in Appendix C.1.

Implementation Details. We adopt the Qwen2.5 family ($\{$0.5B, 1.5B, 3B$\}$) and the Qwen3 family [Yang et al., 2025] ($\{$0.6B, 4B$\}$) as the backbone for both *LCP-b* and *LCP-p*, with pyramid depth K set per benchmark as listed in the caption of Table 1 and varied in the ablations of Section 5.4. The concept embedding dimension matches the backbone hidden size. For *LCP-p*, the backbone is frozen and adapted via LoRA on $\{W_q, W_v\}$ with rank $r = 16$, $\alpha = 32$, dropout 0.05; the back-projection layer, level embeddings, position embeddings, and concept head are trained from scratch. All samples are tokenized with the backbone tokenizer and truncated or right-padded to 512 tokens. At inference, only *LCP-p* is deployed, so the Builder introduces no inference overhead. For the interpretability, we train a small Qwen2.5-0.5B model to reproduce the CoT based on the concept pyramid.

Training. Training proceeds in two stages: Stage 1 trains *LCP-b*, and Stage 2 loads the best-on-eval Builder checkpoint to build the concept pyramid and trains *LCP-p*. Both stages share the same optimization recipe: AdamW with weight decay 0.01, linear warmup over the first 10% of steps followed by linear decay, gradient clipping at $\ell_2 = 1.0$, batch size 4, `float32` precision, and a fixed seed of 42. We train for 10 epochs on `GSM8K` and 15 epochs on `MATH` and `codeforces-cots`, with learning rate 1×10^{-4} for backbones $\leq$ 1.5B and 5×10^{-5} for backbones $\geq$ 3B. To prevent any single term from monopolizing the gradient, every loss weight is derived from measured warm-up magnitudes rather than tuned per cell (Section C.2): λ_{r} anchors $\mathcal{L}_{\text{recon}}$ at a weighted value of 10, λ_{o} anchors the weighted ordering term at 6, and $\lambda_\alpha = \lambda_\beta = 1$. The same three anchors apply unchanged across the full backbone-by-depth grid and both training corpora.

Metrics. We report *Accuracy*, *Token cost*, and *Inference time*, the last being the mean wall-clock latency per instance at batch size 1 in `FP16` on a single GPU, measured end-to-end from the question to the final answer: for *L*CP the answer stage (2.25 tokens on `GSM8K`, 4.37 on `MATH`) is included, while for code generation the latency covers the K concept passes (Appendix F.3). Because *L*CP generates one level of concepts per step, the token-cost column "T" reports *sequential generation steps* (pyramid depth K for *L*CP, generated tokens for CoT methods); Appendix F reports the measured FLOPs of a full pyramid inference.

Selecting the pyramid depth. K is bounded rather than searched: since $J_k = b^k$ and the finest level cannot exceed the width of the CoT it abstracts, with mean length T_{cot} in the training corpus, we require $b^{K-1} \leq T_{\text{cot}}$, i.e., $K \leq \lfloor \log_b T_{\text{cot}} \rfloor + 1$. Each corpus therefore has its own ceiling (Table 5: 7 for `GSM8K`, 8 for `MATH`, 10 for `codeforces-cots`), and the depths of Table 1 lie within them, with `MATH` using its full ceiling and `GSM8K` and `HumanEval` using smaller depths for latency. As latency grows roughly linearly with K over the depths we use, a latency budget supplies a complementary bound, and K is set to the largest depth that satisfies both.

Table 1: Accuracy (%, "A"), token cost (avg. generated tokens, "T"), and inference latency (seconds, "L") of baseline methods on six reasoning and code-generation benchmarks, measured on a single NVIDIA RTX 5090 (32 GB, FP16, batch size=1). Methods: Direct means answer generation, B-CoT=base+zero-shot CoT, I-CoT=instruct+zero-shot CoT, SFT=Supervised Finetuning with CoT. Levels of 4, 4, 8, 8, 8, and 8 are assigned to {`GSM8K`, `MultiArith`, `MATH`, `MATH-500`, `AIME2024`, `HumanEval`} following dataset complexity, and all lie within the depth ceiling $K \leq \lfloor \log_2 T_{\text{cot}} \rfloor + 1$ of their training corpus (Section 5.1). `GSM8K` and `MATH` are evaluated on their test splits, `MATH-500` is the held-out test split of `MATH`, `HumanEval` of `codeforces-cots`, and `AIME2024` and `MultiArith` are unseen datasets evaluated zero-shot on the `MATH`- and `GSM8K`-trained checkpoints, respectively. For `HumanEval`, the *L*CP latency covers the K concept passes (Appendix F.3).

		`GSM8K`			`MATH`			`MATH-500`			`AIME24`			`MultiArith`			`HumanEval`		
Model	**Method**	A	T	L	A	T	L	A	T	L	A	T	L	A	T	L	A	T	L
Qwen2.5-0.5B	Direct	4.5	12	0.17	2.5	14	0.20	3.0	17	0.24	0.0	19	0.27	6.7	8	0.11	3.7	47	0.67
	B-CoT	11.8	1637	20.00	25.8	1116	20.02	12.4	1251	22.20	0.0	1780	20.63	11.7	2264	30.80	9.8	2818	39.77
	I-CoT	41.6	412	5.89	31.2	783	11.19	31.6	826	11.80	0.0	1043	14.90	75.0	158	2.26	30.5	487	6.96
	SFT	51.8	247	3.53	34.5	512	7.31	36.0	538	7.69	0.0	663	9.47	82.2	97	1.39	32.3	316	4.51
	***L*CP**	36.1	**4**	**0.16**	27.0	**8**	**0.26**	22.0	**8**	**0.26**	0.0	**8**	**0.26**	70.0	**4**	**0.16**	23.2	**8**	**0.31**
Qwen2.5-1.5B	Direct	15.7	13	0.29	8.6	16	0.36	4.2	18	0.40	0.0	22	0.49	26.7	9	0.20	12.8	52	1.16
	B-CoT	28.3	1523	33.84	32.4	1087	24.16	16.8	1234	27.42	0.0	1812	40.27	36.7	2176	48.36	23.8	2634	58.53
	I-CoT	73.2	347	7.71	55.2	672	14.93	55.6	708	15.73	3.3	893	19.84	87.8	137	3.04	61.6	419	9.31
	SFT	75.8	213	4.73	56.4	428	9.51	56.8	451	10.02	3.3	572	12.71	90.6	84	1.87	63.4	273	6.07
	***L*CP**	55.7	**4**	**0.25**	42.0	**8**	**0.41**	40.0	**8**	**0.41**	0.0	**8**	**0.41**	83.3	**4**	**0.25**	40.2	**8**	**0.49**
Qwen2.5-3B	Direct	31.2	14	0.52	17.3	17	0.63	8.6	19	0.70	0.0	23	0.85	47.8	9	0.33	25.6	63	2.33
	B-CoT	43.7	1432	53.04	43.6	1056	39.11	24.6	1172	43.41	3.3	1753	64.93	57.2	2087	77.30	36.6	2542	94.15
	I-CoT	86.7	276	10.22	65.9	531	19.67	66.4	574	21.26	6.7	712	26.37	91.7	108	4.00	74.4	338	12.52
	SFT	88.4	168	6.22	67.3	346	12.81	67.8	367	13.59	6.7	463	17.15	93.3	69	2.56	76.2	222	8.22
	***L*CP**	74.0	**4**	**0.41**	50.0	**8**	**0.67**	47.4	**8**	**0.67**	0.0	**8**	**0.67**	85.6	**4**	**0.41**	63.4	**8**	**0.80**
Qwen3-0.6B	Direct	8.4	13	0.24	4.6	14	0.26	3.2	16	0.30	0.0	21	0.39	11.1	9	0.17	6.1	51	0.94
	B-CoT	15.8	1008	16.58	65.2	1840	34.42	45.8	2566	46.59	0.0	4090	73.56	16.7	433	9.19	47.0	2086	39.70
	I-CoT	52.6	643	11.91	68.3	1187	21.98	72.8	1676	31.04	13.3	2647	49.02	78.3	278	5.15	54.9	1347	24.94
	SFT	58.2	236	4.37	69.4	494	9.15	74.0	521	9.65	13.3	647	11.98	82.8	92	1.70	56.1	304	5.63
	***L*CP**	48.0	**4**	**0.21**	67.0	**8**	**0.34**	66.0	**8**	**0.34**	0.0	**8**	**0.34**	74.4	**4**	**0.21**	51.2	**8**	**0.40**
Qwen3-4B	Direct	38.4	14	0.70	20.8	17	0.85	14.2	19	0.95	3.3	24	1.20	51.1	9	0.45	28.0	66	3.30
	B-CoT	42.3	1184	59.20	72.4	2073	103.65	66.2	2847	142.35	13.3	4563	228.15	58.3	521	26.05	64.6	2358	117.90
	I-CoT	87.2	784	39.20	85.4	1432	71.60	90.2	2018	100.90	73.3	3186	159.30	93.3	336	16.80	70.1	1623	81.15
	SFT	88.6	153	7.65	86.7	314	15.70	91.4	342	17.10	76.7	428	21.40	95.0	64	3.20	72.6	208	10.40
	***L*CP**	72.2	**4**	**0.56**	80.0	**8**	**0.91**	77.0	**8**	**0.91**	13.3	**8**	**0.91**	90.6	**4**	**0.56**	69.5	**8**	**1.09**

5.2 Main Results

Table 1 reports accuracy, token cost, and inference latency across five backbones and six benchmarks. ***L*CP reduces inference latency by up to two orders of magnitude compared to the explicit-CoT methods on the mathematical benchmarks**, requiring only 4–8 autoregressive passes (one per pyramid level) versus hundreds of sequential token-generation steps for I-CoT and SFT. For instance, on `GSM8K` with Qwen3-4B, *L*CP completes inference in 0.56 s while I-CoT requires 39.20 s and SFT 7.65 s.

Despite this compression, *L*CP retains competitive reasoning accuracy under a latency budget. **On the Qwen3 family, the absolute accuracy gap to SFT ranges from about 2 to 16 percentage points on `GSM8K`, `MATH`, `MATH-500`, and `MultiArith`**; on `AIME2024`, where all methods are near the 30-problem resolution, the gap is larger (Qwen3-4B: 13.3% vs. SFT 76.7%), and on code generation (`HumanEval`) it is again modest (69.5% vs. 72.6% for Qwen3-4B).

The accuracy gap between *L*CP and SFT narrows consistently as backbone capacity increases: on `GSM8K` the relative gap shrinks from 30% (Qwen2.5-0.5B) to 19% (Qwen3-4B), and on `MATH` from 22% to 8%. **Larger backbones close the accuracy gap while the latency advantage remains intact**, indicating that the concept-space bottleneck becomes less restrictive as the LLM's representational capacity grows. This aligns with the rank-bottleneck design: a more expressive backbone encodes richer semantics into the same number of concept vectors, making the pyramid a tighter approximation of the full reasoning trace.

Because *L*CP leaves the backbone frozen, the natural reference point is zero-shot CoT on the same backbone (B-CoT) rather than SFT, which retrains the backbone on the in-domain CoT and then generates that CoT in full. A latent method must compress the same CoT into a handful of vectors, so a gap to full SFT is expected for any method in this family, and prior latent-reasoning work

Table 2: Comparison between *L*CP and state-of-the-art latent/implicit reasoning methods on GSM8K and MultiArith. All SOTA numbers are from He et al. [2025] Table 1, evaluated on *Llama-2-7B-chat* with NVIDIA H100 80 GB. Baselines: Pause [Goyal et al., 2024b], iCoT [Deng et al., 2024], Coconut [Hao et al., 2025], CODI [Shen et al., 2025], SoftCoT [Xu et al., 2025], SemCoT [He et al., 2025].

Method	GSM8K A	GSM8K T	GSM8K L	MultiArith A	MultiArith T	MultiArith L
State-of-the-art ($k=1$ implicit token)						
Pause	4.33	1	1.46	11.85	1	1.45
iCoT	4.50	1	1.48	8.70	1	1.48
Coconut	3.00	1	1.60	9.63	1	1.62
CODI	5.00	1	1.59	11.85	1	1.58
SoftCoT	8.83	5	1.05	7.96	1	1.03
SemCoT	9.83	1	1.02	15.93	1	0.94
SemCoT	6.52	3	1.19	10.00	3	1.06
SemCoT	2.50	5	1.24	7.22	5	0.92
Ours with Different Levels of Concept Pyramid						
***L*CP** $K=1$	9.24	1	1.05	15.00	1	0.91
***L*CP** $K=2$	10.83	2	1.15	18.89	2	1.06
***L*CP** $K=4$	12.50	4	1.30	22.22	4	1.22
***L*CP** $K=6$	**16.36**	6	1.48	**27.78**	6	1.38

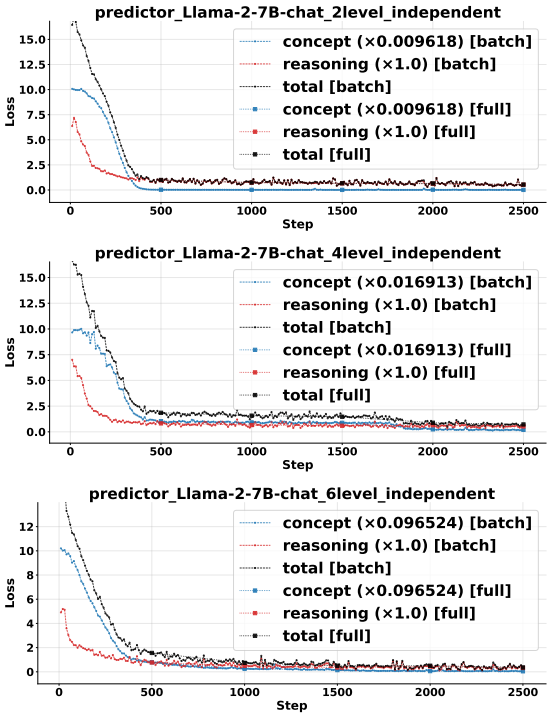


Figure 3: Evaluation loss curves for LCP predictor on GSM8K with $K = 2, 4, 6$ concept pyramid levels (top to bottom).

accordingly treats zero-shot CoT as its primary baseline [Xu et al., 2025, He et al., 2025]. On the largest backbone we evaluate, Qwen3-4B, *L*CP is above B-CoT on five of the six benchmarks (e.g., +29.9 points on GSM8K, +7.6 on MATH, +32.3 on MultiArith) and level with it on AIME2024, at a latency two orders of magnitude lower.

Beyond the held-out test splits above, *L*CP transfers to data it was not fine-tuned on. A Qwen3-4B model trained on GSM8K reaches 90.6% on MultiArith (vs. 58.3% for zero-shot CoT and 95.0% for SFT on the same backbone), so the concept pyramid transfers to a dataset whose questions and CoT style differ from training. On AIME2024, a competition set that no method was trained on, absolute accuracy remains low for every method (Qwen3-4B: 13.3% for both *L*CP and zero-shot CoT). HumanEval is a held-out in-domain test split of the codeforces-cots training corpus; the code-trained Qwen3-4B reaches 69.5% there (vs. SFT 72.6% and B-CoT 64.6% on the same backbone), showing that the identical pyramid factorization applies to code reasoning.

5.3 Comparison with State-of-the-Art Methods

Table 2 compares *L*CP against state-of-the-art latent/implicit reasoning methods under the same backbone (Llama-2-7B-chat) and hardware (H100). At $K = 1$, *L*CP reaches 9.24% on GSM8K and 15.00% on MultiArith, within one percentage point of the strongest single-token baseline, SemCoT (9.83%/15.93%). **As the pyramid depth K increases, *L*CP accuracy improves monotonically**, reaching 16.36% and 27.78% at $K = 6$, which represents a 77% and 85% relative gain over $K = 1$. The latency overhead remains modest: $K = 6$ requires only 1.48 s/1.38 s, comparable to single-token baselines such as Coconut (1.60 s/1.62 s) and CODI (1.59 s/1.58 s). This scaling property arises because each additional pyramid level introduces a structurally distinct granularity of reasoning, so the model receives genuinely new information at every level rather than redundant copies of the same representation.

