LiteStage: Latency-aware Layer Skipping for Multi-stage Reasoning

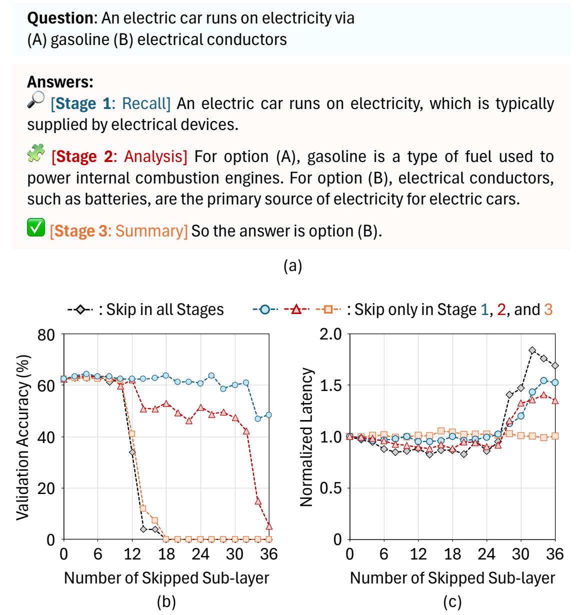
Anonymous ACL submission

Abstract

Multi-stage reasoning has emerged as an effective strategy for enhancing the reasoning capability of small language models by decomposing complex problems into sequential sub-stages. However, this comes at the cost of increased latency. We observe that existing adaptive acceleration techniques, such as layer skipping, struggle to balance efficiency and accuracy in this setting due to two key challenges: (1) stage-wise variation in skip sensitivity, and (2) the generation of redundant output tokens. To address these, we propose LiteStage, a latency-aware layer skipping framework for multi-stage reasoning. LiteStage combines a stage-wise offline search that allocates optimal layer budgets with an online confidence-based generation early exit to suppress unnecessary decoding. Experiments on three benchmarks, *e.g.*, OBQA, CSQA, and StrategyQA, show that LiteStage outperforms prior training-free layer skipping methods.

1 Introduction

Recent research on reasoning spans a broad spectrum, ranging from deep reasoning that rely on long-horizon generation (Hongru et al., 2025; Jin et al., 2024) or self-reflective feedback (Li et al., 2025), to short multi-stage reasoning (Wang et al., 2025a; Yang et al., 2025c). While the former has attracted significant attention, many practical question answering and decision-making tasks fall into the latter category (Chen et al., 2024). In such settings, problems are decomposed into a small number of structured stages such as knowledge recall, local inference, and decision aggregation (Piao and Park, 2024) (see Figure 1(a)), through which they can be effectively solved without requiring long, deeply nested derivations.



Question: An electric car runs on electricity via
(A) gasoline (B) electrical conductors

Answers:
[Stage 1: Recall] An electric car runs on electricity, which is typically supplied by electrical devices.

[Stage 2: Analysis] For option (A), gasoline is a type of fuel used to power internal combustion engines. For option (B), electrical conductors, such as batteries, are the primary source of electricity for electric cars.

☑ **[Stage 3: Summary]** So the answer is option (B).

(a)

(b)

(c)

Figure 1: **Multi-stage Reasoning.** (a) An example of multi-stage reasoning introduced in TinyThinker (Piao and Park, 2024). The process comprises three stages: Stage 1 (*Recall*) generates an initial solution idea, Stage 2 (*Analysis*) evaluates candidate options through explicit reasoning, and Stage 3 (*Summary*) produces the final conclusion. (b)-(c) Accuracy and latency profiles when layer skipping is applied to a *single stage* while keeping the remaining stages at full depth.

This form of *short multi-stage reasoning* is particularly prevalent in small language models, whose limited capacity often prevents reliable reasoning in a single step (Li et al., 2024). However, executing multiple reasoning stages sequentially incurs non-trivial latency (Kim et al., 2024), so inference can remain slow in practice, even for lightweight models. Applying existing acceleration techniques therefore appears natural; however, multi-stage inference introduces unique challenges. Reasoning stages vary substantially in decoding length and token diversity, leading to non-uniform information density across stages (Dai et al., 2024). As a result, some stages tolerate aggressive acceleration, while others are highly sensitive, ultimately bounding achievable accuracy. Effective acceleration thus requires adapting efficiency to the distinct

Table 1: **Comparison of Key Features.** LiteStage is a training-free, stage-wise optimization strategy that adaptively allocates layer budgets across reasoning stages, unlike methods that either apply static layer skipping (training-free) or require additional training for dynamic skipping (*e.g.*, routers).

Method / Contribution	Training-free	Latency-aware	Concise Generation	Sub-layer Skipping	Adaptive Skipping	Importance Metric
LayerSkip (Elhoushi et al., 2024)	✗	✗	-	✗	✓	early exit
MoD (Raposo et al., 2024)	✗	✗	-	✗	✓	router
SkipDecode (Del Corro et al., 2023)	✓	✗	✗	✗	✓	heuristic
UnifiedSkip (Liu et al., 2024)	✓	✗	✗	✗	✗	heuristic
AdaSkip (He et al., 2025)	✓	✗	✗	✓	✗	cosine
LiteStage (Ours)	✓	✓	✓	✓	✓	cosine

computational demands of each reasoning stage.

Adaptive computation in LLMs has been widely explored through *layer skipping* (Raposo et al., 2024; Men et al., 2024; He et al., 2025), which reduces computation by bypassing redundant layers. However, determining how many layers to skip at each reasoning stage remains challenging. As shown in Figure 1(b), the degree of accuracy degradation differs across stages: Stage 1 is notably robust to layer skipping, whereas other stages are more sensitive. In contrast, latency benefits are most pronounced in Stage 2 (see Figure 1(c)), due to its longer generation (**asymmetric trade-offs**). Moreover, approximate decoding often induces models to generate more tokens, which can lead to even slower end-to-end inference than the full-layer, *i.e.*, normalized latency > 1.0, despite reduced per-token latency (**extra generation**).

In this paper, we present **LiteStage**, a latency-aware layer skipping framework designed for multi-stage reasoning in small LLMs. LiteStage comprises two complementary components, an offline configuration and online adjustment, that jointly address the aforementioned key challenges. In the **offline phase**, LiteStage iteratively searches for *layer budget*, *i.e.*, the number of layers to skip, that minimize latency within an accuracy threshold, from the longest stage to the shortest. This stage-wise allocation effectively accelerates slow stages (high priorities) and prevents their accuracy collapse. In the **online phase**, LiteStage addresses the underexplored side effect of layer skipping, the increase in generation length, by monitoring token confidence during decoding and *terminating generation early* when confidence falls below a threshold, thus avoiding redundant generation. LiteStage's key features are summarized in Table 1.

2 Related Works

Multi-stage Generation. Recent works on reasoning tasks employ diverse forms of multi-stage inference. TinyThinker (Piao and Park, 2024) introduces a deductive reasoning cycle of recall, analysis, and summary, showing progressive accuracy gains. DeAR (Xue et al., 2024) adopts a similar decomposition-analysis-rethinking process, refining intermediate answers across stages. CasCoD (Dai et al., 2024) distills decomposed chain-of-thoughts in a cascading manner to enhance reasoning generalization in smaller models. Self-Discover (Zhou et al., 2024) enables models to dynamically organize reasoning structures and select appropriate modules for each problem. LLaVA-CoT (Xu et al., 2025) extends this idea to multi-modal settings (*e.g.*, vision language). However, these works largely prioritize reasoning quality rather than computational efficiency.

Layer Skip. Layer-skipping techniques can fall into two categories: (1) *training-based* and (2) *training-free* methods. Early works such as LayerSkip (Elhoushi et al., 2024), DeeBERT (Xin et al., 2020), and EE-LLM (Chen et al., 2023) perform early exiting by returning outputs at intermediate layers (Fan et al., 2024). More recent router-based approaches, including Mixture-of-Depth (Raposo et al., 2024), dynamically skips intermediate layers but require training both the model and routers. Later studies (Luo et al., 2025b; He et al., 2024; Luo et al., 2025a; Bae et al., 2025) fine-tune only the routers to lower training costs.