In sharp contrast, **SemCoT exhibits consistent performance degradation as the number of latent steps increases**: from 9.83% to 6.52% to 2.50% on GSM8K, and from 15.93% to 10.00% to 7.22% on MultiArith, as k grows from 1 to 3 to 5. This reveals a fundamental limitation of flat sequential latent reasoning: when multiple latent tokens are generated at the same abstraction level

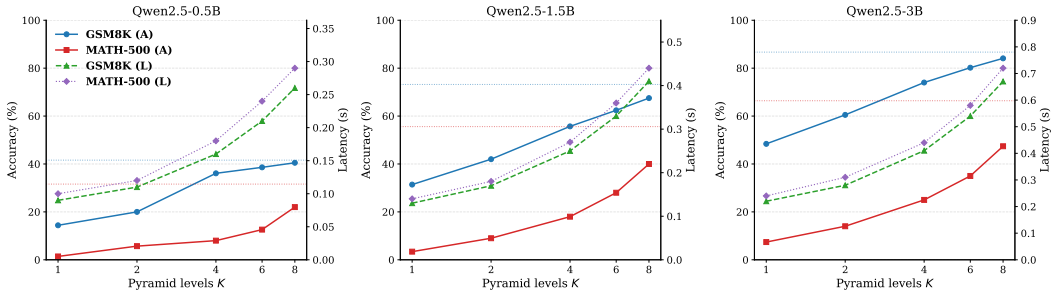


Figure 4: **Pyramid depth K trades latency for accuracy.** Accuracy (solid; `GSM8K` blue circles, `MATH-500` red squares) and per-instance latency (dashed/dotted; green triangles and purple diamonds) as K varies over $\{1, 2, 4, 6, 8\}$ at the default concept dimension (d matched to the backbone hidden size) on Qwen2.5-$\{$0.5B, 1.5B, 3B$\}$. Dotted horizontal lines mark I-CoT accuracy for reference. Hardware: RTX 5090 32 GB, FP16, batch size=1.

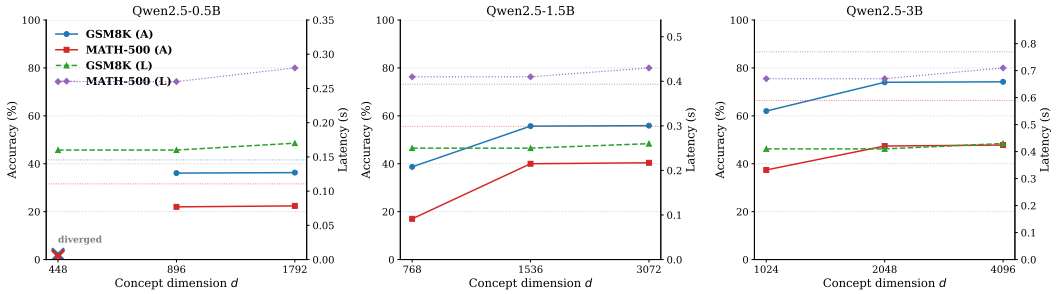


Figure 5: **Matching concept dimension d to backbone hidden size is the robust default.** Accuracy and latency as d varies over $\{\text{half}, \text{match}, \text{double}\}$ of the backbone hidden size (0.5B: 448/896/1792; 1.5B: 768/1536/3072; 3B: 1024/2048/4096) at default levels ($K = 4$ for `GSM8K`, $K = 8$ for `MATH-500`). Hardware: RTX 5090 32 GB, FP16, batch size=1.

without structural differentiation, later tokens lack a clear semantic role and degrade into noise that confuses the final prediction. The hierarchical coarse-to-fine structure of *L*CP resolves this problem by assigning each level an explicit granularity target, from the global reasoning outline at the apex ($k = 0$) to fine-grained local steps at the base ($k = K-1$), ensuring that additional capacity is always directed toward genuinely finer-resolution information.

5.4 Ablation Study

Figure 4 shows that accuracy grows monotonically with pyramid depth K, with the steepest gains at lower levels ($K = 1 \rightarrow 4$ on `GSM8K`; $K = 4 \rightarrow 8$ on `MATH-500`) and diminishing returns beyond $K = 8$. Accuracy saturates sublinearly after $K \approx 8$ because coarser levels already capture the dominant reasoning structure, whereas the number of sequential passes, and hence the latency, keeps growing with K; this is why the reported depths stop at K=8, at or near the corpus-specific depth ceiling of Section 5.1 (7, 8, and 10 for our three training corpora), instead of going deeper. Figure 5 reveals that **halving d below the backbone hidden size causes severe accuracy collapse** (up to 34 percentage-point drop), whereas doubling d yields negligible gain ($\leq 0.4\%$) with moderate latency overhead. This confirms the rank-bottleneck design: the backbone hidden size provides a natural capacity budget for concept vectors, and matching d to it is Pareto-optimal without per-task tuning.

The concept pyramid $\mathbf{P}$ is *decodable*: a lightweight Qwen2.5-0.5B decoder reconstructs a textual CoT from the predicted concepts. As Figure 6 illustrates on `GSM8K` dataset, **the most critical CoT spans (red), namely key quantities, governing equations, and the decisive inference step, are consistently recovered**, while less informative connectives may be paraphrased. This demonstrates that *L*CP effectively captures the essential reasoning structure of the problem, enabling recovery of the key steps of the CoT even when the predicted concepts are not perfectly aligned with the ground-truth.

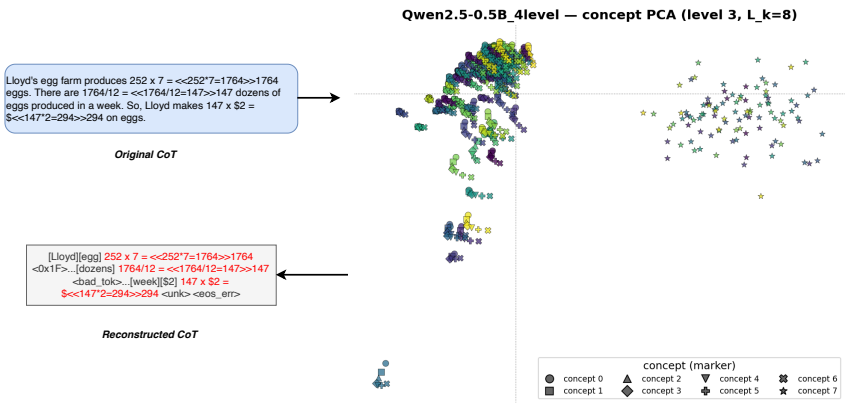


Figure 6: Textual CoT reconstructed by the Qwen2.5-0.5B decoder from the predicted concept pyramid $\mathbf{P}$ on `GSM8K`. The middle subfigure presents the 2D embeddings of the final-level concepts. We randomly select concept pyramids of 200 samples for visualization.

6 Conclusion

We presented *Latent Concept-Pyramid Modeling* (*L*CP), which reformulates LLM reasoning as hierarchical next-level concept generation in a coarse-to-fine manner. By decomposing the Chain-of-Thought into a pyramid of progressively finer-grained concepts via residual decomposition (*LCP-b*) and predicting the pyramid level by level with a causal decoder (*LCP-p*), *L*CP reduces inference latency by up to two orders of magnitude while retaining competitive accuracy under a latency budget across mathematical and coding reasoning benchmarks. The accuracy gap to supervised fine-tuning narrows consistently with larger backbones, and the learned pyramid transfers zero-shot to unseen datasets of the same task. Unlike flat latent reasoning methods whose performance degrades with additional latent steps, *L*CP accuracy improves monotonically as pyramid depth increases, confirming the effectiveness of structuring latent reasoning into explicit granularity levels. Furthermore, the concept pyramid can be decoded back into a textual CoT, making the reasoning trajectory readable.

Acknowledgments and Disclosure of Funding

This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No. 2024A03J0630) and was carried out at the Guangzhou Municipal Key Laboratory of Financial Technology Cutting-Edge Research. The authors declare no competing interests.

References

Hongjun An, Yifan Chen, Zhe Sun, and Xuelong Li. Sentencevae: Enable next-sentence prediction for large language models with faster speed, higher accuracy and longer context. *arXiv preprint arXiv:2408.00655*, 2024.

Loc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R. Costa-juss, David Dale, Hady Elsahar, Kevin Heffernan, Joo Maria Janeiro, Tuan Tran, Christophe Ropers, Eduardo Snchez, Robin San Roman, Alexandre Mourachko, Safiyyah Saleem, and Holger Schwenk. Large concept models: Language modeling in a sentence representation space. In *Proc. Conference on Neural Information Processing Systems*, 2025.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms. In *Proc. International Conference on Machine Learning*, 2025a.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2025b.

Yunlong Chu, Minglai Shao, Yuhang Liu, Bing Hao, Yumeng Lin, Jialu Wang, and Ruijie Wang. Spot: Span-level pause-of-thought for efficient and interpretable latent reasoning in large language models. *arXiv preprint arXiv:2603.06222*, 2026.

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.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Kawin Ethayarajh. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proc. Conference on Empirical Methods in Natural Language Processing*, 2019.

Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Representation degeneration problem in training natural language generation models. In *Proc. International Conference on Learning Representations*, 2019.

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. In *Proc. International Conference on Learning Representations*, 2024a.

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. In *Proc. International Conference on Learning Representations*, 2024b.

Halil Alperen Gozeten, M. Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. In *Proc. International Conference on Learning Representations*, 2026.

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. In *Proc. Conference on Language Modeling*, 2025.

Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. Semcot: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. In *Proc. Conference on Neural Information Processing Systems*, 2025.

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*, 2021.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *Proc. International Conference on Learning Representations*, 2026.

Cong Jiang, Xiaofeng Zhang, Fangzhi Zhu, XiaoWei Chen, Junxiong Zhu, and Zheng Zhang. Rethinking llm reasoning: From explicit trajectories to latent representations. In *Proc. International Conference on Learning Representations*, 2026.

Nan Jiang, Ziming Wu, De-Chuan Zhan, Fuming Lai, and Shaobing Lian. Dart: Distilling autoregressive reasoning to silent thought. In *Proc. Conference on Empirical Methods in Natural Language Processing*, 2025.

Etienne Koechlin and Christopher Summerfield. An information theoretical approach to prefrontal executive function. *Trends in cognitive sciences*, 11(6):229–235, 2007.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *Proc. Conference on Neural Information Processing Systems*, 2022.

Anna Kuzina, Maciej Pioro, Paul N. Whatmough, and Babak Ehteshami Bejnordi. Kava: Latent reasoning via compressed kv-cache distillation. In *Proc. International Conference on Learning Representations*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Luyang Liu, Jonas Pfeiffer, Jiaxing Wu, Jun Xie, and Arthur Szlam. Deliberation in latent space via differentiable cache augmentation. In *Proc. International Conference on Machine Learning*, 2025.

Yuliang Liu, Yunchong Song, Yixuan Wang, Kewen Ge, Alex Lamb, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Next concept prediction in discrete latent space leads to stronger language models. *arXiv preprint arXiv:2602.08984*, 2026.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *Proc. Conference on Language Modeling*, 2024.

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. In *Proc. ICLR Workshop on Latent and Implicit Thinking*, 2026.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *Proc. International Conference on Learning Representations*, 2025.

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 *Proc. Conference on Empirical Methods in Natural Language Processing*, 2025.

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 *Proc. International Conference on Machine Learning*, 2025.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. In *Proc. Advances in Neural Information Processing Systems*, 2022.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, et al. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of llm reasoning chains. In *Proc. Conference on Neural Information Processing Systems*, 2025.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *Proc. Conference on Neural Information Processing Systems*, 2024.

Wenshuo Wang. Llm reasoning is latent, not the chain of thought. *arXiv preprint arXiv:2604.15726*, 2026.

Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. System-1.5 reasoning: Traversal in language and latent spaces with dynamic shortcuts. In *Proc. Conference on Neural Information Processing Systems*, 2025.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration. In *Proc. Conference on Empirical Methods in Natural Language Processing*, 2025.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. In *Proc. International Conference on Learning Representations*, 2026.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2025.

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.

Xinlei Yu, Zhangquan Chen, Yongbo He, Tianyu Fu, Cheng Yang, Chengming Xu, Yue Ma, Xiaobin Hu, Zhe Cao, Jie Xu, Guibin Zhang, Jiale Tao, Jiayi Zhang, Siyuan Ma, Kaituo Feng, Haojie Huang, Youxing Li, Ronghao Chen, Huacan Wang, Chenglin Wu, Zikun Su, Xiaogang Xu, Kelu Yao, Kun Wang, Chen Gao, Yue Liao, Ruqi Huang, Tao Jin, Cheng Tan, Jiangning Zhang, Wenqi Ren, Yanwei Fu, Yong Liu, Yu Wang, Xiangyu Yue, Yu-Gang Jiang, and Shuicheng Yan. The latent space: Foundation, evolution, mechanism, ability, and outlook. *arXiv preprint arXiv:2604.02029*, 2026.

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 *Proc. Conference on Neural Information Processing Systems*, 2025.

Boyi Zeng, Yiqin Hao, He Li, Shixiang Song, Feichen Song, Zitong Wang, Siyuan Huang, Yi Xu, ZiWei He, Xinbing Wang, and Zhouhan Lin. Pretraining with token-level adaptive latent chain-of-thought. *arXiv preprint arXiv:2602.08220*, 2026.

Fangwei Zhu and Zhifang Sui. Colt: Reasoning with chain of latent tool calls. *arXiv preprint arXiv:2602.04246*, 2026.

A Limitations

While *L*CP achieves competitive accuracy together with substantial reductions in token cost and inference latency, several aspects of its current realization warrant a more cautious reading of the results.

A first limitation concerns the way the pyramid is configured during training. Both the depth K and the per-level width $J_k = 2^k$ are treated as fixed hyper-parameters in our experiments, so the same schedule is applied to every input regardless of problem complexity, even though Table 2 indicates that the optimal K varies across benchmarks. This rigidity is further compounded by the **two-stage optimization with a frozen builder**, in which *LCP-p* is trained against the stop-gradient $\mathrm{sg}[\mathbf{C}_k]$: any sub-optimality in the concept pyramid produced by *LCP-b* therefore propagates downstream, and the compound objective $\mathcal{L}_{\mathrm{b}} = \lambda_{\mathrm{r}}\mathcal{L}_{\text{recon}} + \lambda_{\mathrm{o}}\mathcal{L}_{\text{order}} + \lambda_{\alpha}\mathcal{L}_{\text{res}} - \lambda_{\beta}\mathcal{L}_{\mathrm{CE}}^{(\mathrm{b})}$ is calibrated from warm-up measurements at each $(\text{backbone}, K)$ cell rather than tuned per input (Appendix C.2).

A second limitation lies in the modeling assumption itself. The block-wise causal mask treats the J_k concepts within a level as conditionally exchangeable given $\mathbb{Q}$ and $\mathbf{C}_{<k}$, with intra-level ordering enforced only *softly* during builder training through $\mathcal{L}_{\text{order}}$ on the expected positions $\bar{t}_{k,j}$. This design is precisely what enables the K-pass speedup, yet it is also **the strongest modeling assumption made by *LCP-p***: for reasoning chains in which the intra-level steps carry strong sequential dependence, the parallel readout may dilute the positional cue that the builder aimed to preserve, which in turn limits how aggressively J_k can be widened without degrading reasoning accuracy.

A third limitation concerns the breadth of our experiments and the quality of the recovered reasoning trajectory. Our evaluation covers the Qwen2.5 family ($\{0.5, 1.5, 3\}$B) and the Qwen3 family ($\{0.6, 4\}$B) on mathematical reasoning (`GSM8K`, `MATH`, `MATH-500`, `AIME2024`, `MultiArith`) and code generation (`HumanEval`, `codeforces-cots`); although these two regimes already exhibit different CoT styles, they both share a coarse-to-fine reasoning structure that lends itself to pyramid factorization, and **whether the same factorization remains effective for other LLM families and for open-ended domains** such as long-horizon agentic planning or free-form dialogue is therefore an empirical question we do not settle here. A related issue applies to interpretability: although the concept pyramid $\mathbf{P}$ is in principle decodable, the textual readout used in our experiments is produced by an auxiliary Qwen2.5-0.5B decoder, whose capacity upper-bounds the quality of the recovered reasoning trajectory even when $\mathbf{P}$ itself is informative.

B Broader Implications

Beyond the efficiency-accuracy trade-off reported in our experiments, the design choices behind *L*CP point to several broader research directions that go beyond a single benchmarking effort.

The most direct consequence is methodological. The combination of per-level queries $\mathbf{\Phi}_k$ and the block-wise causal mask reduces autoregressive generation from $\sum_k J_k$ steps to K passes whenever the target factorizes into a coarse-to-fine hierarchy, which makes ***multi-level concept generation* a candidate primitive for structured generation more broadly**, including code synthesis with hierarchical templates, multi-step tool use, and plan decomposition where intra-level outputs are conditionally exchangeable given coarser context. The corresponding reduction in token cost and inference latency further translates into lower compute and energy consumption at deployment scale, which becomes increasingly relevant as LLM-based reasoning systems are pushed into latency-sensitive and resource-constrained settings.