However, multi-stage reasoning models are often tuned on carefully curated reasoning paths generated from larger models, which are rarely accessible. Thus, even though computational costs can be insignificant in training-based ones, training-free approaches are more practical for our setting. Representative examples include SkipDecode (Del Corro et al., 2023), which gradually skips deeper layers during decoding; Unified Skipping (Liu et al., 2024), which periodically skips layers (*e.g.*, 1-st, 4-th, 7-th); and ShortGPT (Men et al., 2024), which uses cosine similarity as a proxy for block importance. Building on this, AdaSkip (He et al., 2025) introduces sub-layer-level importance estimation. These methods typically apply uniform skipping policies, leading to suboptimal efficiency–accuracy trade-offs.

Generation Early Exit. Prior works primarily address redundant explanations of long reasoning models. Zhang et al. (2025) trains probing heads to estimate confidence on intermediate answers and terminate decoding once it is sufficient. Training-free methods, in contrast, typically rely on heuristics. ES-CoT (Mao et al., 2025) stops generation when the same answer repeatedly appears. Logit-based approaches (Yang et al., 2025b; Wang et al., 2025b) monitor confidence or next-token entropy when *</think>* token appears in the reasoning trace. However, these efforts remain limited to reasoning-oriented models that inherently produce verbose outputs, with little attention to mitigating the prolonged generation induced by model compression.

3 Proposed Methods

Problem Statement. Our primary objective is to search the stage-wise layer budget $\mathbb{L}$, *i.e.*, a set of sub-layer indices to skip for each stage, that produces the minimal latency within a given accuracy threshold ϵ. Formally, the objective is given as:

$$\arg\min_{\mathbb{L}} \frac{1}{|\mathbb{D}|} \sum_{d \in \mathbb{D}} \mathcal{T}(\mathcal{M}_{\mathbb{L}}(d)) \tag{1}$$

$$s.t.\ \mathcal{A}(\mathcal{M}_{\mathbb{L}}(d)) \leq \mathcal{A}(\mathcal{M}(d)) - \epsilon \tag{2}$$

where $\mathcal{T}$ and $\mathcal{A}$ denote the inference latency and accuracy of the model; $\mathcal{M}_{\mathbb{L}}$ and $\mathcal{M}$ represent models with layer skipping under the layer budget $\mathbb{L}$ and full layers, respectively; and $\mathbb{D}$ is a test dataset.

3.1 Overview of LiteStage

LiteStage introduces a stage-wise layer skipping strategy that effectively balances the accuracy and latency in multi-stage inference. The details of each component are discussed in the following sections, and an overview is illustrated in Figure 2.

Its mechanism consists of two phases: (1) an *offline configuration* determines the optimal set of sub-layers to skip at each stage. This incorporates the two tasks: which layers to skip and how many layers to skip. We first estimate the layer importance using cosine similarity to pre-define the priority of layers to be skipped (**Step 1**) and then take greedy search that determines the number of layers to skip from the longest reasoning stage to the shortest to effectively reduce the latency within an accuracy threshold (**Step 2**). (2) an *online adjustment* addresses inefficiencies that may occur due to unexpectedly extended generation. We observe that layer skipping induces extra token generation but their confidence level diminishes. Considering that, we jointly apply generation early exit with layer skipping (**Step 3**).

3.2 Step 1: Estimate Layer Importance

Before deciding how many layers to skip in each stage, we first need a layer skipping policy. That is, a criterion for selecting which layers to skip given a target skip count, enabling an efficient search of the accuracy-latency trade-off. We adopt cosine similarity as a proxy for layer importance, as it has been shown to perform effectively in prior studies (Men et al., 2024; He et al., 2025).

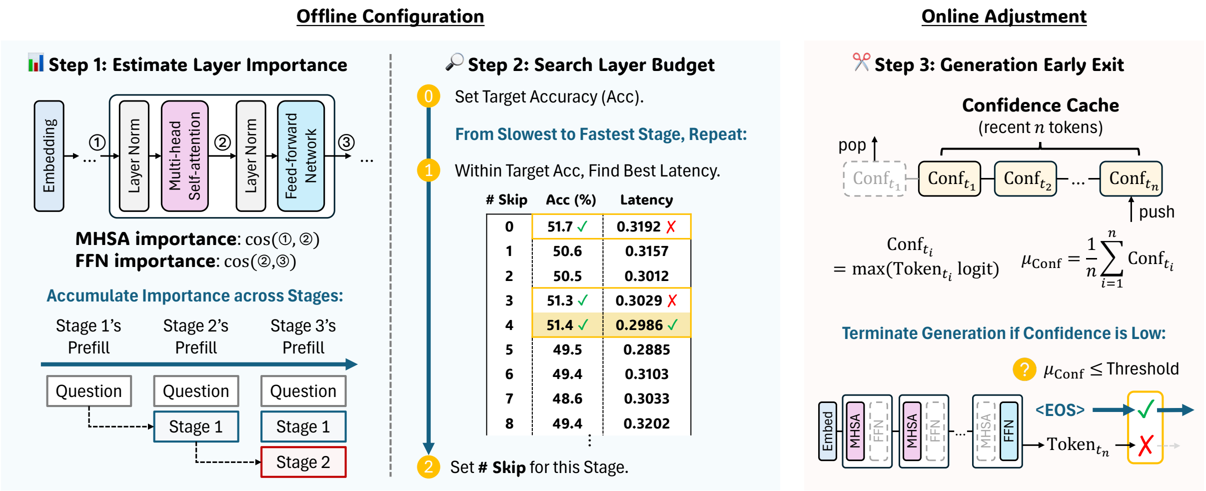
Skipping at Sub-Layer Granularity. We estimate layer importance at the sub-layer level, following the approach of AdaSkip (He et al., 2025). It is important to note that our key contribution lies *not* in designing a new proxy for importance estimation, but in how we explicitly and systematically *balance* the layer budget across reasoning stages. This finer granularity enables us to independently assess the influence of multi-head self-attention (MHSA) and feed-forward network (FFN) sub-layers.

Specifically, the importance of each sub-layer is estimated as follows:

$$I_{\text{MHSA}}^{(j)} = 1 - \frac{1}{N} \sum_{n=0}^{N-1} \cos(\text{MHSA}^{(j)}(x) + x, x) \tag{3}$$

$$I_{\text{FFN}}^{(j)} = 1 - \frac{1}{N} \sum_{n=0}^{N-1} \cos(\text{FFN}^{(j)}(x) + x, x) \tag{4}$$

where $I_{\text{MHSA}}^{(j)}$ and $I_{\text{FFN}}^{(j)}$ denote the importance of the j-th MHSA and FFN layers, respectively. We compute cosine similarity between input and output of each sub-layer, as described in Figure 2 (Step 1), during prefilling and average over N validation samples. Equations (3) and (4) are also averaged across stages, though omitted here for clarity. Higher similarity indicates that input and output representations are more redundant, *i.e.*, less important sub-layer. Given a skip budget, these estimates guide the selection of sub-layers to skip.



# Skip	Acc (%)	Latency
0	51.7 ✓	0.3192 ✗
1	50.6	0.3157
2	50.5	0.3012
3	51.3 ✓	0.3029 ✗
4	51.4 ✓	0.2986 ✓
5	49.5	0.2885
6	49.4	0.3103
7	48.6	0.3033
8	49.4	0.3202

Figure 2: **Overview of LiteStage.** The proposed method consists of an *offline configuration* (Steps 1–2) and an *online adjustment* (Step 3). In the offline phase, (1) layer importance is estimated at the sub-layer level (MHSA and FFN) and accumulated across stages, followed by (2) a search for the optimal layer budget that minimizes latency within a target accuracy. In the online phase, (3) *generation early exit* dynamically terminates decoding when the average confidence of recent tokens falls below a threshold, preventing excessive generation length and ensuring consistent efficiency gains.

This estimation is first performed at Stage 1 using the corresponding prompt, and is subsequently accumulated in Stages 2 and 3 by computing the cosine similarities with their respective prompts. Due to the recursive nature of multi-stage inference, outputs generated at one stage serve as inputs for the next. Consequently, the importance scores incorporate information from both input and generated tokens. This process is conducted offline and only once per dataset. We report I_{MHSA} and I_{FFN} on evaluation datasets in Figure 9 (Appendix), where we observe little variation across datasets.

3.3 Step 2: Search Layer Budget

Our key contribution is in how to search the optimal *layer budget*, *i.e.*, the number of layers to skip. To optimize latency under a fixed accuracy constraint, we first construct an accuracy-latency profile by varying the layer budget in the *longest* reasoning stage using validation data, while keeping all other stages full-layer (like Figure 1(a)-(b)). We then progressively explore layer skipping in the remaining stages, prioritizing acceleration of the longest stage to effectively reduce end-to-end latency.

Skipping from the Longest Stage. Figure 2 (Step 2) illustrates this process using the actual accuracy-latency profile of Stage 2, which is the longest stage, on CSQA with a TinyLlama 1.1B model (see Table 4 (Appendix) for data statistics). Let us consider an accuracy threshold of 1.0% as an example. Given that the baseline accuracy without layer skipping ("# Skip": 0) is 51.7%, the target accuracy becomes 50.7% (*i.e.*, 51.7%-1.0%). The layer budgets that satisfy this target accuracy are highlighted with orange table borders. Among these, skipping four layers ("# Skip": 4) yields the lowest latency of 0.2986, representing the optimal layer budget. This completes a single greedy search iteration. For the next longest stages, we repeat this profiling process in the same manner, but with the previously optimized stages already under their selected layer budgets (*e.g.*, applying "# Skip": 4 for Stage 2). We maintain the same target accuracy of 50.7%, allowing most of the accuracy degradation to occur in the first search, which is desirable since the longest stage will be effectively accelerated.

Why Accuracy-Latency Profiling Matters. Despite its simplicity, search-based layer budget allocation provides several key advantages, that are not captured when relying solely on cosine similarity.

(1) *avoiding sub-optimal latency*: we often observe that accuracy and latency do not change monotonically with the number of skipped layers. As illustrated in Figure 2, beyond the "# Skip" of 5, the latency gain diminishes, and further skipping can even increase end-to-end inference time. This counterintuitive behavior arises because, although per-token latency is reduced, aggressive approximation often increases the total number of generated tokens, resulting in higher overall latency. Since our

Table 2: **Offline Search Cost.** Average evaluation latency (minutes) per skipping configuration. The total search time scales with the number of layers (*e.g.*, 22 for TinyLlama-1.1B) and reasoning stages (*e.g.*, 3 in our setup). Results are measured on a single A6000 GPU.

Model \ Benchmark	**OBQA**	**CSQA**	**StrategyQA**
TinyLlama-1.1B	2.51	5.57	0.78
Qwen2.5-0.5B	1.89	4.23	0.68

search procedure jointly optimizes accuracy and latency, such configurations are naturally pruned.

(2) *interaction between stages*: the profile ensures that the interaction between reasoning stages under different layer budgets are accurately reflected in the search. For example, if applying layer skipping at one stage and then searching another stage's layer budget, this process differs from searching this stage's layer budget with all others full-layer.

(3) *identifying most sensitive stage*: as shown in Figure 1(b), uniformly skipping layers across all stages yields an accuracy profile nearly identical to skipping only in Stage 3. This indicates that the final accuracy is bottlenecked by the most sensitive stage. The profile explicitly reveals such sensitivity, enabling non-uniform allocation that protects critical stages (*e.g.*, Stage 3), allowing more aggressive skipping in robust stages (*e.g.*, Stages 1 and 2).

Offline Search Cost. Table 2 reports the overhead of the offline search, measured as the evaluation latency per skipping configuration (*e.g.*, "# Skip": n). Searching for the optimal configuration under a target accuracy requires exploring combinations of decoding layers and reasoning stages (*e.g.*, "# Skip"$\in[0, n]$). In practice, this one-time offline search is conducted on a single GPU and typically completes within a few hours: approximately 2.8/6.1/0.9 hours on OBQA/CSQA/StrategyQA for TinyLlama-1.1B, and 1.5/3.4/0.5 hours for Qwen2.5-0.5B. Once identified, the optimal configuration is reused for all subsequent inference, incurring no additional search overhead.

3.4 Step 3: Generation Early Exit

While Step 2 determines a balanced layer budget, it leaves the unexpectedly prolonged outputs as is. To further extend the speedup, we investigate how to reduce these redundant output tokens and thereby recover the original latency gains achieved through layer skipping. Our hypothesis is that the extra tokens induced by layer skipping contribute little to the final reasoning outcome.

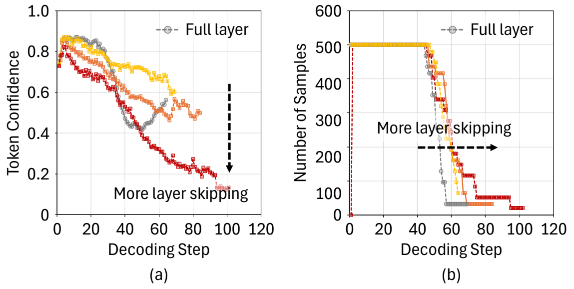


Figure 3: **Confidence of Output Tokens.** (a) Average token-level confidence across decoding steps under different layer-skipping levels. (b) Number of remaining samples during decoding (not yet exited), where more aggressive layer skipping leads to earlier exits due to confidence decay. Results are shown for skipping of 10 (yellow), 15 (orange), and 20 (red) sub-layers in Stage 1 on the OBQA validation set using TinyLlama-1.1B.

Extra Tokens may not be Useful. Figure 3 illustrates how token confidence evolves over decoding steps under four different skip configurations. Here, token confidence is defined as the maximum logit value of each generated token. The confidence trajectories differ between models with and without layer skipping, *e.g.*, the layer-skip models exhibit a consistent decrease in confidence, whereas the full-layer model partially recovers confidence at later steps. However, a common property emerges: high-confidence predictions (above 0.5) occur primarily in the early decoding steps. This pattern becomes more evident as more layers are skipped; for instance, the red line (20-layer skip) shows a consistently lower confidence curve, extended generation length, and a lower minimum confidence value than the yellow line (10-layer skip). Accordingly, we apply confidence-based generation early exit, assuming that terminating these unconfident extra output tokens may not hurt the accuracy much.

Confidence-based Termination. Our approach is straightforward: as shown in Figure 2 (Step 3), if the confidence of an output token falls below a threshold, we replace it with an end-of-sequence (EOS) token, thereby stopping further generation. However, the confidence values can fluctuate significantly across decoding steps. Therefore, relying on a single token's confidence may trigger premature termination. To stabilize the decision, we maintain a *confidence cache* that stores the confidence values of the most recent n tokens. From the n-th step, we compute the mean confidence μ_{Conf} across the cache and compare it with a threshold. We heuristically set $n{=}5$ and the confidence threshold to 0.5.

We also incorporate generation early exit into the search process (Step 2), ensuring that the resulting accuracy-latency profile reflects real evaluation conditions. In addition, without generation early exit, the effective search space can become severely restricted. For example, when generation length is highly sensitive to layer skipping, latency improvements may disappear beyond few "# Skip" in the accuracy-latency profile. In such cases, generation early exit mitigates excessive token generation and thereby expands the feasible search space, yielding a larger set of valid layer-skipping configurations.

4 Experimental Results

Datasets and Implementation Details. We adopt the three-stage reasoning flow from TinyThinker (Piao and Park, 2024) on three question-answering benchmarks: OpenBookQA (OBQA) (Mihaylov et al., 2018), CommonSenseQA (CSQA) (Talmor et al., 2018), and StrategyQA (Geva et al., 2021). Models are first fine-tuned on TinyThinker's augmented training data before applying layer skipping. During validation and testing, reasoning paths are excluded.

We primarily evaluate our method using TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024) and Qwen2.5-0.5B (Yang et al., 2024). Our training and evaluation procedures follow TinyThinker (Piao and Park, 2024), with adjusted hyperparameters: training for 10 epochs with a batch size of 16 on OBQA and CSQA, and 24 on StrategyQA, using an initial learning rate of 5×10^{-5}. Evaluation follows TinyThinker's self-consistency (*i.e.*, majority voting) protocol with 10 iterations. More details are available in Appendix B.2.

Baselines. We consider recent training-free layer-skipping methods, SkipDecode (Del Corro et al., 2023), UnifiedSkip (Liu et al., 2024), and AdaSkip (He et al., 2025), as our baselines. Our main baseline is AdaSkip, as our layer-importance estimation also adopts their sub-layer-wise cosine similarity. We apply layer skipping only during decoding stages, and our implementation of the baselines also follows the AdaSkip's code[1].