A second implication concerns reinforcement learning with process-level supervision. Because each predicted level $\hat{\mathbf{C}}_k$ is an intermediate, decodable representation of the reasoning trajectory rather than an opaque hidden state, **the pyramid offers a natural interface for process-level supervision**: each $\hat{\mathbf{C}}_k$ can serve as a checkpoint for process-level reward models, enabling level-wise credit assignment for reinforcement learning without resorting to token-by-token rollouts of the full CoT. The same property also supports concept-level intervention before answer generation, since editing or constraining individual concepts $\boldsymbol{c}_{k,j}$ operates at the concept level rather than the token level, which is useful for safety-oriented deployment, counterfactual analysis, and human-in-the-loop reasoning.

A third implication is for the broader latent-reasoning literature. The *soft boundaries* formulation, together with the rank-bottlenecked residual decomposition that organizes $\mathbf{P}$ into a coarse-to-fine hierarchy, is in principle compatible with prior latent-CoT compression pipelines that currently rely on hard segmentation [Qu et al., 2026, Hao et al., 2025, He et al., 2025], and **can be viewed as a drop-in replacement for hard segmentation in those pipelines**. This may mitigate the degradation observed as the number of latent steps grows (Table 2) and offers a more principled mechanism for organizing flat latent representations into multi-level hierarchies, which we view as a natural follow-up direction.

A fourth implication is for long-context compression beyond reasoning. Existing compression approaches typically encode long inputs into a flat sequence of latent tokens [Jiang et al., 2025, 2026, Liu et al., 2025, Xu et al., 2025], whose uniform granularity leaves downstream modules unable to choose which details are preserved and which are discarded. The concept pyramid $\mathbf{P}$ points to a **hierarchical alternative in which long inputs are compressed into coarse-to-fine levels** rather than into a single-granularity latent sequence, so that coarser levels retain the global outline while finer levels retain local details, and downstream modules can selectively access the granularity they need. Such a hierarchical compact representation may prove useful for long-document summarization, retrieval-augmented generation over extended contexts, and memory compression in long-horizon agents, where the uniform granularity of flat latent tokens is a known bottleneck.

C Experimental Setup Details

C.1 Dataset Summary

Table 3: Dataset sources from HuggingFace. "Q Tok." = average question tokens, "CoT Tok." = average CoT tokens.
Token lengths are computed using the `Qwen3-0.6B` tokenizer over 200 random samples (or the full set where smaller). `MultiArith` does not contain step-by-step reasoning in its raw data, so its CoT token count is marked `N/A`.

Dataset	Source	Tr.	Tst.	Split	Q Tok.	CoT Tok.
GSM8K	openai/gsm8k	7,470	1,320	train/test	61	120
MATH	DigitalLearningGmbH/MATH-lighteval	7,500	5,000	train/test	52	148
MATH500	HuggingFaceH4/MATH-500	–	500	test	70	218
AIME2024	HuggingFaceH4/aime_2024	–	30	test	107	1,635
MultiArith	ChilleD/MultiArith	420	180	train/test	41	N/A
HumanEval	openai/openai_humaneval	–	164	test	147	54
CodeForces-CoTS	open-r1/codeforces-cots	9,556 (py)	–	train	367	943

C.2 Concept Predictor (*LCP-p*) Detailed Procedure

As described in Section 4.2, *LCP-p* employs a causal decoder-only LLM f_{θ_p} as its backbone and predicts the concept pyramid *level by level in parallel*. The core idea is that, given the contextual features accumulated so far (the question $\mathbb{Q}$ and all coarser levels $<k$), *LCP-p* predicts which concepts at level k are needed to *better approximate the target CoT*. This mirrors the builder's residual decomposition: the builder's $\mathbf{U}^{(k+1)} = \mathbf{U}^{(k)} - \boldsymbol{R}_k$ represents what coarser levels have not yet captured, and the predictor correspondingly learns to generate concepts that further refine the CoT approximation at each successively finer granularity.

To achieve parallel intra-level prediction, per-level *level queries* $\boldsymbol{\Phi}_k = [\boldsymbol{\phi}_{k,0}, \ldots, \boldsymbol{\phi}_{k,J_k-1}] \in \mathbb{R}^{J_k \times d_e}$ are appended to the input sequence. Under a *block-wise causal mask* [Tian et al., 2024], each $\boldsymbol{\phi}_{k,j}$ attends to (i) $\mathbb{Q}$, (ii) all concept slots from levels $<k$, and (iii) sibling queries within the same level k, but *cannot* see concept slots at level $\geq k$. After the forward pass, the output hidden states at the $\boldsymbol{\Phi}_k$ positions are decoded into $\hat{\mathbf{C}}_k$ via a small head g_c. Algorithms 2 and 3 detail the training and inference procedures, respectively.

Training (Algorithm 2). Under teacher forcing, ground-truth concepts are lifted to encoder space via the back-projection $\boldsymbol{W}_b$ and tagged with slot markers $\boldsymbol{m}_{k,j}$ (level and position embeddings encoding each slot's identity), forming the concept-slot matrix $\mathbf{X}_c = \boldsymbol{W}_b\mathbf{C} + \mathbf{M} \in \mathbb{R}^{(\sum_k J_k)\times d_e}$. The full input $[\mathbf{q}; \mathbf{X}_c; \boldsymbol{\Phi}]$ is processed by f_{θ_p} in a single forward pass. The block-wise causal

Algorithm 2 *LCP-p* Training (Teacher Forcing)

Require: $\mathbb{Q}$, GT pyramid $\{\mathbf{C}_k\}_{k=0}^{K-1}$ from frozen builder, backbone f_{θ_p}, back-projection $\boldsymbol{W}_b$, level queries $\{\boldsymbol{\Phi}_k\}_{k=0}^{K-1}$, head g_c

// lift GT concepts to encoder space
$\mathbf{X}_c \leftarrow \boldsymbol{W}_b\,\mathrm{cat}(\mathbf{C}_0, \ldots, \mathbf{C}_{K-1}) + \mathbf{M}$
// $\boldsymbol{m}_{k,j}$: level & position embedding for slot (k, j); $\mathbf{M} = [\boldsymbol{m}_{k,j}]$
// concatenate level queries as prediction slots
$\boldsymbol{\Phi} \leftarrow \mathrm{cat}(\boldsymbol{\Phi}_0, \ldots, \boldsymbol{\Phi}_{K-1})$
// single forward pass under block-wise causal mask
$f_{\theta_p}\big([\mathbf{q};\, \mathbf{X}_c;\, \boldsymbol{\Phi}]\big)$
// mask: $\boldsymbol{\Phi}_k$ sees $\mathbb{Q}$, $\mathbf{X}_{c,<k}$, $\boldsymbol{\Phi}_{\le k}$
for $k = 0$ **to** $K-1$ **do**
  // decode $\boldsymbol{\Phi}_k$'s output to concept space
  $\hat{\mathbf{C}}_k \leftarrow g_c\big(f_{\theta_p}.\mathrm{output}[\boldsymbol{\Phi}_k]\big) \in \mathbb{R}^{J_k \times d_c}$
end for
$\mathcal{L}_{\mathrm{p}} \leftarrow \frac{1}{K}\sum_{k=0}^{K-1} \big\|\hat{\mathbf{C}}_k - \mathrm{sg}[\mathbf{C}_k]\big\|_2^2 + \lambda_{\mathrm{CE}}\mathcal{L}_{\mathrm{CE}}^{(\mathrm{p})}$
return $\mathcal{L}_{\mathrm{p}}$

Algorithm 3 *LCP-p* Inference (K Passes with KV Cache)

Require: question $\mathbb{Q}$ with embedding $\mathbf{q}$, predictor f_{θ_p} whose trainable parameters include level queries $\{\boldsymbol{\Phi}_k\}_{k=0}^{K-1}$, back-projection $\boldsymbol{W}_b$, slot markers $\{\boldsymbol{m}_{k,j}\}$, concept head g_c

// accumulated predictor context (grows via the KV cache); starts with the question only
$\mathbf{U}_0 \leftarrow \mathbf{q}$
for $k = 0$ **to** $K-1$ **do**
  // trainable level-k queries read $\mathbf{U}_k$ under the block-wise causal mask;
  // $\boldsymbol{\Phi}_k$ are not *user inputs but internal slots of f_{θ_p}*
  $\mathbf{h}_k \leftarrow f_{\theta_p}\big(\mathbf{U}_k;\, \boldsymbol{\Phi}_k\big)\big[\boldsymbol{\Phi}_k\big] \in \mathbb{R}^{J_k \times d_e}$
  // decode level-k slot outputs to concept space
  $\hat{\mathbf{C}}_k \leftarrow g_c(\mathbf{h}_k) \in \mathbb{R}^{J_k \times d_c}$
  // lift predicted concepts to encoder space with slot markers $\mathbf{M}_k = [\boldsymbol{m}_{k,0}, \ldots, \boldsymbol{m}_{k,J_k-1}]$
  $\mathbf{X}_k \leftarrow \boldsymbol{W}_b\hat{\mathbf{C}}_k + \mathbf{M}_k$
  // accumulate into context (appended to KV cache) for the next pass
  $\mathbf{U}_{k+1} \leftarrow [\mathbf{U}_k;\, \mathbf{X}_k]$
end for
// answer generation from the question and the predicted pyramid
$\mathbb{A} \sim p_{\theta_p}\big(\cdot \mid \mathbb{Q},\, \hat{\mathbf{P}}\big)$
return pyramid $\hat{\mathbf{P}} = \{\hat{\mathbf{C}}_k\}_{k=0}^{K-1}$ and answer $\mathbb{A}$

mask ensures that each $\boldsymbol{\phi}_{k,j}$ sees only the question and coarser levels' concept slots (providing the "accumulated context so far"), plus sibling queries at level k, but never concept slots at level $\ge k$. Conceptually, $\boldsymbol{\Phi}_k$ learns to predict *which concepts at this granularity are still needed* given what coarser levels have already captured. The head g_c maps each query's output state to concept space, producing $\hat{\mathbf{C}}_k \approx \mathbf{C}_k$.

Inference (Algorithm 3). Without ground-truth concepts, the predictor generates the pyramid autoregressively over K passes with KV caching. We maintain an accumulated context $\mathbf{U}_k$ that starts with the question embedding, $\mathbf{U}_0 = \mathbf{q}$. At pass k, the level-k queries $\boldsymbol{\Phi}_k$ read $\mathbf{U}_k$ under the block-wise causal mask and produce hidden states $\mathbf{h}_k = f_{\theta_p}(\mathbf{U}_k;\, \boldsymbol{\Phi}_k)[\boldsymbol{\Phi}_k]$, which are decoded into $\hat{\mathbf{C}}_k = g_c(\mathbf{h}_k)$. The predicted concepts are then lifted back to encoder space with slot markers, $\mathbf{X}_k = \boldsymbol{W}_b\hat{\mathbf{C}}_k + \mathbf{M}_k$, and appended to the context, $\mathbf{U}_{k+1} = [\mathbf{U}_k;\, \mathbf{X}_k]$, which is realized in practice by extending the KV cache. Because the coarser-level keys and values are cached across passes, each pass only processes the newly added positions, reducing the total sequential calls from $\sum_k J_k$ (flat autoregression) to only K passes. After the K passes, the answer $\mathbb{A}$ is generated conditionally on $\mathbb{Q}$ and the predicted pyramid $\hat{\mathbf{P}}$, so the latency reported in the experiments covers both concept prediction and answer generation.

Hardware Information. All training runs are conducted on a server equipped with $4\times$ NVIDIA RTX 5090 (32 GB) GPUs. Inference for the main results (Table 1) and ablations (Figures 4 and 5) is performed on a single RTX 5090 (32 GB, FP16, batch size$=1$); for the state-of-the-art comparison in Table 2, inference follows the hardware setup of He et al. [2025] on an NVIDIA H100 (80 GB) GPU. All experiments run under CUDA 12.8.

Loss Weight Calibration. Both training stages mix loss components whose natural magnitudes vary across (backbone, K) cells by orders of magnitude, so a fixed weighting schedule would let a single

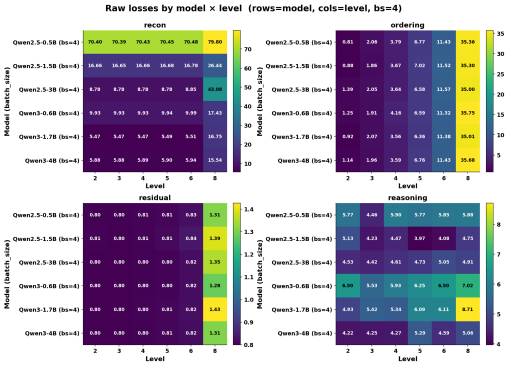
term dominate the gradient. The Builder combines four components (reconstruction $\mathcal{L}_{\text{recon}}$, ordering $\mathcal{L}_{\text{order}}$, residual $\mathcal{L}_{\text{res}}$, and reasoning cross-entropy $\mathcal{L}_{\text{CE}}^{(\text{b})}$) whose raw scales span distinct regimes: $\mathcal{L}_{\text{recon}}$ lands near 77 for Qwen2.5-0.5B but closer to 9 for Qwen3-0.6B, while $\mathcal{L}_{\text{order}}$ grows combinatorially with K, e.g., from ≈ 0.6 at $K = 2$ to ≈ 35 at $K = 8$ on GSM8K; by contrast, $\mathcal{L}_{\text{res}}$ and $\mathcal{L}_{\text{CE}}^{(\text{b})}$ already sit in narrow bands of $[0.8, 1.3]$ and $[4, 8]$, respectively. The Predictor then couples a per-level concept-matching MSE $\mathcal{L}_{\text{concept}}$ with its own reasoning CE $\mathcal{L}_{\text{CE}}^{(\text{p})}$, and the former inherits an even wider spread, with raw values ranging from ≈ 1040 for (Qwen2.5-0.5B, $K = 2$) down to ≈ 20 for (Qwen3-0.6B, $K = 8$). We therefore derive every weight from measured warm-up magnitudes with a single rule rather than setting it by hand. First, one warm-up run per (backbone, K) cell and stage (a few forward passes at the training batch size with no optimizer step) logs every component's raw (pre-weight) magnitude; the resulting per-component heatmaps are shown in Figure 7 for the Builder landscape on GSM8K and MATH and the Predictor landscape on GSM8K. Second, the rule anchors the two structural terms and expresses everything else relative to the reasoning cross-entropy at unit weight: $\mathcal{L}_{\text{recon}}$ is anchored at a weighted value of 10 and $\mathcal{L}_{\text{order}}$ at 6, a term whose raw magnitude already lies below its anchor is not rescaled, and the remaining terms, whose raw scales vary little, are held at a one-to-one or one-to-two ratio against the reasoning cross-entropy. Concretely, the Builder uses a capped anchoring rule $\lambda_{\text{r}} = \min(1,\, 10/\bar{\ell}_{\text{recon}})$ (e.g., ≈ 0.13 for Qwen2.5-0.5B, while for Qwen3-0.6B the raw value is already ≈ 9.4 so the rule returns the cap 1.00), keeps $\lambda_\alpha = \lambda_\beta = 1$ since the residual and Builder CE terms already lie in the target band, and lets λ_{o} decrease monotonically with k as $\min(1,\, 6/\bar{\ell}_{\text{order},k})$ so that the weighted $\lambda_{\text{o}}\mathcal{L}_{\text{order}}$ is held near 6 across all K; the Predictor follows the same rule with $w_{\text{concept}} = 10/\bar{\ell}_{\text{concept}}$ per cell and $w_{\text{CE}} = 1$. Because the rule's only input is a measurement, it is a form of adaptive loss balancing, specifically *static scale normalization*, and it is task-agnostic: the anchors 10 (reconstruction) and 6 (ordering) are the only quantities chosen by the authors, and they apply unchanged across the entire backbone-by-depth grid on both GSM8K and MATH, because the reasoning cross-entropy that fixes their scale comes from a frozen backbone whose next-token-prediction loss varies little across tasks. Table 4 showcases this rule on the Predictor stage of GSM8K as a concrete example: raw concept losses ranging from ≈ 20 to ≈ 1040 all map to a weighted concept term of exactly 10, and the resulting weighted totals stay within the narrow band $[14, 18]$ regardless of backbone or K. Appendix F compares the rule against three learned weighting schemes (uncertainty weighting, GradNorm, and dynamic weight average) under the same setup.