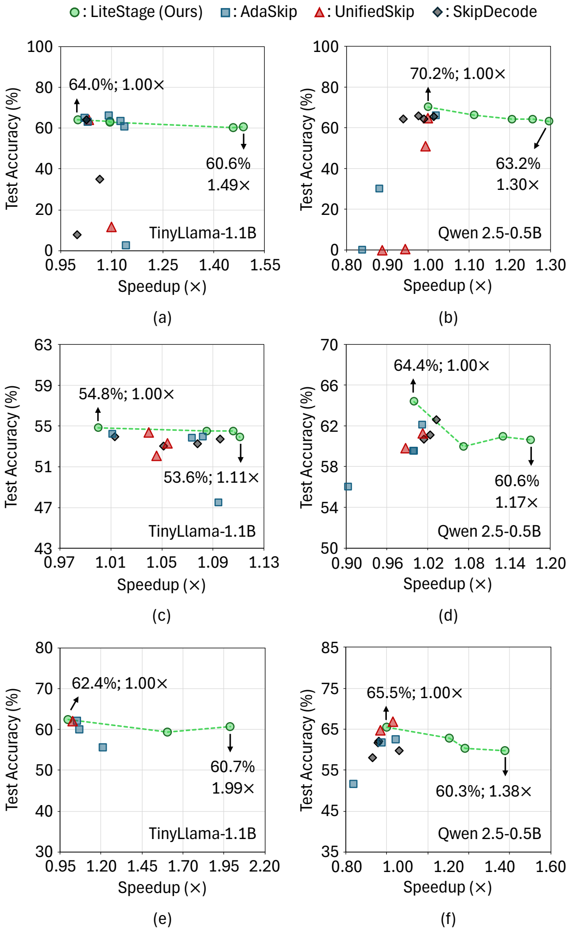


Figure 4: **Comparison with Baselines.** Accuracy–speedup trade-offs on the OBQA (row-1), CSQA (row-2), and StrategyQA (row-3) datasets, respectively. Results are experimented using TinyLlama-1.1B (col-1) and Qwen2.5-0.5B (col-2) models. The performance of the full-layer model is marked in the upper-left region of each plot (*e.g.*, 64.0% and 70.2% for OBQA), and the speedup is normalized by the full-layer latency.

4.1 Comparison with Baselines

Figure 4 provides a comprehensive comparison between our proposed LiteStage and three baseline methods across the OBQA, CSQA, and StrategyQA datasets. The baseline methods are evaluated by progressively increasing the number of skipped layers until its speedup saturates. Our approach consistently outperforms the baselines, particularly in high-speedup ranges, clearly extending their performance boundaries. For example, in the OBQA results (Figure 4(a)), the primary baseline AdaSkip maintains accuracy comparable to ours up to a speedup of 1.10×. Beyond this point, however, its performance collapses to nearly 0% accuracy, whereas our method remains robust, achieving a 1.49× speedup with 60.6% accuracy.

We highlight two key observations from these results: (1) how LiteStage mitigates severe accuracy degradation (0%→60%) via *non-uniform layer budget*, and (2) extends the latency limit

---

[1] https://github.com/ASISys/AdaSkip

Table 3: **Non-uniform Layer Budget.** The number of skipped layers allocated to each stage by LiteStage across the three benchmarks. Lower rows represent more aggressive layer skipping with lower latency. Each row corresponds to a data point in Figure 4.

	TinyLlama-1.1B			Qwen2.5-0.5B		
	Stage 1	Stage 2	Stage 3	Stage 1	Stage 2	Stage 3
OBQA	5	3	4	6	0	1
	2	4	5	15	0	1
	21	4	4	15	1	1
	19	6	4	11	2	1
CSQA	0	3	0	0	2	0
	1	3	5	1	2	1
	1	3	4	9	1	3
StrQA	12	10	0	14	0	1
	0	15	0	3	5	0
	-	-	-	3	6	1

(1.10 × →1.49×) through *generation early exit*.

4.2 Benefits of Non-uniform Layer Budget

Protection from Accuracy Collapse. Table 3 presents the number of layers allocated to each stage by LiteStage. Notably, LiteStage consistently avoids skipping more than five (TinyLlama) and three (Qwen) layers in Stage 3, highlighting its ability to adaptively constrain aggressive compression in sensitive stages while intensively accelerating more robust ones. This is because Stage 3 is substantially more sensitive to layer skipping than other stages, thus upper-bounding the overall accuracy (see Figure 12-17 (Appendix)). By assigning fewer skipped layers to such sensitive stages, our method maintains accuracy, highlighting the effectiveness of our non-uniform layer skipping.

Layer Budget Distribution. Beyond simply protecting accuracy-sensitive stages, our approach adapts the layer budget distribution across datasets. As shown in Figures 1(b)-(c), Stage 1 is robust to accuracy degradation, whereas Stage 2 dominates the overall latency. Accordingly, when accuracy needs to be maintained, LiteStage applies more aggressive layer skipping to Stage 1. For example, on OBQA (see Table 3), most variations occur in the Stage 1 budget, increasing from 5→21 (TinyLlama) and from 6→15 (Qwen).

As stronger speedup is desired, LiteStage increase the layer budget of Stage 2 while reducing that of Stage 1. For instance, in the most aggressive configuration on OBQA (row 3 in Table 3), the budgets change from $21, 4 \rightarrow 19, 6$ (TinyLlama) and from $15, 1 \rightarrow 11, 2$ (Qwen) for Stages 1 and 2, respectively. As a result, LiteStage first minimizes the latency of Stage 1 and subsequently optimizes that of Stage 2 as larger latency reductions are required (see Figure 5). A similar trend is observed on the StrategyQA dataset as well. For CSQA, Stage 2 is even more sensitive to layer skipping, hence larger speedup is achieved by further accelerating Stage 1.

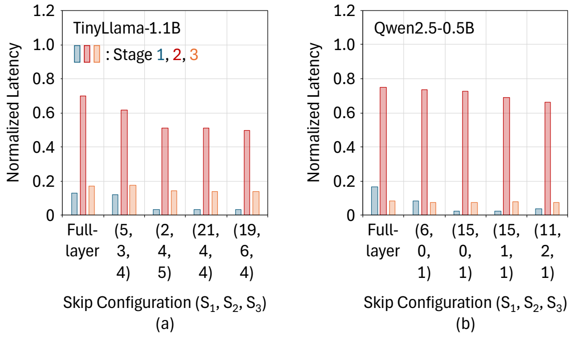
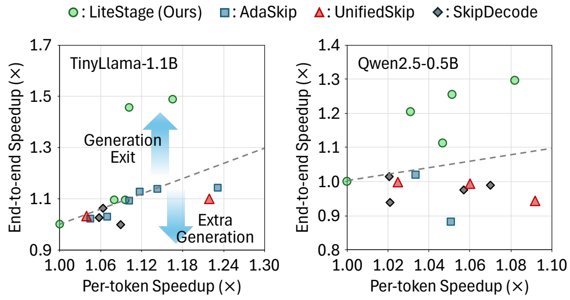


Figure 5: **Stage-wise Latency.** (a)-(b) show the normalized latency of each stages in TinyLlama-1.1B and Qwen2.5-0.5B, respectively, on OBQA. In the full-layer configuration, the sum of stage-wise latencies is normalized to 1.0. S_1, S_2, S_3 on the x-axis denotes the number of skipped layers at Stages 1, 2, and 3.

4.3 Benefits of Generation Early Exit

Per-token vs End-to-End Speedup. Extended generation hinders end-to-end speedup, although per-token latency is reduced by layer skipping. Figure 6 illustrates this by comparing per-token and end-to-end speedup. We expect enhanced per-token speedup to translate into end-to-end speedup; this ideal behavior is indicated by the dotted line ($y{=}x$) in the figure. However, for the baselines, while per-token speedup continues to increase, the end-to-end speedup saturates or decreases. This degradation is particularly pronounced under aggressive layer skipping, where per-token speedup is largest. In contrast, by incorporating generation early exit, LiteStage effectively enhances end-to-end speedup that often exceeds per-token speedup, reaping the benefits of layer skipping.

With and Without Generation Early Exit. Figure 7 presents ablation studies analyzing the effect of applying generation early exit along with layer skipping. When only a few layers are skipped, *e.g.*, 12 (TinyLlama) and 2 (Qwen) sub-layers, the differences in accuracy, latency, and decoding steps between with and without generation early exit are marginal. This suggests that, under mild skipping, the models still generate tokens with high confidence, resulting in decoding lengths compa-



(a) (b)

Figure 6: **Per-token and End-to-End Speedup.** (a)-(b) show the per-token and end-to-end speedup of TinyLlama-1.1B and Qwen2.5-0.5B, respectively, on the OBQA dataset. The full-layer baseline is normalized to a per-token and end-to-end speedup of 1.0. The gray dotted line denotes the $y{=}x$ reference.

rable to those of the full-layer baseline. Consequently, the observed speedup in this regime primarily stems from the non-uniform layer budget rather than reduced generation length. As more layers are skipped, however, the gap in decoding steps increases consistently (Figures 7(e)-(f)), leading to shorter generated sequences than those of the full-layer baseline and yielding proportional latency reductions (Figures 7(c)-(d)).