Table 4: Calibrated Predictor-stage loss weights on GSM8K (independent mode). $\bar{\ell}_\star$ columns report raw pre-weight magnitudes from the warm-up pass; $w_\star$ columns are the resulting weights set so that the weighted concept term is anchored to 10 and the reasoning term is kept at unity.

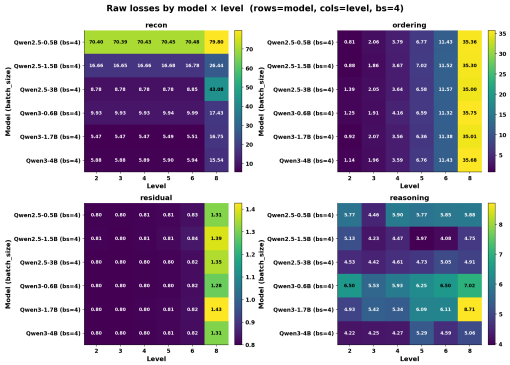
Backbone	K	$\bar{\ell}_{\text{concept}}$	$\bar{\ell}_{\text{CE}}$	w_{concept}	w_{CE}	$\text{wt}_{\text{concept}}$	wt_{total}
Qwen2.5-0.5B	2	1039.74	5.99	0.0096	1.00	10.00	15.99
Qwen2.5-0.5B	4	591.26	6.92	0.0169	1.00	10.00	16.92
Qwen2.5-0.5B	6	103.60	5.40	0.0965	1.00	10.00	15.40
Qwen2.5-0.5B	8	51.16	5.69	0.1955	1.00	10.00	15.69
Qwen3-0.6B	2	374.19	7.70	0.0267	1.00	10.00	17.70
Qwen3-0.6B	4	425.82	7.21	0.0235	1.00	10.00	17.21
Qwen3-0.6B	6	67.64	6.41	0.1479	1.00	10.00	16.41
Qwen3-0.6B	8	20.50	7.43	0.4877	1.00	10.00	17.43
Qwen2.5-1.5B	2	746.00	4.41	0.0134	1.00	10.00	14.41
Qwen2.5-1.5B	4	944.14	4.13	0.0106	1.00	10.00	14.13
Qwen2.5-1.5B	6	132.65	4.81	0.0754	1.00	10.00	14.81
Qwen2.5-1.5B	8	22.18	5.79	0.4508	1.00	10.00	15.79

D Learning Curve Analysis

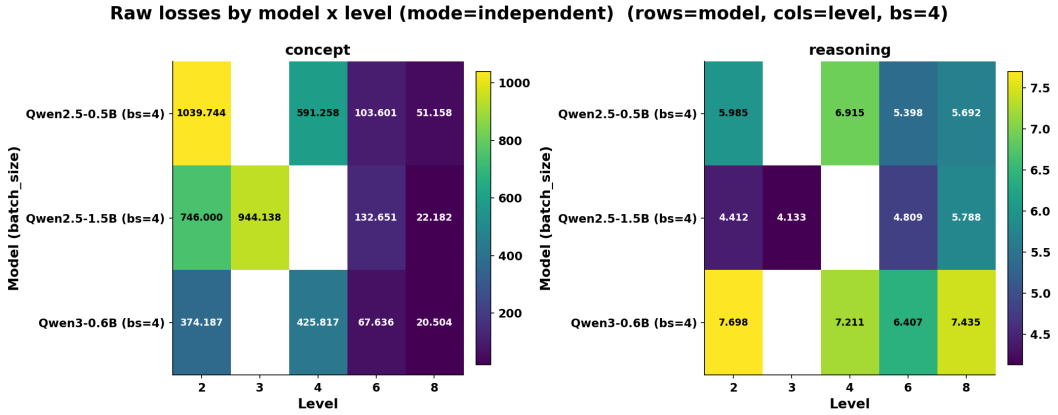
This section presents the training and evaluation loss curves for both *LCP-b* and *LCP-p*, organized by dataset, backbone, and pyramid depth K. To avoid the gradient-domination effect documented in the loss-weight study (**Loss Weight Calibration** paragraph, Section C.2), all curves shown here use the AutoWeighted (AW) loss-balancing schedule that anchors each weighted component to the natural scale of the reasoning cross-entropy. Predictor evaluation curves for `GSM8K`–Qwen2.5-0.5B at $K \in \{2, 4, 6\}$ already appear in Figure 3 of the main paper and are therefore omitted here. Within every



(a) GSM8K Builder.



(b) MATH Builder.



(c) GSM8K Predictor.

Figure 7: Raw (pre-weight) loss magnitudes across the $(\text{backbone}, K)$ grid, used to calibrate $\{\lambda_{\text{r}}, \lambda_{\text{o}}, \lambda_{\alpha}, \lambda_{\beta}\}$ at each stage.

subfigure, the plotted curves are the per-component losses tracked during training (reasoning cross-entropy, concept reconstruction and concept cross-entropy for *LCP-b*; concept MSE and concept cross-entropy for *LCP-p*) together with their AutoWeighted total.

D.1 *LCP-b* Learning Curves on `GSM8K`

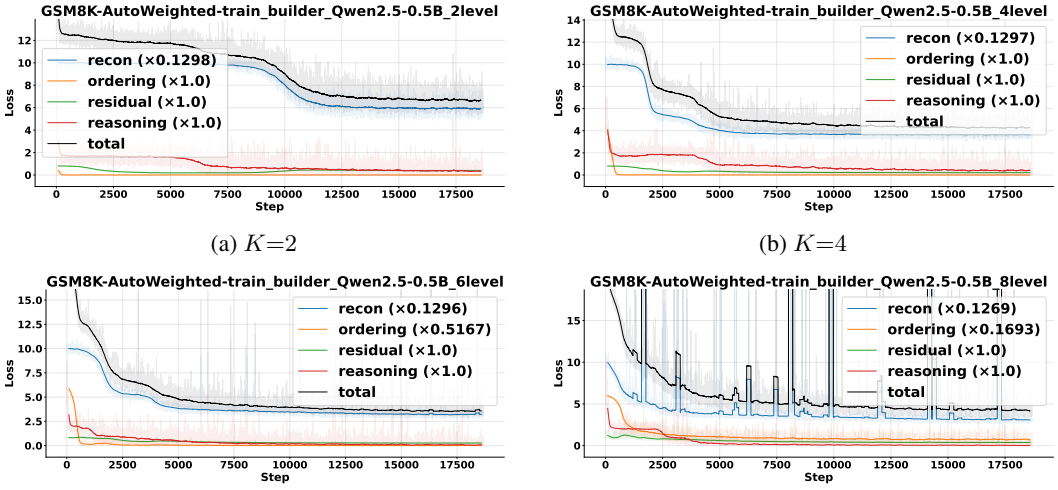
D.2 *LCP-b* Learning Curves on `MATH`

D.3 *LCP-p* Learning Curves on `GSM8K`

D.4 *LCP-p* Ablation Comparisons on `GSM8K`–Qwen2.5-0.5B

The **three reference cells** at $K \in \{2, 4, 6\}$ on `GSM8K`–Qwen2.5-0.5B are also used to isolate two design choices: (i) whether AutoWeighted loss balancing is necessary, by comparing the `independent` predictor with and without AutoWeighting; and (ii) whether per-level parameters should be `independent` or `shared` across pyramid levels, by comparing the two non-AutoWeighted variants. The non-AutoWeighted runs and the `shared` runs are only available at $K \in \{2, 4\}$, so the $K{=}6$ cell appears solely as the AutoWeighted reference in Figures 24 and 3.

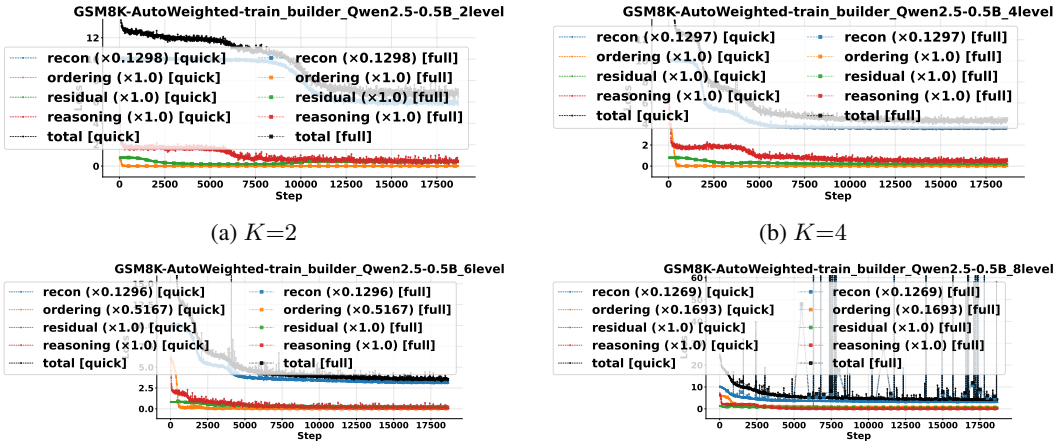
E More Visual Analysis



(a) K=2 (b) K=4

(c) K=6 (d) K=8

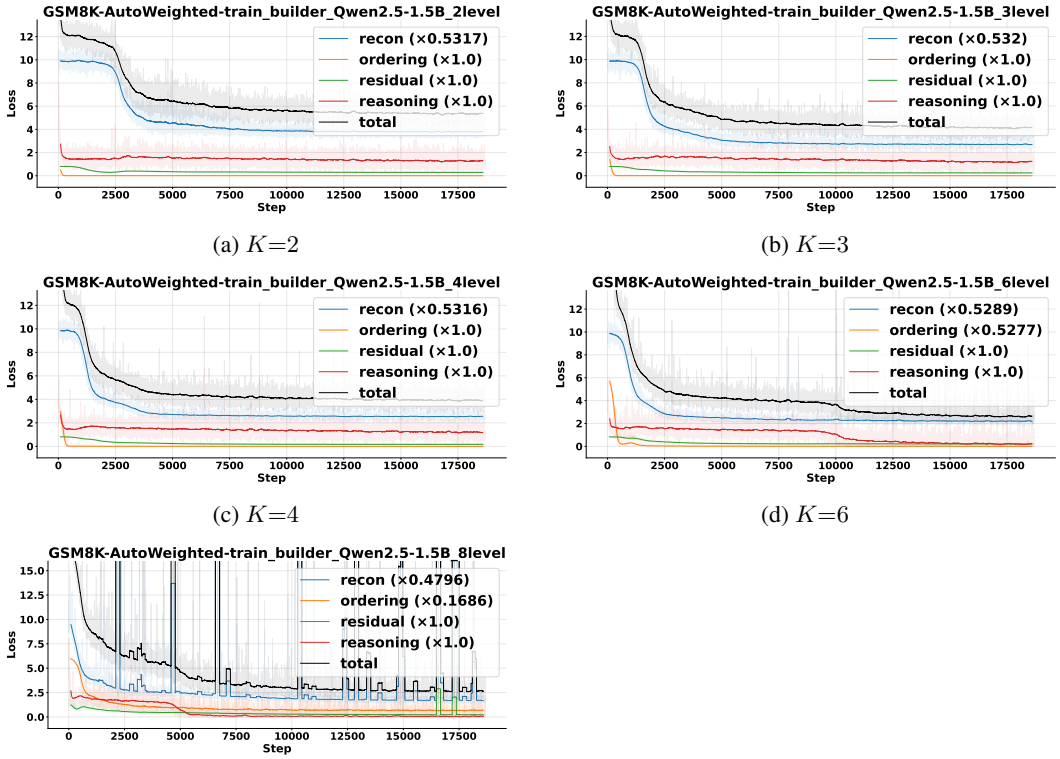
Figure 8: ***LCP-b* training dynamics across pyramid depths on GSM8K–Qwen2.5-0.5B (AutoWeighted).** Each panel overlays the per-step reasoning cross-entropy, concept reconstruction, concept cross-entropy, and the AutoWeighted total, plotted against optimizer steps at a single pyramid depth K; all components decrease monotonically and plateau jointly.



(a) K=2 (b) K=4

(c) K=6 (d) K=8

Figure 9: ***LCP-b* held-out generalization across pyramid depths on GSM8K–Qwen2.5-0.5B (AutoWeighted).** Same per-component overlay as Figure 8 but evaluated on the validation split; curves track the training losses without visible over-fitting across all K.

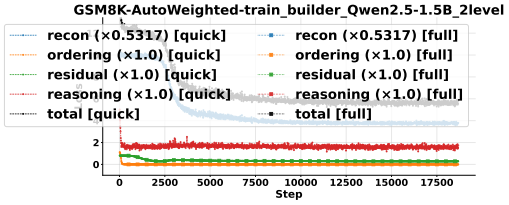


(a) K=2 (b) K=3

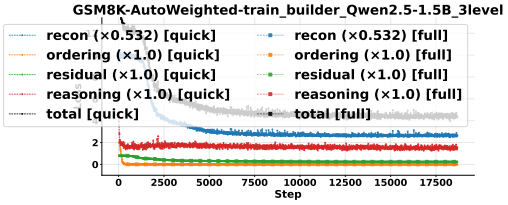
(c) K=4 (d) K=6

(e) K=8

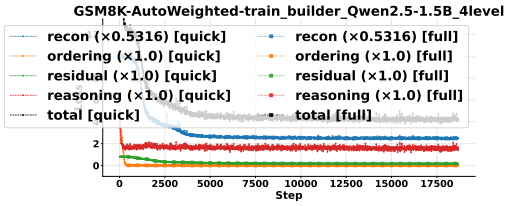
Figure 10: ***LCP-b* training dynamics across pyramid depths on GSM8K–Qwen2.5-1.5B (AutoWeighted).** Per-component training losses (reasoning CE, concept reconstruction, concept CE, AutoWeighted total) against optimizer steps for each $K \in \{2, 3, 4, 6, 8\}$; the larger backbone reaches a lower plateau on every component than the 0.5B counterpart.



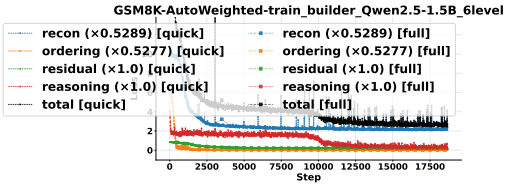
(a) K=2



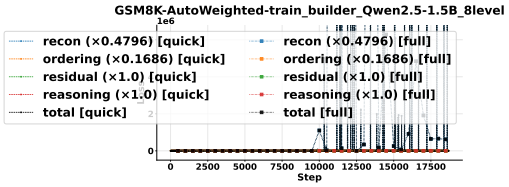
(b) K=3



(c) K=4

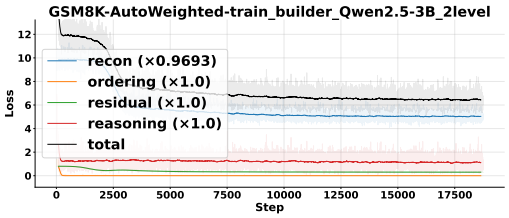


(d) K=6

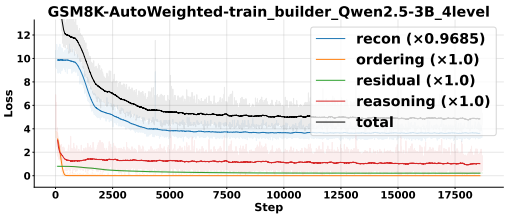


(e) K=8

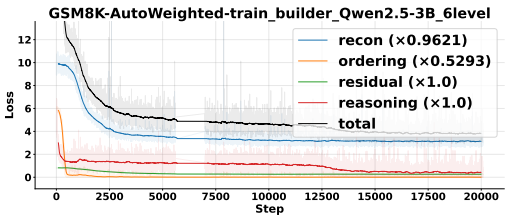
Figure 11: ***LCP-b* held-out generalization across pyramid depths on GSM8K–Qwen2.5-1.5B (AutoWeighted).** Same per-component overlays as Figure 10 on the validation split; evaluation and training curves stay tightly coupled for every K.