Accuracy Improvements. Interestingly, in Figures 7(a)-(b), the models with generation early exit achieve even higher accuracy than those without it. This indicates that redundantly generated (*i.e.*, low-confidence) tokens under aggressive layer skipping not only fail to contribute meaningfully but can even make it challenging to produce correct final outputs. These results further highlight that the effectiveness of LiteStage stems from jointly applying layer skipping and generation early exit.

4.4 Diagnostic Study on Deep Reasoning

While LiteStage is designed for short multi-stage reasoning, it is natural to ask whether our strategies can also benefit *deep reasoning* tasks (*e.g.*, mathematics, coding). To this end, we conduct a diagnostic study on deep reasoning benchmarks, following the similar evaluation protocol used for short multi-stage tasks. In this study, we focus on examining how basic layer skipping behaves in deep reasoning settings and how its performance changes when combined with generation early exit. Please refer the details in Appendix D.

Our results show that combining layer skipping with generation early exit consistently achieves higher speedups. However, we observe substantial accuracy degradation across *all* evaluated configurations, even under moderate levels of approximation. This behavior indicates that the efficiency-accuracy trade-off in deep reasoning differs fundamentally from that in short multi-stage reasoning.

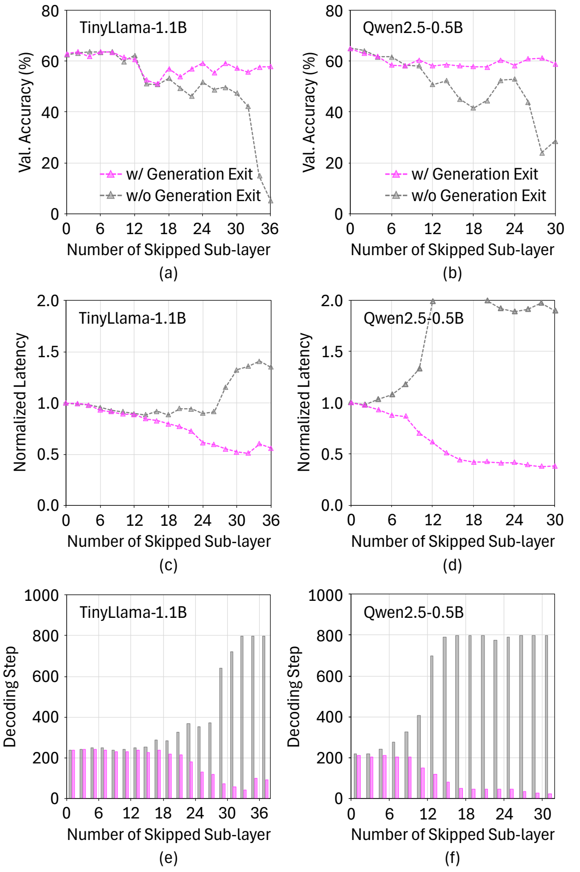


Figure 7: **Ablation Study.** (a)–(b) validation accuracy, (c)–(d) normalized latency, and (e)–(f) decoding steps as a function of the number of skipped sub-layers for TinyLlama-1.1B and Qwen2.5-0.5B on the OBQA dataset. Results apply layer skipping only at Stage 2, with or without generation early exit.

5 Conclusion

We introduced LiteStage, a latency-aware layer-skipping framework for efficient multi-stage reasoning in small LLMs. By jointly optimizing stage-wise layer budgets and applying confidence-based generation early exit, LiteStage effectively balances accuracy and latency. Experiments on OBQA, CSQA, and StrategyQA demonstrate that LiteStage surpasses prior training-free methods with a large speedup margin. Our results highlight the importance of stage-aware optimization and adaptive decoding in realizing truly efficient multi-stage reasoning.

6 Limitations

LiteStage involves an offline profiling step to estimate accuracy-latency characteristics and select stage-wise layer budgets. While this profiling is performed once per model and setting and is amortized over repeated inference, it may introduce additional overhead compared to approaches that rely solely on fixed heuristics. In addition, generation early exit in our implementation is based on a simple confidence criterion, and more adaptive or task-specific exit strategies may further improve robustness. Finally, our analysis primarily focuses on short multi-stage reasoning, and the diagnostic results on deep reasoning suggest that the behavior under long-horizon generation can differ, indicating that extending LiteStage to such settings may require additional investigation.

References

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and 1 others. 2025. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

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

Chengwei Dai, Kun Li, Wei Zhou, and Songlin Hu. 2024. Improve student's reasoning generalizability through cascading decomposed cots distillation. *arXiv preprint arXiv:2405.19842*.

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

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, and 1 others. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics (TACL)*.

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, and Dong Yu. 2024. Router-tuning: A simple and effective approach for enabling dynamic-depth in transformers. *arXiv preprint arXiv:2410.13184*.

Zhuomin He, Yizhen Yao, Pengfei Zuo, Bin Gao, Qinya Li, Zhenzhe Zheng, and Fan Wu. 2025. Adaskip: Adaptive sublayer skipping for accelerating long-context llm inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24050–24058.

WANG Hongru, Deng Cai, Wanjun Zhong, Shijue Huang, Jeff Z Pan, Zeming Liu, and Kam-Fai Wong. 2025. Self-reasoning language models: Unfold hidden reasoning chains with few reasoning catalyst. In *Workshop on Reasoning and Planning for Large Language Models*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. 2024. The impact of reasoning step length on large language models. *arXiv preprint arXiv:2401.04925*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*.

Xiang Li, Shizhu He, Fangyu Lei, JunYang JunYang, Tianhuang Su, Kang Liu, and Jun Zhao. 2024. Teaching small language models to reason for knowledge-intensive multi-hop question answering. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7804–7816.

Yanyang Li, Michael R Lyu, and Liwei Wang. 2025. Learning to reason from feedback at test-time. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5241–5253.

Yijin Liu, Fandong Meng, and Jie Zhou. 2024. Accelerating inference in large language models with a unified layer skipping strategy. *arXiv preprint arXiv:2404.06954*.

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

Xuan Luo, Weizhi Wang, and Xifeng Yan. 2025b. Diffskip: Differential layer skipping in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7221–7231.

Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. 2025. Early stopping chain-of-thoughts in large language models. *arXiv preprint arXiv:2509.14004*.

Mathematical Association of America. 2025. Aime 2025: Competition problems for mathematical reasoning evaluation. Used as a benchmark for evaluating mathematical reasoning in large language models.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Shengmin Piao and Sanghyun Park. 2024. Tinythinker: Distilling reasoning through coarse-to-fine knowledge internalization with self-reflection. *arXiv preprint arXiv:2412.08024*.

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

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Siyuan Wang, Enda Zhao, and Xiang Ren. 2025a. Stepwise informativeness search for improving llm reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25291–25309.

Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. 2025b. Entropy after $\langle$/Think$\rangle$ for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*.

Guowei Xu, Peng Jin, Ziang Wu, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. 2025. Llava-cot: Let vision language models reason step-by-step. *Preprint*, arXiv:2411.10440.

Shangzi Xue, Zhenya Huang, Jiayu Liu, Xin Lin, Yuting Ning, Binbin Jin, Xin Li, and Qi Liu. 2024. Decompose, analyze and rethink: Solving intricate problems with human-like reasoning cycle. *Advances in Neural Information Processing Systems*, 37:357–385.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115v2*.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Zheng Lin, Li Cao, and Weiping Wang. 2025b. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*.

Wen Yang, Minpeng Liao, and Kai Fan. 2025c. Markov chain of thought for efficient mathematical reasoning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7132–7157.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Ed Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024. Self-discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems*, 37:126032–126058.

A Overview

This appendix provides additional experimental details and analyses that supplement our manuscript "LiteStage: Latency-aware Layer Skipping for Multi-stage Reasoning". We first report statistics on stage-wise decoding length to characterize the computational properties of multi-stage reasoning (Appendix B.1), followed by implementation details and training dynamics for decoder-only models adapted from the TinyThinker framework (Appendix B.2-B.3). We then present analyses of sub-layer importance estimation based on cosine similarity, along with extensive ablation studies evaluating the effect of layer skipping across individual reasoning stages, models, and datasets (Appendix C). Finally, we include additional analyses on deep reasoning tasks to examine the behavior of LiteStage under long-horizon generation settings (Appendix D).

Code is provided in the supplementary material.

B Data and Implementation Details

B.1 Data Statistics

Table 4 summarizes the average number of decoding steps per reasoning stage in LiteStage. The statistics are estimated using full-layer models, allowing us to analyze the intrinsic complexity of each reasoning stage in its original form, without the influence of layer skipping or early exit. Across all three datasets of OBQA, CSQA, and StrategyQA, the *Analyze* stage consistently exhibits the largest number of decoding steps, indicating that it dominates the overall generation length and computational cost. In contrast, the *Summarize* stage is the shortest, as it typically follows a fixed answer template (*e.g.*, "So, the answer is (·)"), requiring minimal generation. The *Recall* stage lies between these extremes, reflecting moderate and relatively stable generation length across datasets. This clear imbalance in stage-wise decoding length motivates our stage-aware optimization strategy, which prioritizes accelerating the longest stages to achieve effective end-to-end speedup.

Table 4: **Decoding Step Statistics.** The number of decoding steps for each stage (Stage 1: Recall, Stage 2: Analyze, and Stage 3: Summarize) is estimated and averaged over the test set of the three datasets (OBQA, CSQA, and StrategyQA) for our models, TinyLlama-1.1B and Qwen2.5-0.5B.

TinyLlama-1.1B	**OBQA**	**CSQA**	**StrategyQA**
Recall	55.5	46.4	68.6
Analyze	236.9	250.6	155.8
Summarize	7.0	7.0	7.0
Qwen2.5-0.5B	**OBQA**	**CSQA**	**StrategyQA**
Recall	49.1	36.9	58.7
Analyze	206.4	220.1	159.8
Summarize	7.0	7.0	7.0

B.2 Training Setups

Our training and evaluation pipelines are largely based on the TinyThinker codebase[2]. Since the base models such as TinyLlama-1.1B and Qwen2.5-0.5B are not instruction-tuned for multi-stage reasoning, we fine-tune them to follow structured reasoning stages (*e.g.*, recall, analysis, and summarization). The training data consist of questions from the official OBQA, CSQA, and StrategyQA datasets, along with reasoning paths generated by larger models such as GPT, following the TinyThinker protocol. Training on these reasoning paths can therefore be viewed as a form of knowledge distillation from GPT to smaller models such as TinyLlama-1.1B and Qwen2.5-0.5B. Further details of the training procedure can be found in the original TinyThinker paper (Piao and Park, 2024).

However, TinyThinker's original experiments are based on T5 models. Accordingly, we adopt a different set of hyperparameters to better support fine-tuning of decoder-only architectures (reported in the main paper). We perform full fine-tuning, updating all model parameters using the complete OBQA, CSQA, and StrategyQA datasets. Training is conducted with the AdamW optimizer (as in the default `SFTTrainer`), using a weight decay of 0.0, β_1=0.9, β_2=0.999, and ϵ=10^{-8}. All experiments are conducted on a single NVIDIA A6000 GPU (48GB).

B.3 Training Dynamics

Recent progress in reasoning tasks has been largely driven by decoder-only architectures such as LLaMA and Qwen. Motivated by this trend, we extend the TinyThinker framework, that is originally developed for encoder–decoder T5 models, to the LLaMA and Qwen families. Specifically, we fine-tune TinyLlama-1.1B and Qwen2.5-0.5B using TinyThinker's three-stage reasoning supervision on the OBQA, CSQA, and StrategyQA datasets.

The largest TinyThinker model (T5-Large,

---

[2] https://github.com/shengminp/TinyThinker

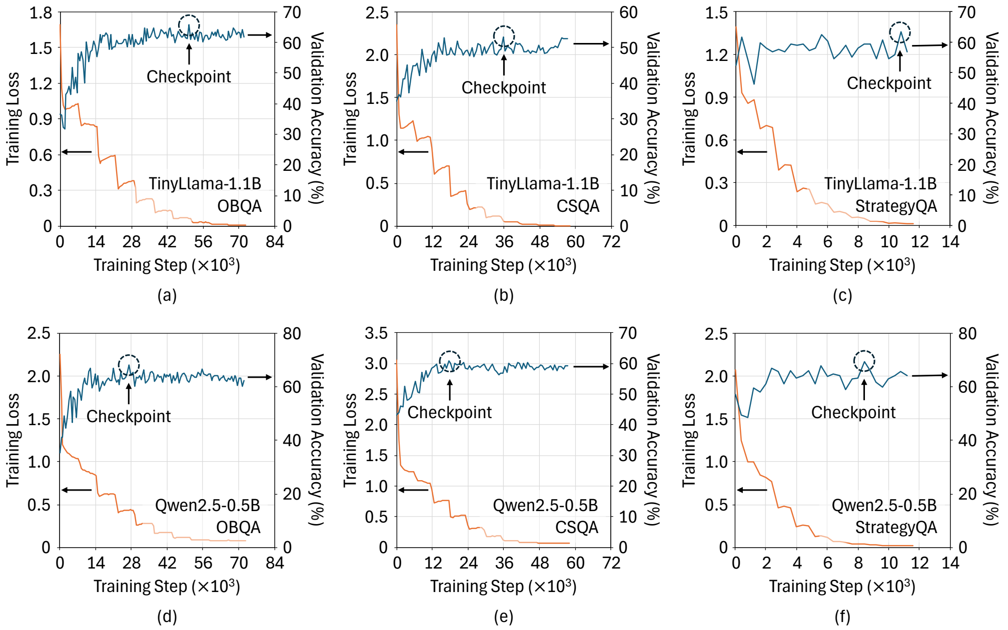


Figure 8: **Training Dynamics.** Training loss and validation accuracy over training steps for TinyLlama-1.1B and Qwen2.5-0.5B on OBQA, CSQA, and StrategyQA. The top row shows results for TinyLlama-1.1B, and the bottom row shows results for Qwen2.5-0.5B. In each plot, training loss is shown on the left y-axis and validation accuracy on the right y-axis. Circles denote the selected checkpoints used for evaluation.

770M parameters) reports test accuracies of 65.4% (CSQA), 68.8% (OBQA), and 69.0% (StrategyQA). In comparison, TinyLlama-1.1B achieves 54.8%, 64.0%, and 62.4% on the same datasets, while Qwen2.5-0.5B attains 64.4%, 70.2% , and 65.5%. Notably, despite having fewer parameters, Qwen2.5-0.5B consistently outperforms TinyLlama-1.1B, reflecting the stronger instruction-following and reasoning priors of the Qwen family.

Figure 8 illustrates the training dynamics of both models, including training loss and validation accuracy. Across all datasets, the training loss decreases smoothly, while validation accuracy saturates early and remains stable thereafter. This behavior indicates effective convergence under structured multi-stage reasoning supervision. We also observe that validation accuracy is consistently lower than test accuracy, since test results are computed using self-consistency with ten sampled generations (majority voting), whereas validation accuracy reflects single-pass decoding.

For TinyLlama-1.1B, we attribute part of the remaining performance gap relative to T5-Large to differences in training configuration. TinyThinker's original experiments utilize four A100 GPUs, enabling substantially larger batch sizes, whereas our experiments are conducted on a single A6000 GPU. This difference likely affects optimization stability and final performance, particularly for models trained to follow multi-stage reasoning patterns.