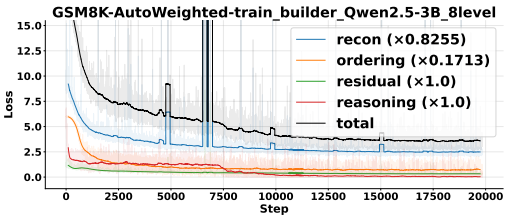
(a) K=2



(b) K=4

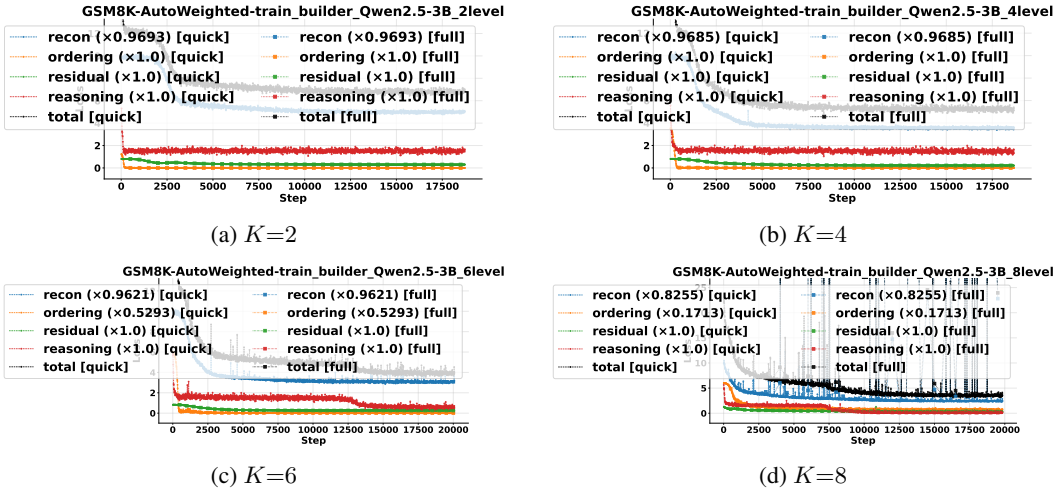


(c) K=6



(d) K=8

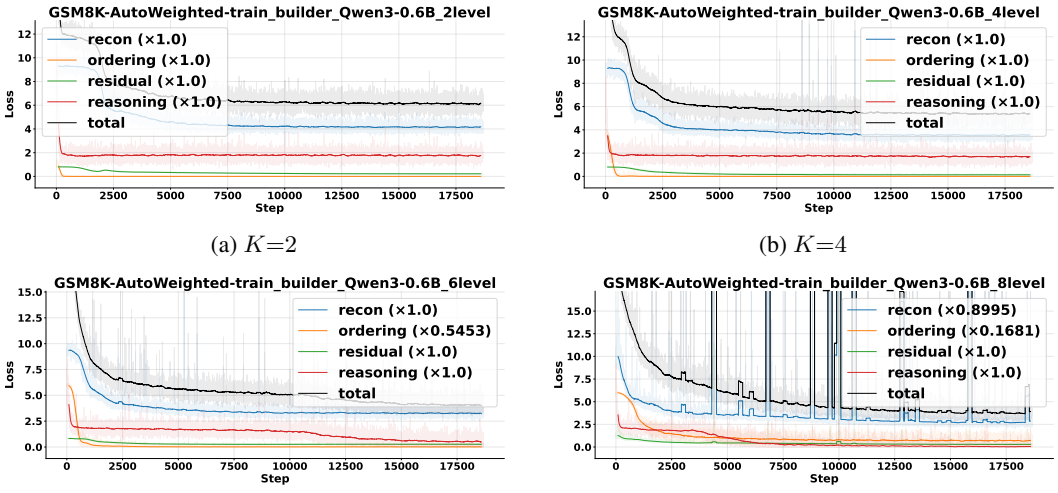
Figure 12: ***LCP-b* training dynamics across pyramid depths on GSM8K–Qwen2.5-3B (AutoWeighted).** Per-component training losses against optimizer steps for $K \in \{2, 4, 6, 8\}$; convergence is consistently faster and to lower absolute levels than at the 0.5B and 1.5B scales.



(a) K=2 (b) K=4

(c) K=6 (d) K=8

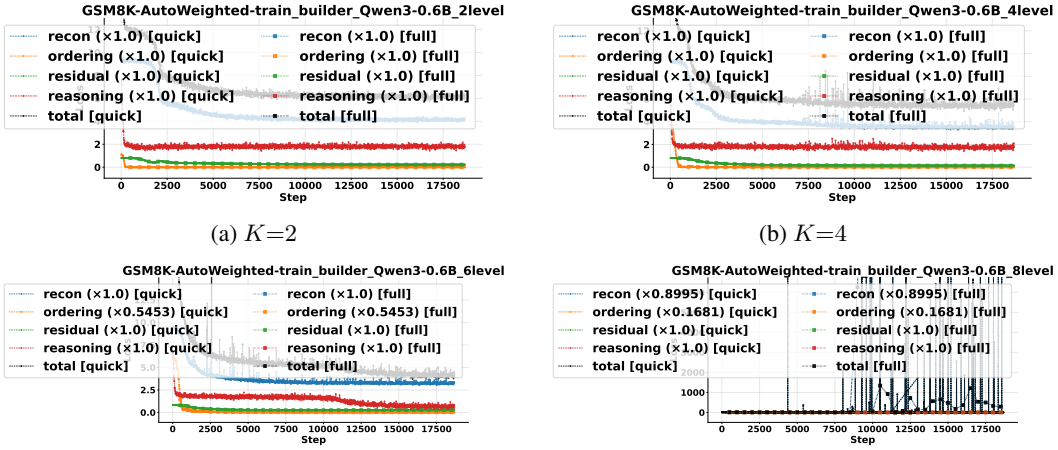
Figure 13: ***LCP-b* held-out generalization across pyramid depths on GSM8K–Qwen2.5-3B (AutoWeighted).** Validation-split counterpart of Figure 12; evaluation losses closely follow the training trajectories across all K.



(a) K=2 (b) K=4

(c) K=6 (d) K=8

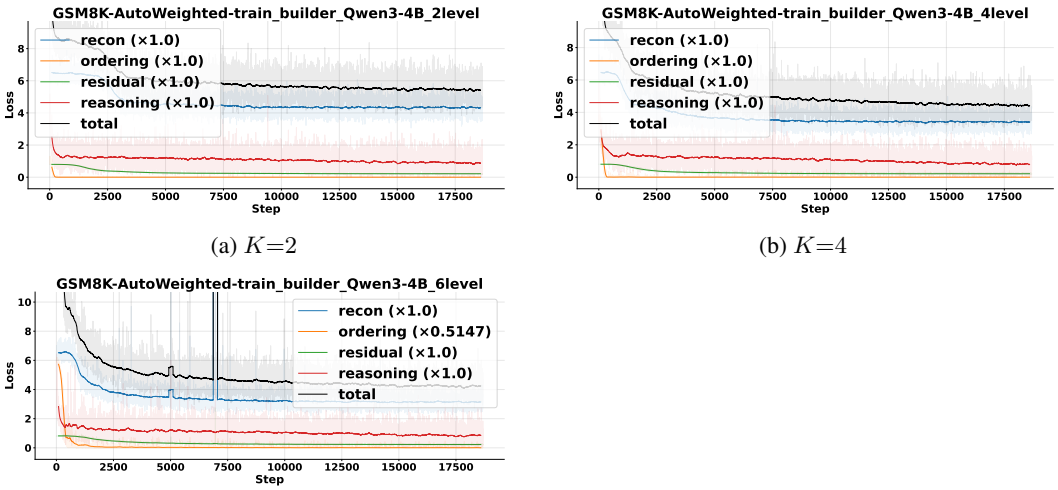
Figure 14: ***LCP-b* training dynamics across pyramid depths on GSM8K–Qwen3-0.6B (AutoWeighted).** Per-component training losses against optimizer steps for $K \in \{2, 4, 6, 8\}$; the Qwen3 family converges on a trajectory qualitatively similar to the size-matched Qwen2.5-0.5B backbone.



(a) K=2 (b) K=4

(c) K=6 (d) K=8

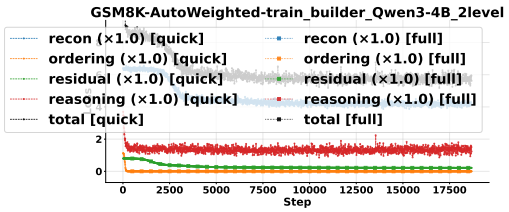
Figure 15: ***LCP-b* held-out generalization across pyramid depths on GSM8K–Qwen3-0.6B (AutoWeighted).** Validation-split counterpart of Figure 14; the evaluation components remain flat-to-decreasing throughout training.



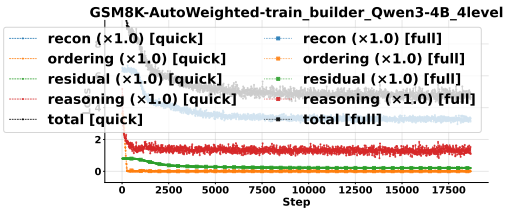
(a) K=2 (b) K=4

(c) K=6

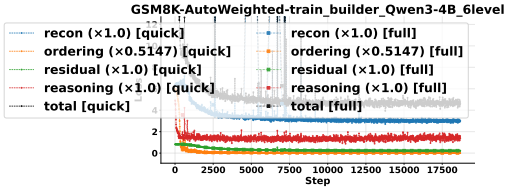
Figure 16: ***LCP-b* training dynamics across pyramid depths on GSM8K–Qwen3-4B (AutoWeighted).** Per-component training losses against optimizer steps for $K \in \{2, 4, 6\}$; the 4B backbone delivers the lowest stable losses of all tested backbones at each K.



(a) K=2

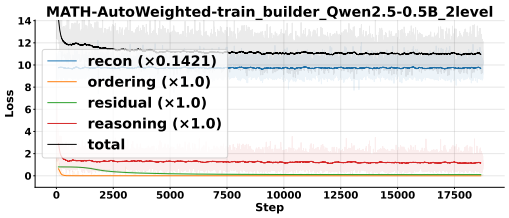


(b) K=4

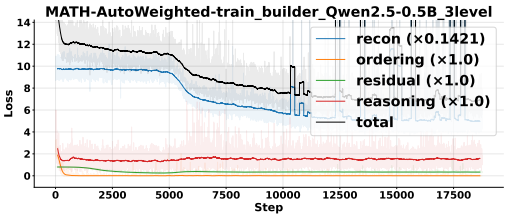


(c) K=6

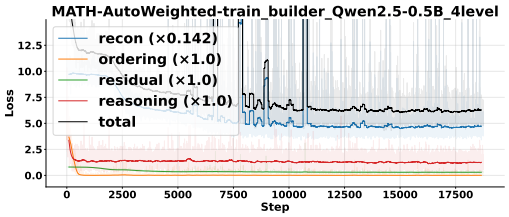
Figure 17: ***LCP-b* held-out generalization across pyramid depths on GSM8K–Qwen3-4B (AutoWeighted).** Validation-split counterpart of Figure 16; evaluation curves track training curves without observable divergence for every K.



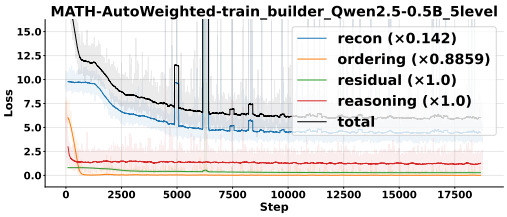
(a) K=2



(b) K=3



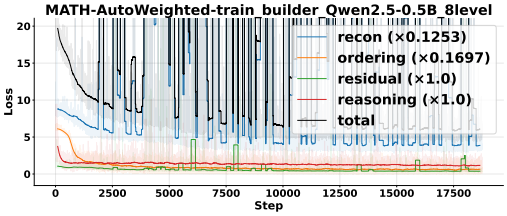
(c) K=4



(d) K=5

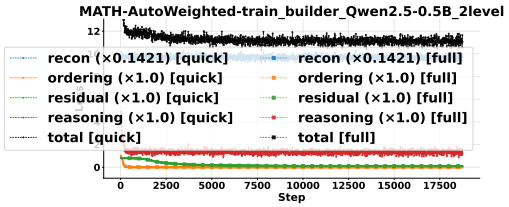


(e) K=6



(f) K=8

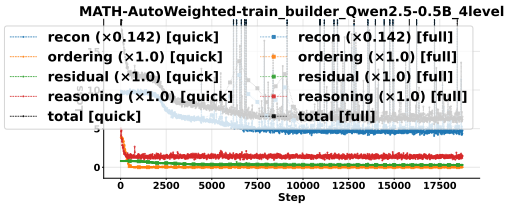
Figure 18: ***LCP-b* training dynamics across pyramid depths on MATH–Qwen2.5-0.5B (AutoWeighted).** Per-component training losses (reasoning CE, concept reconstruction, concept CE, AutoWeighted total) against optimizer steps for $K \in \{2, 3, 4, 5, 6, 8\}$; depths closer to the dataset-default K=5 converge to the lowest reasoning-CE plateau.



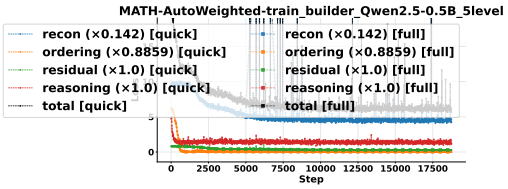
(a) K=2



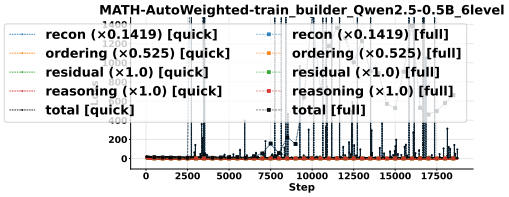
(b) K=3



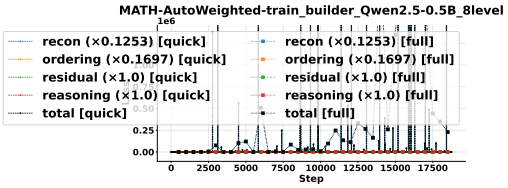
(c) K=4



(d) K=5

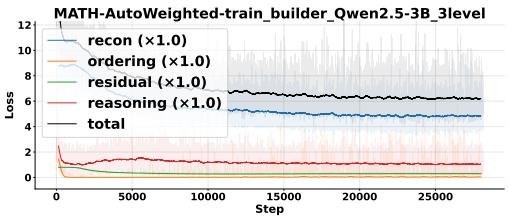


(e) K=6



(f) K=8

Figure 19: ***LCP-b* held-out generalization across pyramid depths on MATH–Qwen2.5-0.5B (AutoWeighted).** Validation-split counterpart of Figure 18; the evaluation-curve ordering across K mirrors the training-curve ordering, with K=5 attaining the best validation losses.

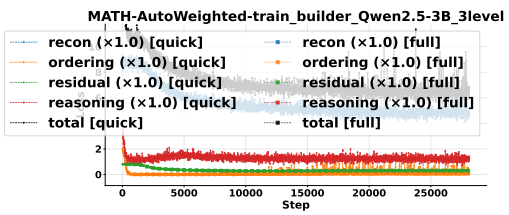


(a) K=3

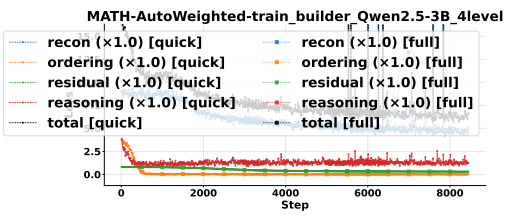


(b) K=4

Figure 20: ***LCP-b* training dynamics on MATH–Qwen2.5-3B (AutoWeighted).** Per-component training losses against optimizer steps at the two depths with completed runs ($K \in \{3, 4\}$); the larger backbone reaches substantially lower reasoning-CE plateaus than the 0.5B counterpart at the same K.

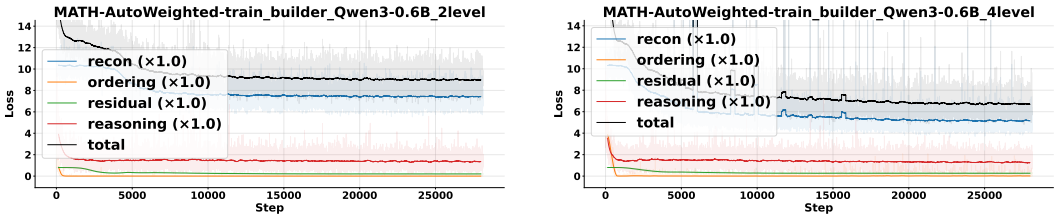


(a) K=3



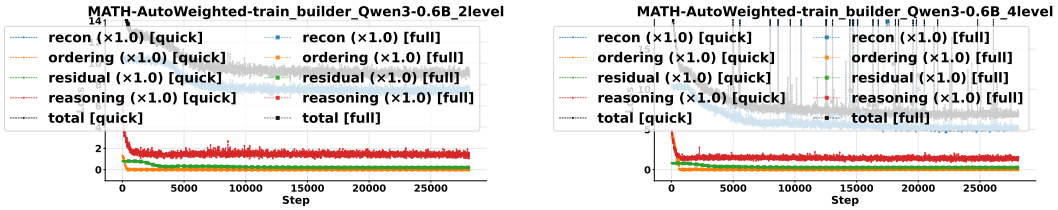
(b) K=4

Figure 21: ***LCP-b* held-out generalization on MATH–Qwen2.5-3B (AutoWeighted).** Validation-split counterpart of Figure 20; evaluation and training curves remain tightly coupled over both depths.



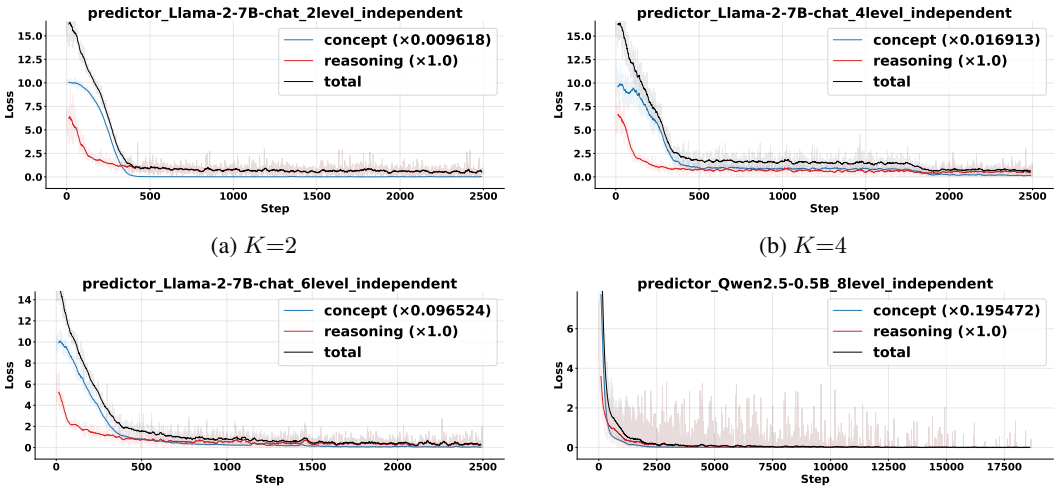
(a) K=2 (b) K=4

Figure 22: ***LCP-b* training dynamics on MATH–Qwen3-0.6B (AutoWeighted).** Per-component training losses against optimizer steps for $K \in \{2, 4\}$; the Qwen3 family behaves consistently with the size-matched Qwen2.5-0.5B backbone on the same dataset.



(a) K=2 (b) K=4

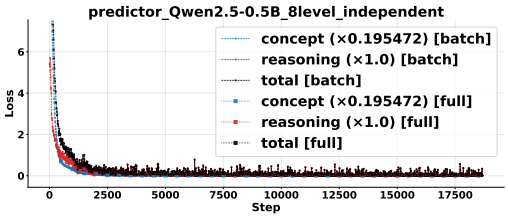
Figure 23: ***LCP-b* held-out generalization on MATH–Qwen3-0.6B (AutoWeighted).** Validation-split counterpart of Figure 22; evaluation losses follow the training trajectories with no observable over-fitting at either depth.



(a) K=2 (b) K=4

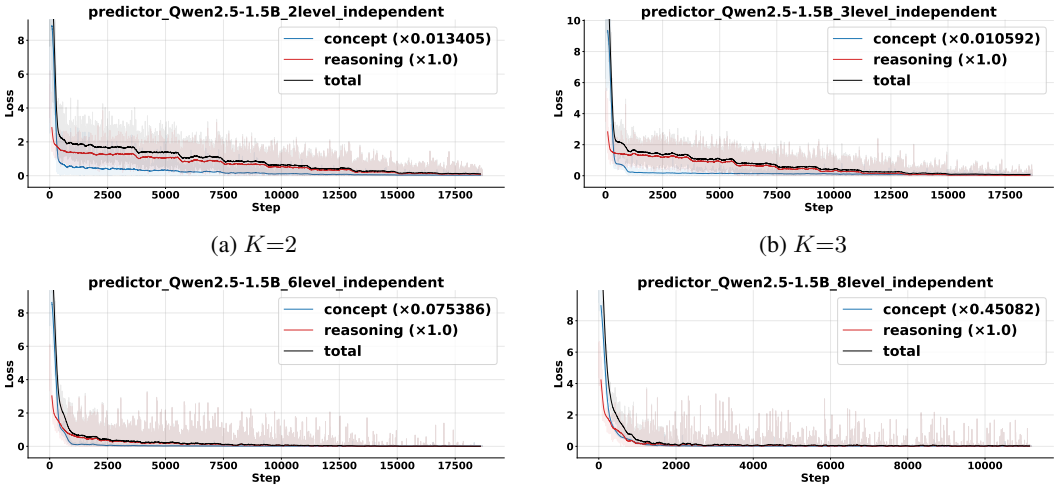
(c) K=6 (d) K=8

Figure 24: ***LCP-p* training dynamics across pyramid depths on GSM8K–Qwen2.5-0.5B (independent, AutoWeighted).** Each panel overlays the predictor's per-step concept MSE, concept cross-entropy, and the AutoWeighted total against optimizer steps for a single pyramid depth K; both supervision terms decrease in lockstep thanks to AW balancing.



(a) K=8

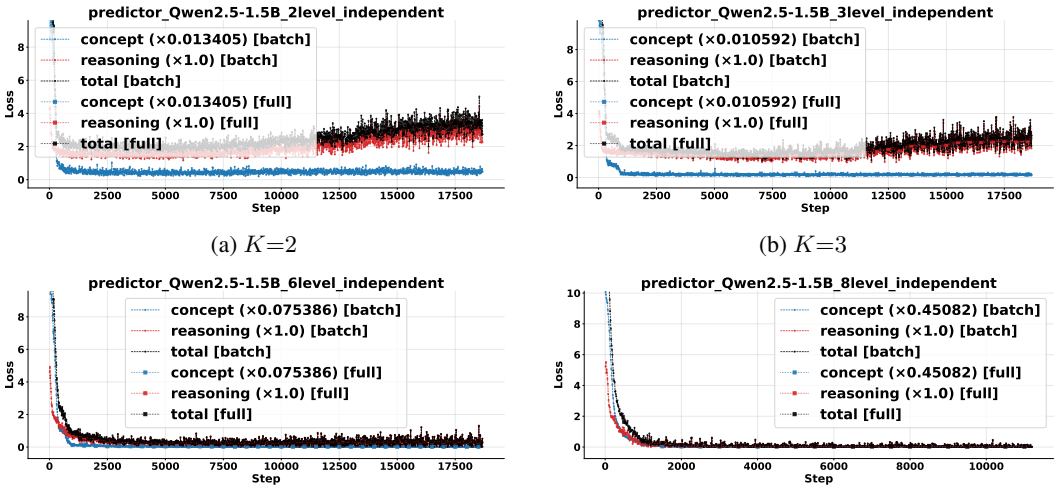
Figure 25: ***LCP-p* held-out generalization at K=8 on GSM8K–Qwen2.5-0.5B (independent, AutoWeighted).** Validation-split concept-MSE, concept-CE, and AutoWeighted total against training progress; the $K \in \{2, 4, 6\}$ counterparts are reported in Figure 3 of the main paper.



(a) K=2 (b) K=3

(c) K=6 (d) K=8

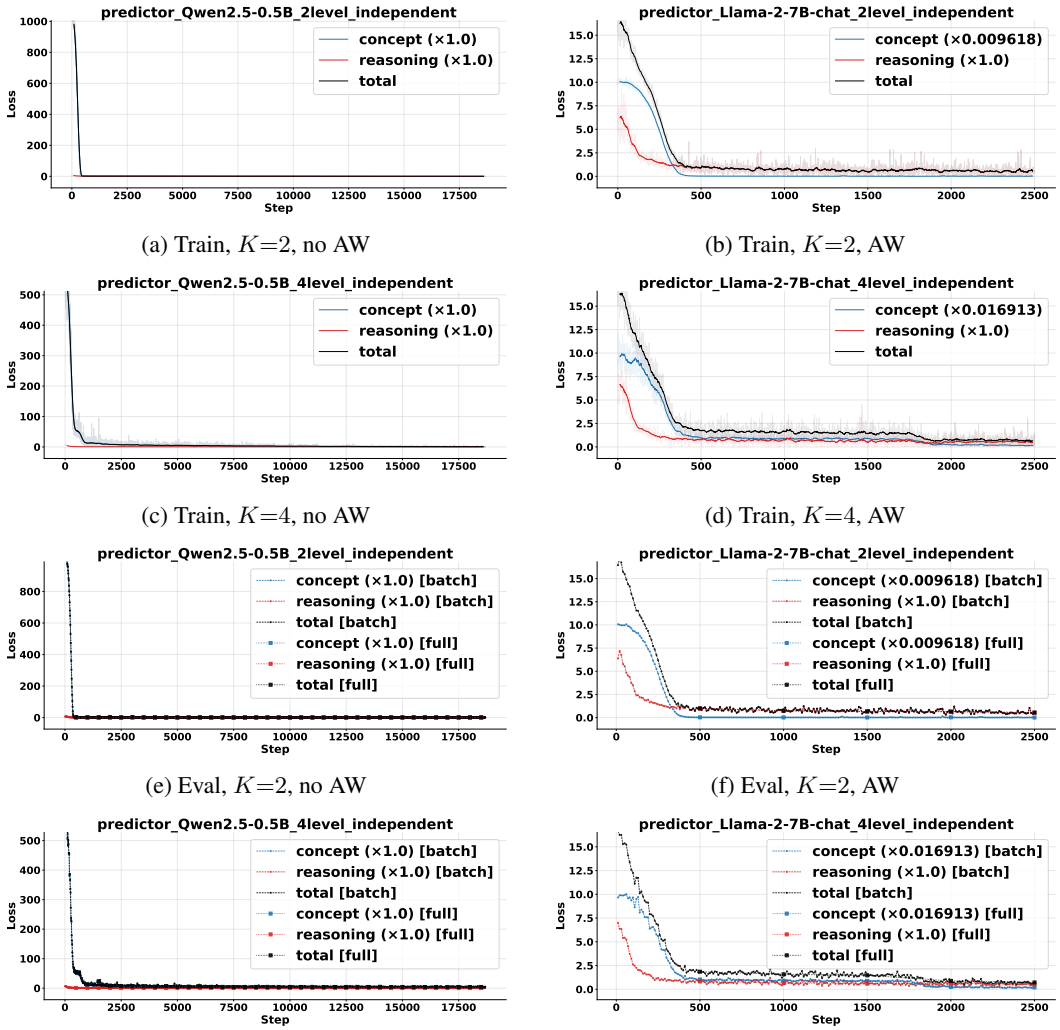
Figure 26: ***LCP-p* training dynamics across pyramid depths on GSM8K–Qwen2.5-1.5B (independent, AutoWeighted).** Predictor concept-MSE, concept-CE, and AutoWeighted total against optimizer steps for $K \in \{2, 3, 6, 8\}$; the 1.5B predictor reaches a lower plateau on both supervision components than the 0.5B counterpart.



(a) K=2 (b) K=3

(c) K=6 (d) K=8

Figure 27: ***LCP-p* held-out generalization across pyramid depths on GSM8K–Qwen2.5-1.5B (independent, AutoWeighted).** Validation-split counterpart of Figure 26; evaluation curves shadow the training curves for every K with no sign of over-fitting.



(a) Train, K=2, no AW

(b) Train, K=2, AW

(c) Train, K=4, no AW

(d) Train, K=4, AW

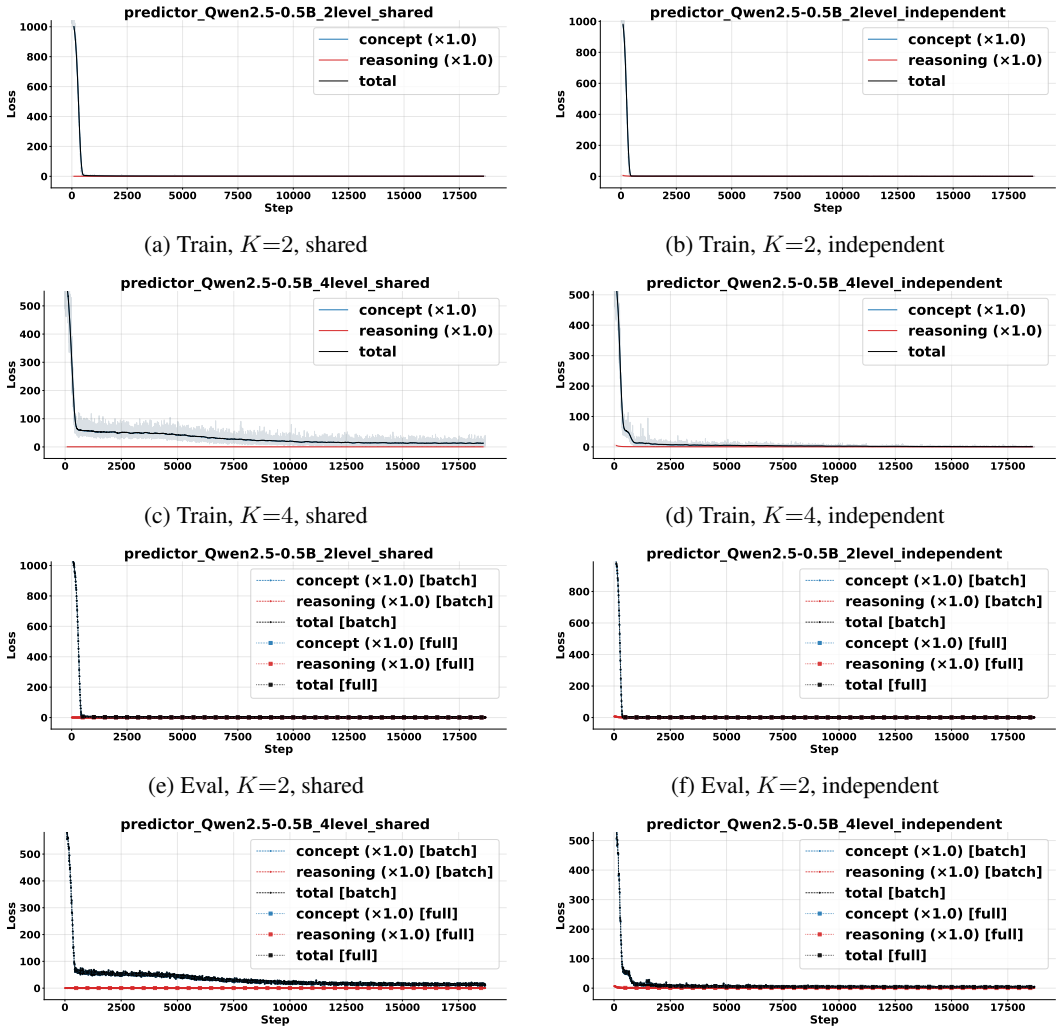
(e) Eval, K=2, no AW

(f) Eval, K=2, AW

(g) Eval, K=4, no AW

(h) Eval, K=4, AW

Figure 28: **AutoWeighted balancing is necessary for stable *LCP-p* optimization.** Each panel overlays the predictor's concept MSE, concept cross-entropy, and AutoWeighted (or unweighted) total for the `independent` variant with and without AW, on `GSM8K`–Qwen2.5-0.5B. Rows 1–2 show training losses; rows 3–4 show evaluation losses, at $K \in \{2, 4\}$. Without AW the individual components fall outside the calibrated envelope of Section C.2, producing noticeably less stable training dynamics and higher evaluation losses than the AW counterpart at the same K.