C Additional Experimental Results

C.1 Example of Generation Early Exit

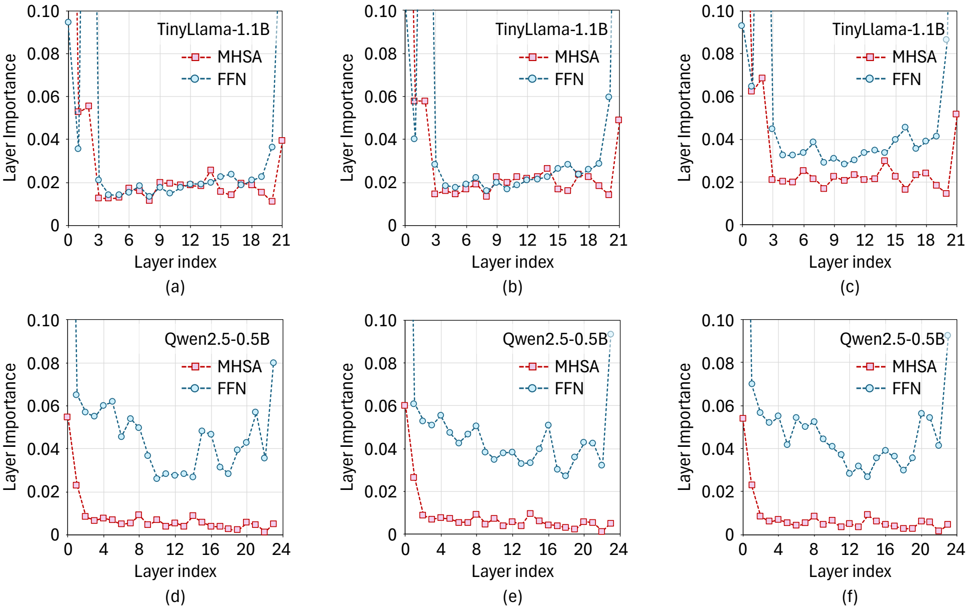
Figure 11 presents a CSQA test example comparing three-stage reasoning outputs produced by models with and without generation early exit. Due to the size of the visualization, the figure is placed on later pages. For reference, the full-layer baseline output is shown in Figure 11(a), while Figures 11(b) and (c) correspond to layer skipping without and with generation early exit, respectively.

Generation early exit terminates decoding once token-level confidence falls below a predefined threshold. As a result, the generated tokens remain identical to the baseline up to the exit point, and only the low-confidence suffix is truncated. In the figure, the truncated segments are highlighted in blue for clarity. For example, the full-layer reasoning phrase

```
For option C, photo copy refers to a
visual representation of a person, which
is unrelated to the biological process
of producing offspring,
```

is shortened to

```
For option C, photo copy refers to a
visual representation of a
```



(a) (b) (c)

(d) (e) (f)

Figure 9: **Layer Importance.** Sub-layer-wise importance estimates for TinyLlama-1.1B (a)–(c) and Qwen2.5-0.5B (d)–(f) across OBQA, CSQA, and StrategyQA. Importance is computed using cosine similarity between sub-layer inputs and outputs, separately for multi-head self-attention (MHSA) and feed-forward network (FFN) layers.

This behavior indicates that the model maintains high confidence up to the phrase "visual representation", while the subsequent explanatory tokens exhibit lower confidence and contribute marginally to decision making. Importantly, despite the truncated reasoning, both models arrive at the same final conclusion in Stage 3, correctly rejecting option C and selecting option D as the answer. This example demonstrates that generation early exit effectively removes redundant low-confidence reasoning tokens without altering the final prediction, thereby reducing decoding length while preserving reasoning correctness.

C.2 Layer Importance Estimation

Figure 9 presents the sub-layer-wise importance estimates obtained in Step 1 of our pipeline. It reveals several consistent patterns in sub-layer importance that directly inform our skipping strategy. Across all three datasets, the relative ordering of layer importance remains largely stable, suggesting that the contribution of individual layers is not highly task-specific. However, we observe dataset-dependent shifts in the balance between multi-head self-attention (MHSA) and feed-forward network (FFN) sub-layers.

In particular, for TinyLlama-1.1B, FFN sub-layers on StrategyQA exhibit comparatively higher importance than on OBQA and CSQA, leading the resulting skipping policy to prioritize MHSA layers while preserving FFN computation. A more pronounced distinction emerges across model families. In Qwen2.5-0.5B, MHSA sub-layers consistently demonstrate substantially higher importance than FFN sub-layers across all datasets. Consequently, layer skipping in Qwen models is predominantly applied to MHSA, whereas FFN layers are largely retained.

Despite these differences in absolute importance levels, the overall layer-wise trends remain consistent across datasets, indicating that sub-layer importance is primarily governed by model architecture rather than dataset-specific characteristics. These observations motivate our use of sub-layer–level importance estimation and justify decoupling the skipping decisions for MHSA and FFN, enabling LiteStage to adapt its computation budget to both architectural and dataset-level variations.

C.3 Ablation Studies

To analyze stage-wise accuracy and latency behavior under layer skipping, we conduct extensive ablation studies on two models, TinyLlama-1.1B and Qwen2.5-0.5B, across three benchmarks: OBQA, CSQA, and StrategyQA. In these experiments, we vary the number of skipped sub-layers while applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3), and measure validation accuracy along with normalized end-to-end latency, where 1.0 corresponds to the full-layer baseline. We further ablate each configuration with and without generation early exit to isolate its effect when combined with layer skipping. Due to space constraints, all ablation figures are provided in the later pages: Figures 12-14 report results for TinyLlama-1.1B, and Figures 15-17 report results for Qwen2.5-0.5B.

From these ablations, we highlight two key observations: (1) the importance of explicitly profiling the accuracy–latency trade-off, and (2) the consistent benefits of generation early exit across models, datasets, and reasoning stages.

(1) *importance of accuracy-latency profiling:* Across all datasets, we observe a broadly consistent sensitivity pattern across reasoning stages: Stage 1 is generally the most robust to layer skipping, followed by Stage 2, while Stage 3 is the most sensitive. However, a critical finding is that the degree of sensitivity varies substantially across both models and datasets. For example, when skipping only Stage 2 on OBQA, Qwen2.5-0.5B exhibits markedly higher accuracy robustness than TinyLlama-1.1B, as shown in Figures 12(b) and 15(b). Such differences are non-trivial and cannot be reliably inferred from static proxy metrics such as cosine similarity alone; instead, they only become apparent through direct profiling of the accuracy-latency behavior.

Moreover, the ablations reveal that accuracy and latency do not vary monotonically with the number of skipped sub-layers. In some cases, skipping more layers unexpectedly yields higher accuracy, or skipping fewer layers results in higher speedup. For instance, in Figure 12(a), skipping 18 sub-layers achieves higher validation accuracy than skipping 16 sub-layers, while in Figure 12(d), skipping 16 sub-layers results in higher normalized latency than skipping 14 sub-layers, even when generation early exit is applied. These irregularities highlight that naïvely increasing or decreasing the skip budget can lead to suboptimal or even counterproductive outcomes.

LiteStage's "Step 2: Search Layer Budget" directly addresses this challenge by selecting the optimal number of skipped sub-layers based on the empirically observed accuracy-latency profile, rather than relying on monotonic assumptions or heuristic thresholds. This search-based strategy allows LiteStage to avoid suboptimal configurations that would otherwise degrade end-to-end performance.

(2) *benefits of generation early exit:* A second consistent observation across all ablations is the crucial role of generation early exit in realizing practical latency gains. Without early exit, aggressive layer skipping often increases generation length, which can negate or even reverse latency improvements. This phenomenon is clearly visible across all datasets and both models, particularly when skipping Stage 2, where generation length is typically the longest.

By incorporating generation early exit, latency growth is effectively suppressed, and normalized latency remains stable or decreases even under aggressive skipping. This trend is consistently observed across OBQA, CSQA, and StrategyQA for both TinyLlama-1.1B and Qwen2.5-0.5B. While Figure 7 (main paper) already demonstrates this effect on a representative setting, the supplementary ablations confirm that the benefit of generation early exit generalizes across models, datasets, and reasoning stages.

Taken together, these results underscore that layer skipping alone is insufficient for achieving reliable end-to-end acceleration. Instead, effective efficiency gains require joint consideration of stage-wise accuracy-latency profiling and generation early exit, motivating their integrated use in LiteStage.

D Diagnostic Study on Deep Reasoning

Setups. To examine the applicability of LiteStage to deep reasoning tasks, we evaluate layer-skipping behavior using Qwen3.0-1.7B in reasoning mode (Yang et al., 2025a). We use the off-the-shelf model without additional fine-tuning. Evaluation benchmarks include AIME 2025 (AIME 25; mathematics) (Mathematical Association of America, 2025), GPQA-Diamond (GPQA-D; question answering) (Rein et al., 2024), and LiveCodeBench release v5 (LCB-v5; coding) (Jain et al., 2024) datasets. Our evaluation largely follows the pipeline provided by QwQ (Team, 2025)[3].