(a) Train, K=2, shared

(b) Train, K=2, independent

(c) Train, K=4, shared

(d) Train, K=4, independent

(e) Eval, K=2, shared

(f) Eval, K=2, independent

(g) Eval, K=4, shared

(h) Eval, K=4, independent

Figure 29: **Per-level `independent` parameters outperform `shared` parameters in *LCP-p*.** Each panel overlays the predictor's concept MSE, concept cross-entropy, and unweighted total (both runs are without AW for a controlled comparison) on `GSM8K`–Qwen2.5-0.5B. Rows 1–2 show training losses; rows 3–4 show evaluation losses, at $K \in \{2, 4\}$. The `independent` variant consistently reaches lower losses than the `shared` variant at the same K on both splits, motivating the per-level parameterization used in the main results.

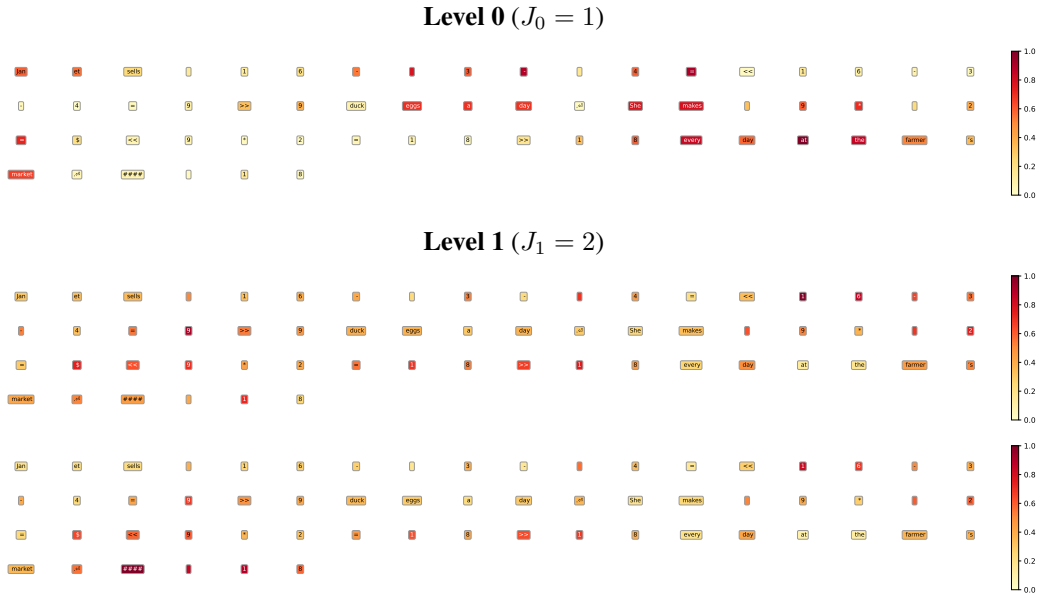


Figure 30: **Per-concept attention weights over CoT tokens** for Sample 1, Levels 0–1 (`GSM8K`, Qwen2.5-0.5B, $K = 4$). Each row visualizes the attention distribution of one concept slot. The learned soft boundaries cause each concept to focus on overlapping segments of the CoT.

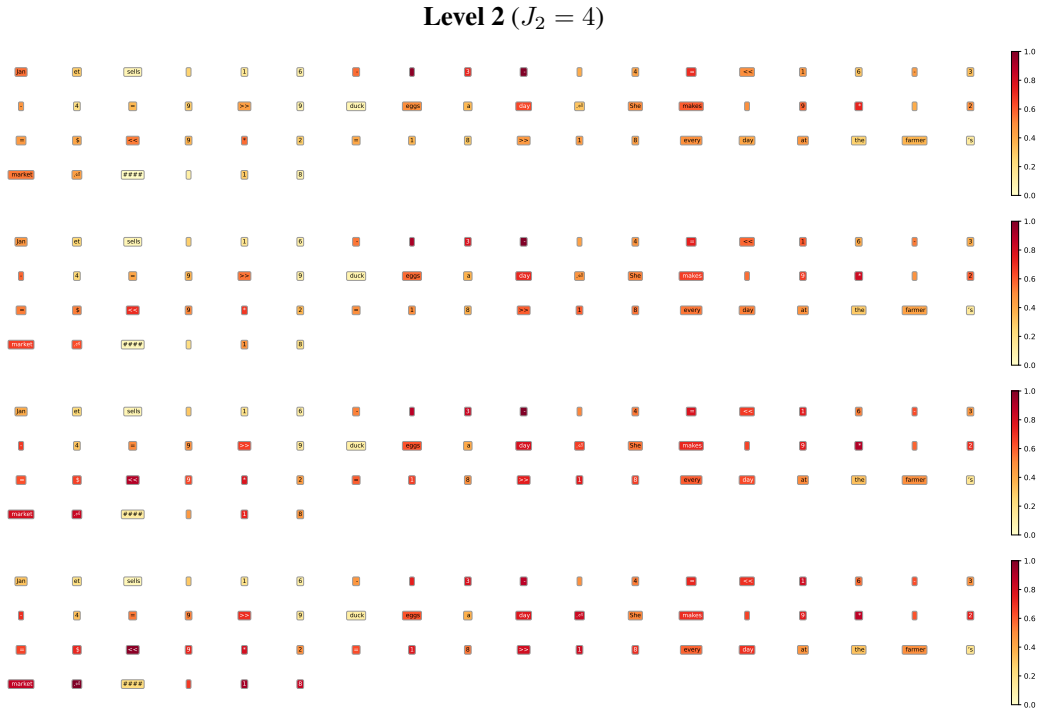


Figure 31: **Per-concept attention weights over CoT tokens** for Sample 1, Level 2 (`GSM8K`, Qwen2.5-0.5B, $K = 4$). The four concept slots at this level each attend to approximately one quarter of the CoT, partitioned by learned soft boundaries.

Level 3 ($J_3 = 8$, showing slots 0, 2, 4, 6)

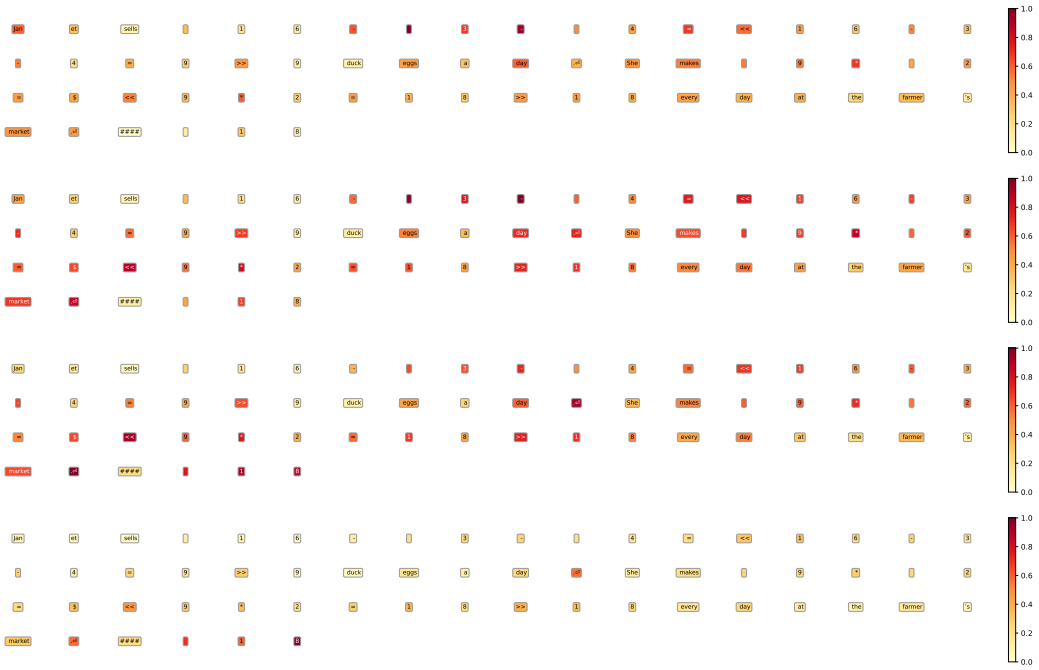


Figure 32: **Per-concept attention weights over CoT tokens** for Sample 1, Level 3 (`GSM8K`, Qwen2.5-0.5B, $K = 4$). Slots 0, 2, 4, 6 are shown (even-indexed). Each slot attends to a short, localized segment; together the eight slots tile the full CoT via soft boundaries.

Level 0 ($J_0 = 1$)



Level 1 ($J_1 = 2$)

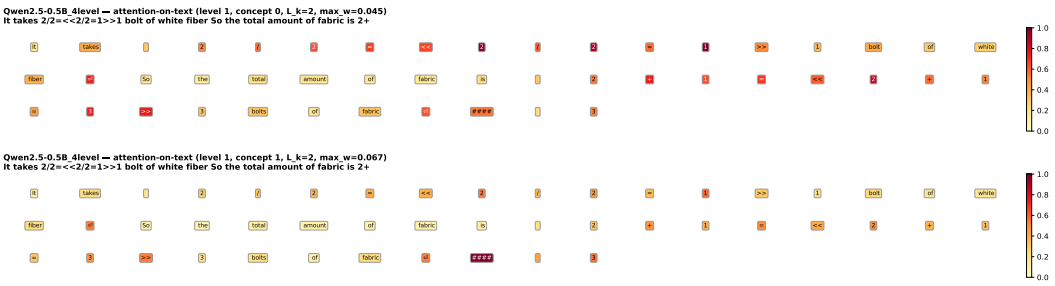


Figure 33: **Per-concept attention weights over CoT tokens** for Sample 2, Levels 0–1 (`GSM8K`, Qwen2.5-0.5B, $K = 4$). The same coarse-to-fine pattern is observed: the single level-0 concept attends globally while the two level-1 concepts split the CoT into two halves.

Level 2 ($J_2 = 4$)

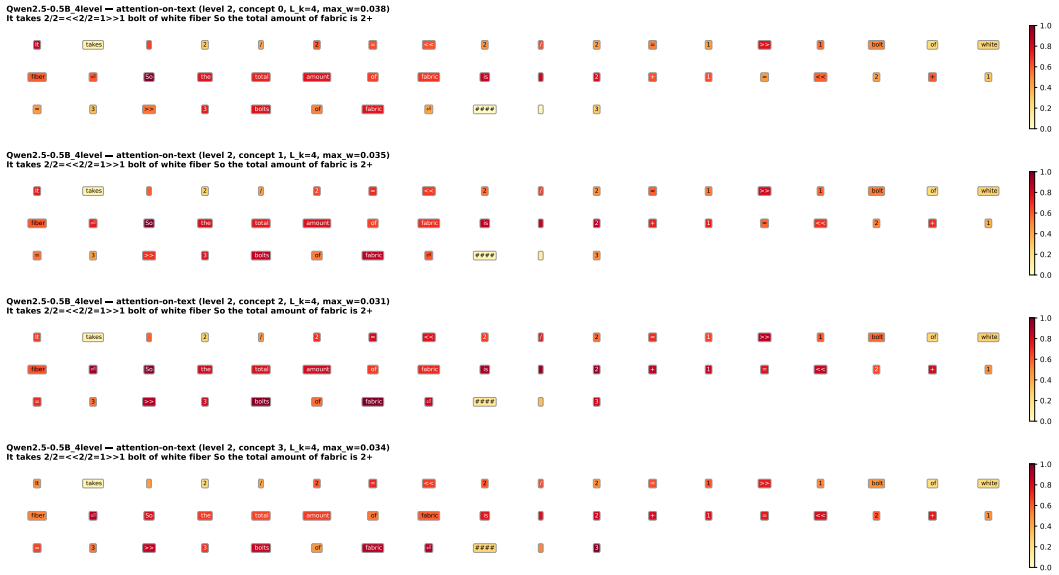


Figure 34: **Per-concept attention weights over CoT tokens** for Sample 2, Level 2 (GSM8K, Qwen2.5-0.5B, $K = 4$). Each of the four concepts focuses on a distinct quarter of the reasoning trace, demonstrating the soft-boundary segmentation.

Level 3 ($J_3 = 8$, showing slots 0, 2, 4, 6)

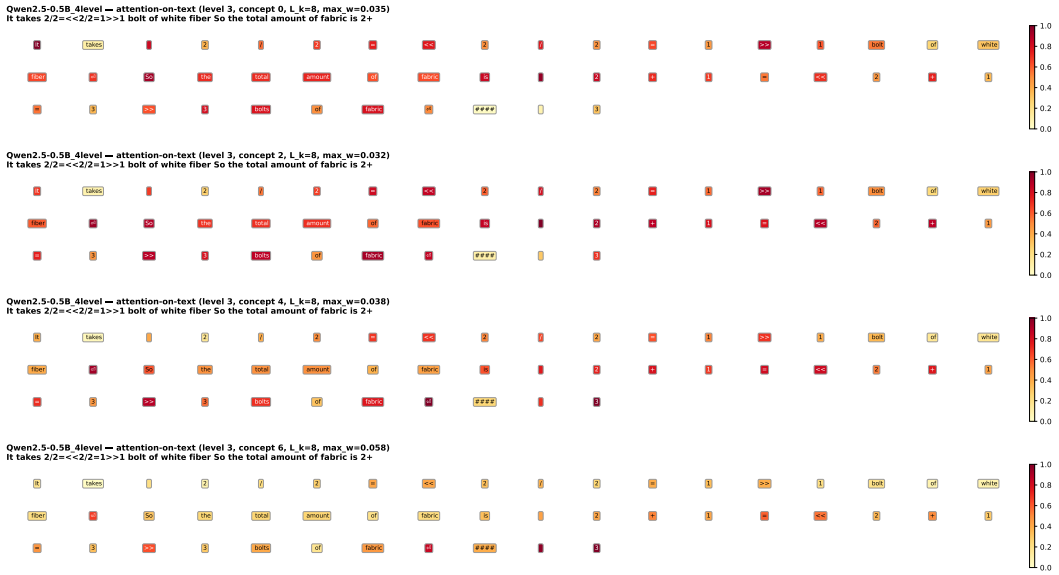
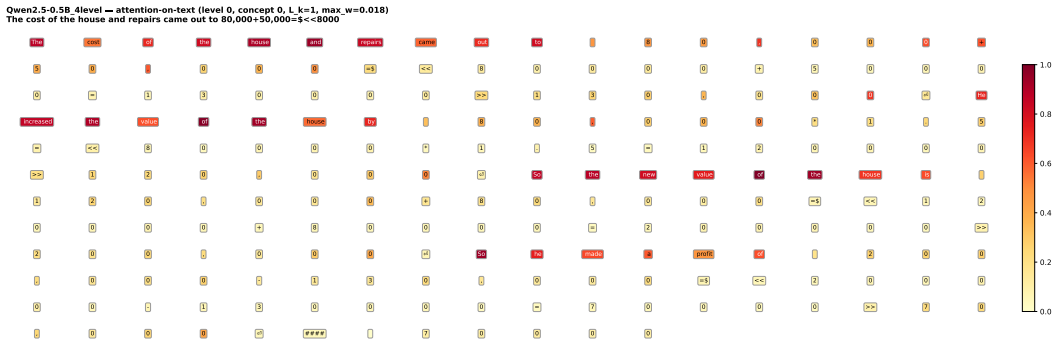


Figure 35: **Per-concept attention weights over CoT tokens** for Sample 2, Level 3 (GSM8K, Qwen2.5-0.5B, $K = 4$). Even-indexed slots shown. The finer granularity results in each concept attending to a narrow, localized reasoning step.

Level 0 ($J_0 = 1$)



Level 1 ($J_1 = 2$)

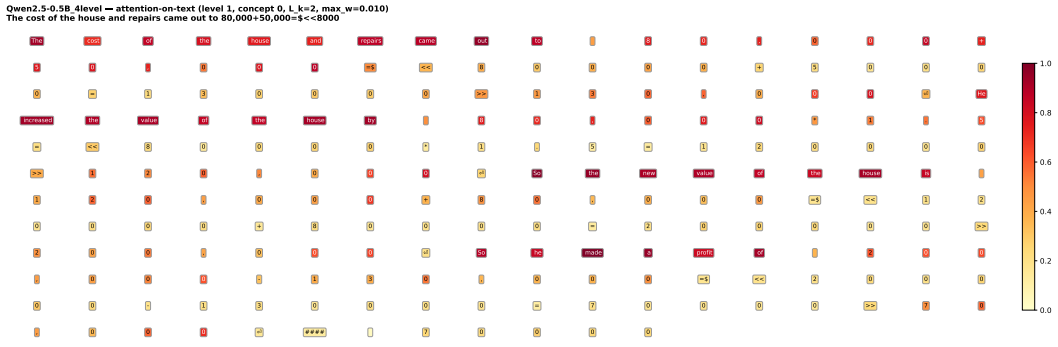


Figure 36: **Per-concept attention weights over CoT tokens** for Sample 3, Levels 0–1 (`GSM8K`, Qwen2.5-0.5B, $K = 4$). Consistent with other samples, the level-0 concept captures global context while level-1 concepts bisect the trace.

Level 2 ($J_2 = 4$)

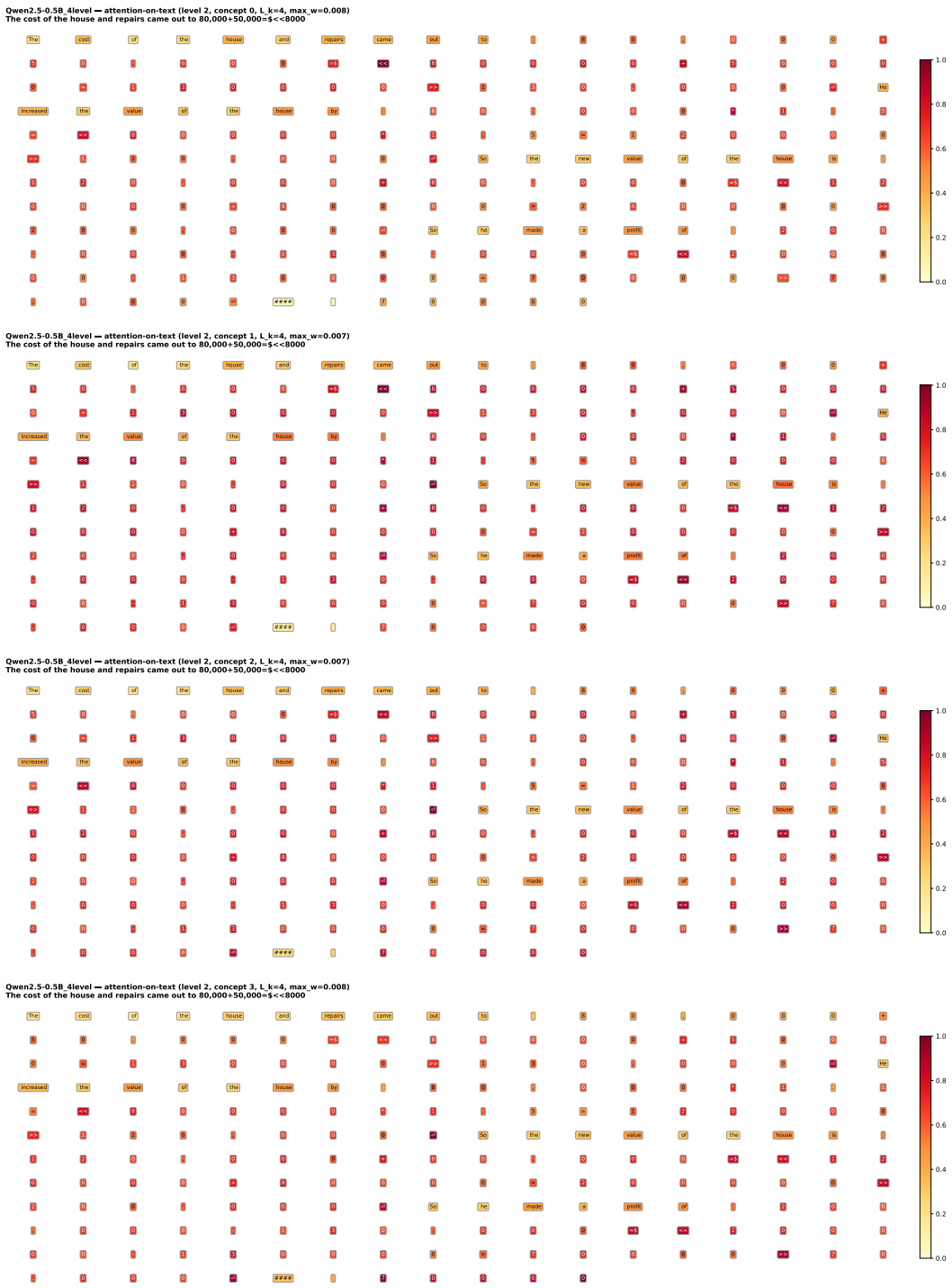


Figure 37: **Per-concept attention weights over CoT tokens** for Sample 3, Level 2 (GSM8K, Qwen2.5-0.5B, $K = 4$). Four concepts partition the CoT into four segments via soft boundaries, each encoding a distinct sub-goal of the reasoning process.

Level 3 ($J_3 = 8$, showing slots 0, 2, 4, 6)

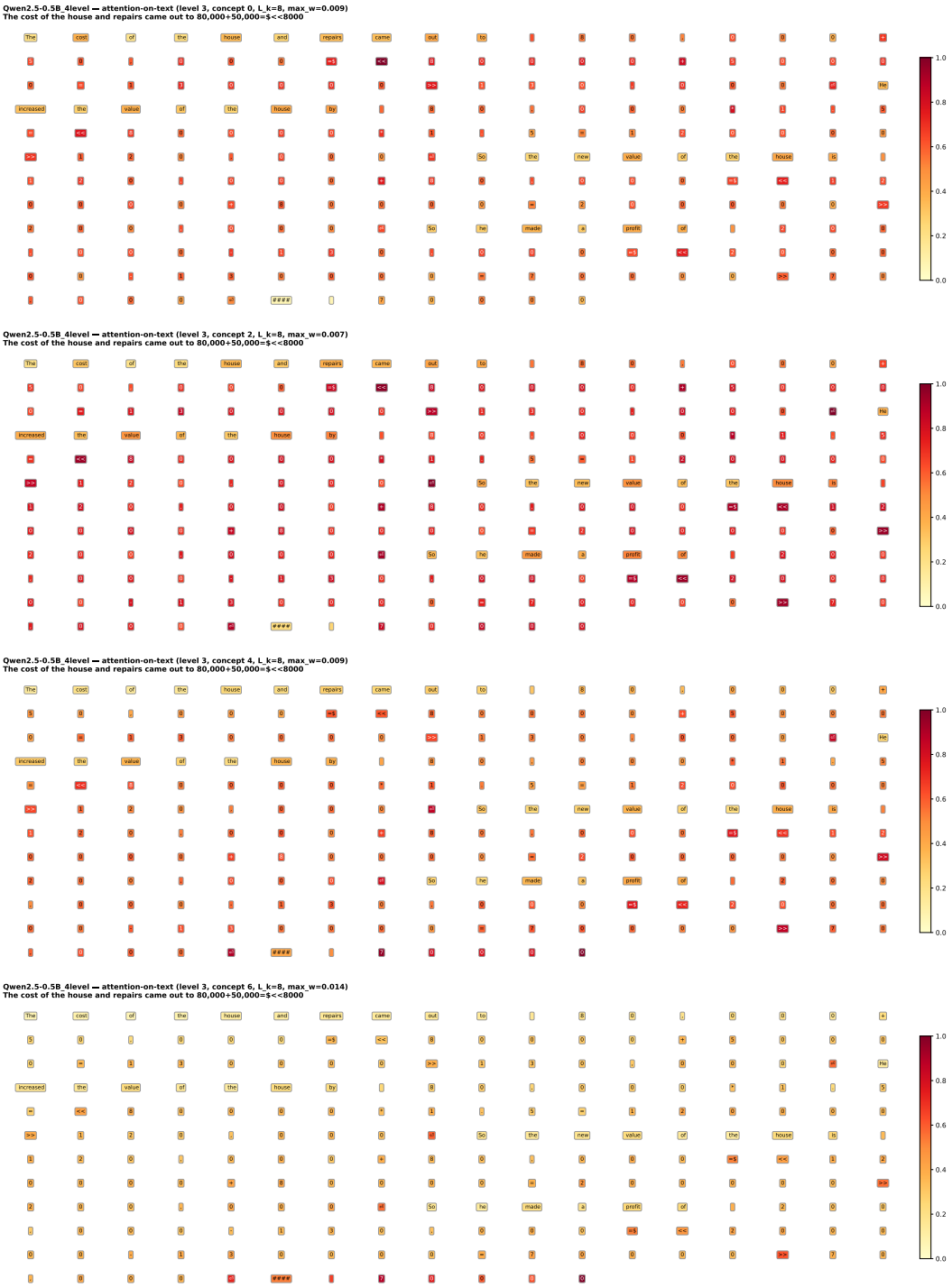


Figure 38: **Per-concept attention weights over CoT tokens** for Sample 3, Level 3 (GSM8K, Qwen2.5-0.5B, $K = 4$). Even-indexed slots shown. Across all three samples, the visualization confirms that learned soft boundaries.

F Additional Analyses

This appendix collects three additional analyses: the depth ceiling that follows from the CoT length of the training corpus (Appendix F.1), the accounting of inference cost (Appendix F.2), and the protocol for code generation (Appendix F.3).

F.1 Depth Ceiling

Since $J_k = b^k$ and the finest level cannot exceed the width of the CoT it abstracts, whose mean length in the training corpus is T_{cot}, the admissible depth satisfies $b^{K-1} \leq T_{\text{cot}}$, i.e., $K \leq \lfloor \log_b T_{\text{cot}} \rfloor + 1$. The ceiling is therefore a property of the training corpus rather than a free hyper-parameter: it rises with the length of the traces, so each training corpus has its own admissible depth range (Table 5).

Table 5: Depth ceiling at b=2 for each training corpus, with T_{cot} the mean CoT length reported in Appendix C.1. The K column reports the depth used for the corresponding benchmarks in Table 1.

Training corpus	Mean CoT length T_{cot}	Ceiling $\lfloor \log_2 T_{\text{cot}} \rfloor + 1$	K in Table 1	Benchmarks evaluated
`GSM8K`	120	7	4	`GSM8K`, `MultiArith`
`MATH`	148	8	8	`MATH`, `MATH-500`, `AIME2024`
`codeforces-cots`	943	10	8	`HumanEval`

The depths used in Table 1 all lie within these ceilings, and on `MATH` the ceiling coincides with the depth at which Figure 4 saturates. Beyond the ceiling the rank budget $\min(J_k, T_{\text{cot}}, d_c)$ saturates at T_{cot} (Section 4.1), so additional levels add latency without contributing new independent directions.

F.2 Efficiency Accounting

Table 6 reports the measured FLOPs of one full pyramid inference, i.e., accumulated over the K sequential passes.

Table 6: Predictor inference FLOPs for one full pyramid inference (GFLOPs), accumulated over the K sequential passes and measured with PyTorch `FlopCounterMode` at batch size 1. The Builder is a training-time teacher and is excluded from the inference cost; – denotes a configuration for which no predictor checkpoint was trained.

Model	K=2	K=4	K=6	K=8
Qwen2.5-0.5B	–	291.68	519.50	1,043.89
Qwen2.5-1.5B	425.56	907.50	1,615.58	3,238.09
Qwen3-0.6B	166.74	355.74	634.64	1,286.34

Total inference FLOPs grow roughly linearly with the pyramid depth K, because the predictor performs K sequential backbone passes (one per level), and near-linearly with the backbone parameter count at fixed K. This arithmetic cost does not translate proportionally into latency: an autoregressive decode performs one pass per token and re-reads the full backbone weights to produce that single token, so its wall-clock cost is governed by memory traffic rather than by arithmetic, whereas *LCP* produces a whole level in one pass, amortizing each weight read over all J_k concepts of that level. Latency therefore scales with K rather than with the T tokens an explicit chain generates, which is the source of the reduction reported in Table 1. FLOPs bound the arithmetic a pass performs; they do not by themselves determine the wall-clock cost of generating a reasoning trace.

F.3 Code Generation with *LCP*

HumanEval asks for an executable Python function and is scored by pass@1 under the official execution harness. *LCP* enters this setting unchanged because only the intermediate reasoning is latent: from the function signature and docstring $\mathbb{Q}$, *LCP-p* predicts the concept pyramid $\hat{\mathbf{P}}$ level by level (the apex encodes the algorithm outline, intermediate levels the sub-goals, the base the local steps and edge cases), and the frozen backbone then generates the complete Python body in

language space conditioned on $\mathbb{Q}$ and $\hat{\mathbf{P}}$; the resulting program is executed against the hidden tests. Every token of the submitted program is therefore still produced by the language model, so syntax, API usage and executability remain the backbone's responsibility, while the pyramid replaces the step-by-step CoT that *LCP-b* was trained to compress (`codeforces-cots`, mean CoT length 943 tokens, depth ceiling 10; we use $K{=}8$ within it). Because the answer here is a full program rather than a short numeric string, the latency reported for code generation in Table 1 counts only the K concept passes that produce the reasoning (Section 5.1). The latent representation thus affects what the model plans to compute rather than whether the program can run.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Section 1) state three main contributions: hierarchical concept-pyramid formulation, significant token-cost and inference-latency reduction with competitive accuracy, and interpretability via pyramid decoding. The accuracy claim is explicitly scoped to a latency budget (Section 5.2), the zero-shot claims are limited to the benchmarks listed there, and every claim is supported by Section 5 and the appendix, including the additional analyses of Appendix F.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations section (Appendix A) discusses the fixed pyramid configuration and the two-stage optimization with a frozen builder, the intra-level exchangeability assumption, the empirical scope restricted to the Qwen families and two reasoning domains, and the interpretability ceiling imposed by the auxiliary decoder.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include formal theorems or proofs; it is an empirical contribution presenting a new architecture and training methodology.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 5.1 specifies all training hyper-parameters (optimizer, learning rate, epochs, batch size, seed, precision, gradient clipping), the depth-selection rule, and the measurement-based loss-weight rule. The appendix further documents the warm-up raw magnitudes used by that rule, the per-cell loss weights, the FLOPs and latency measurement protocol, and the hardware. All training datasets are publicly available and the code is released.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code, training scripts, and comprehensive training details are publicly available at https://github.com/AgenticFinLab/Latent-Concept-Reasoning (see the footnote in the abstract). All training datasets (GSM8K, MATH, codeforces-cots) and backbone models (Qwen2.5/Qwen3 family) are publicly available; the paper additionally provides sufficient architectural and hyper-parameter detail to reproduce the method independently.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 and the appendix (Experimental Setup Details) report all necessary information: optimizer (AdamW), learning rate schedules, batch size, epochs, data splits, LoRA configuration, the depth-selection rule ($K \leq \lfloor \log_2 T \rfloor + 1$), the measurement-based loss-weight rule, the FLOPs/latency measurement protocol, and the evaluation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report single-seed (seed=42) results without error bars. Running multiple seeds across all (backbone, K, dataset) cells would be prohibitively expensive given the two-stage training pipeline. The FLOPs measurement of Appendix F is deterministic given a checkpoint and is reported as a supplementary cost metric, not as a statistical test.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The appendix (Experimental Setup Details) specifies training hardware ($4\times$ NVIDIA RTX 5090 32 GB), inference hardware (NVIDIA H100 80 GB and a single RTX 5090), CUDA version, and per-run training wall-clock (e.g., Qwen2.5-0.5B at $K{=}4$: $\approx$ 2 h for the Builder stage and $\approx$ 2 h for the Predictor stage from the training logs); Table 1 reports per-instance inference latency, and Appendix F additionally reports measured FLOPs for one full pyramid inference.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available datasets and open-weight models, does not involve human subjects, and does not present foreseeable negative societal impacts beyond those inherent to LLM efficiency research.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section B (Broader Implications) in the appendix discusses positive impacts including reduced compute/energy at deployment scale, process-level supervision interfaces, and potential for long-context compression. The work is foundational and does not introduce direct negative societal impacts beyond those shared by all LLM efficiency research.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper proposes a training methodology rather than releasing a pre-trained model or dataset. The approach builds on existing publicly available models (Qwen family) and does not introduce new risks of misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets (GSM8K, MATH, codeforces-cots, HumanEval) and backbone models (Qwen2.5/Qwen3) are cited with their original publications. The Qwen models are released under Apache-2.0; datasets used are publicly available for research purposes.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper releases its code base, including training and evaluation scripts and comprehensive training details, at `https://github.com/AgenticFinLab/Latent-Concept-Reasoning`. No new datasets or pre-trained model checkpoints are introduced, and all existing assets used are credited in Appendix C.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs (Qwen2.5/Qwen3 family) are core components of the proposed method: they serve as the backbone for both the Concept Pyramid Builder and the Concept Predictor. Their usage is fully described in Sections 4.1 and 4.2.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.