Since these benchmarks do not provide validation splits, we first perform "Step 1: Estimate Layer Importance" using the OBQA, CSQA, and StrategyQA datasets, and then compute the average cosine similarities across decoding layers. We treat deep reasoning inference as a single-stage process, which eliminates the need for "Step 2: Search Layer Budget" for stage-wise optimization; accordingly, we apply uniform layer skipping across the entire decoding process. For "Step 3: Generation Early

---

[3] https://github.com/QwenLM/QwQ

Table 5: **Deep Reasoning Performance.** Test accuracy, end-to-end speedup, and the number of decoding steps are estimated on the three benchmarks, AIME 2025 (AIME 25), GPQA-Diamond (GPQA-D), and LiveCodeBench v5 (LCB-v5). The speedup is normalized by the full-layer baseline, and the decoding steps are averaged over test samples. LS, PF, and GE denote Layer Skipping, Periodic Full-decoding, and Geneartion Early Exit, respectively. Results are experimented using Qwen3-1.7B.

Method	**# Skip**	**Accuracy (Pass@1)**			**End-to-End Speedup (×)**			**Decoding Steps**		
		AIME 25	GPQA-D	LCB-v5	AIME 25	GPQA-D	LCB-v5	AIME 25	GPQA-D	LCB-v5
Full-layer	0	30.0	46.0	34.8	1.00	1.00	1.00	18530	9374	16175
LS	1	30.0	32.8	34.1	0.91	0.75	0.88	22230	12511	19989
	2	13.3	30.8	11.8	1.05	1.48	0.98	20584	7195	18655
	3	0.0	20.2	0.0	0.86	0.92	0.68	25681	11706	27273
LS+PF	1	26.7	38.9	34.8	0.91	0.89	0.93	20221	10523	18108
	2	20.0	35.9	35.5	1.06	1.10	0.96	18992	8766	17224
	3	20.0	32.8	25.5	1.11	1.15	1.04	18149	8714	16780
LS+PF+GE	1	26.7	36.4	31.2	1.06	1.08	1.35	18681	8808	12568
	2	20.0	33.3	26.2	1.40	1.84	1.36	14699	5296	12643
	3	16.7	31.8	17.6	1.28	1.36	1.46	15768	7081	11863

Exit", we observe that the Qwen3-1.7B generally exhibits higher confidence than the models used in short multi-stage reasoning. We therefore increase the confidence threshold to 0.8. Other generation hyperparameters are set to a temperature of 0.6, top-p of 0.95, top-k of 20, and a maximum of 32,768 new tokens.

Layer Skipping. We apply sub-layer-level skipping (*e.g.*, MHSA or/and FFN), except for the first and last four decoding layers. Layer skipping is applied only during decoding, while the prefill stage is always executed at full depth. We observe that applying layer skipping alone leads to substantial accuracy degradation, even when skipping only one or two layers (*i.e.*, two or four sub-layers). For instance, on AIME 25 and LCB-v5, skipping two layers reduces accuracy from $30.0\% \rightarrow 13.3\%$ and $34.8\% \rightarrow 11.8\%$, respectively. On GPQA-D, skipping a single layer already decreases accuracy from $46.0\% \rightarrow 32.8\%$. These results are summarized in Table 5 (LS; row 1-3).

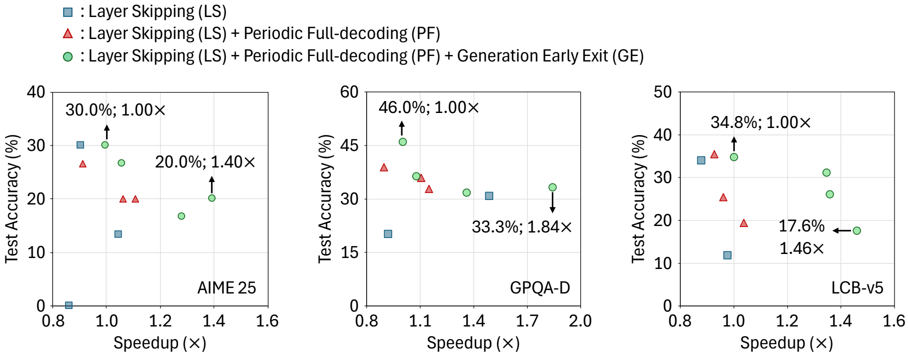
These findings indicate that the efficiency-accuracy trade-off in deep reasoning fundamentally differs from that in short multi-stage reasoning. We attribute this gap to the intrinsic characteristics of deep reasoning. Unlike short multi-stage reasoning, where intermediate reasoning can be decomposed into relatively independent stages, deep reasoning requires maintaining and refining long-horizon intermediate states over extended decoding steps. In this regime, approximation errors introduced by layer skipping tend to accumulate rather than being corrected in later stages, leading to rapid accuracy collapse.

Periodic Full-decoding. Another contributing factor to this discrepancy is the difference in computational flow between multi-stage reasoning and deep reasoning. In multi-stage reasoning, once a stage is completed, the model performs a prefill step that incorporates both the output of the previous stage and the prompt of the current stage. Because our acceleration targets only the decoding phase rather than this prefill step, the model naturally reprocesses the previous stage's output at full depth, which helps mitigate accumulated approximation errors.

Extending this behavior to deep reasoning models, we introduce periodic full-layer decoding to reduce error accumulation caused by sustained layer skipping. Specifically, we adopt a simple heuristic in which the model decodes the first 1000 tokens using full-layer computation, followed by the next 1000 tokens under layer skipping, and repeats this alternating pattern throughout generation. As shown in Table 5 (LS+PF; row 4-6), this strategy substantially alleviates accuracy degradation. For example, at "# Skip": 3, accuracy improves from $0.0\% \rightarrow 20.0\%$ on AIME 25, $20.2\% \rightarrow 32.8\%$ on GPQA-D, and $0.0\% \rightarrow 25.5\%$ on LCB-v5.

However, the overall speedup remains marginal, reaching at most $1.15\times$ (at "# Skip": 3 on GPQA-D). As observed consistently in our multi-stage reasoning experiments, layer skipping often induces longer generation lengths, preventing per-token speedups from translating into meaningful end-to-end latency reductions.

Generation Early Exit. Finally, we evaluate the end-to-end speedup achieved by combining genera-



(a) (a) (c)

Figure 10: **Deep Reasoning Performance.** Test accuracy and end-to-end speedup are compared between the three configurations (LS, LS+PF, and LS+PF+GE) on the three benchmarks, (a) AIME 25, (b) GPQA-D, and (c) LCB-v5. Results are experimented using Qwen3-1.7B.

tion early exit with layer skipping and periodic full-layer decoding. As shown in Table 5 (LS+PF+GE; row 7-9), this configuration attains speedups of up to 1.40× on AIME 25, 1.84× on GPQA-D, and 1.46× on LCB-v5. These gains are primarily driven by reduced generation lengths, which are also reported in the table.

Although generation early exit introduces some accuracy degradation at a given number of skipped layers, Figure 10 demonstrates that incorporating early exit consistently yields superior efficiency–accuracy trade-offs. This result highlights the importance of combining generation early exit with layer skipping to translate per-token computational savings into meaningful end-to-end speedups.

Discussion. We find that combining layer skipping with generation early exit improves inference speed. At the same time, *all* evaluated configurations exhibit noticeable accuracy degradation, even under moderate approximation. These observations imply that deep reasoning exhibits a distinct efficiency–accuracy trade-off compared to short multi-stage reasoning.

This diagnostic study reinforces the design choice of LiteStage. While stage-aware layer skipping is effective for short multi-stage reasoning where stage-level heterogeneity can be exploited to balance latency and accuracy, it is fundamentally mismatched with deep reasoning tasks that demand strong long-horizon consistency. Addressing deep reasoning efficiently likely requires complementary mechanisms, such as the periodic full-decoding, which is however beyond the scope of this work.

E Ethics Statement

Although language models inherently present concerns regarding misuse, bias, and fairness, this work focuses solely on algorithmic and efficiency-oriented contributions. We do not foresee introducing any additional risks beyond those already associated with the base models.

Large Language Models (LLMs) were not used in developing research ideas, designing methodologies, or performing analyses. Their involvement was strictly limited to editorial refinement—such as improving clarity, grammar, and phrasing—of text originally written by the authors. No scientific content, reasoning, or experimental descriptions were produced by LLMs.

Question: When wildlife reproduce we often refer to what comes out as what?
(A) raise children (B) have children (C) photo copy (D) offspring (E) accidently got pregnant somehow

Full-layer

🔎 **[Stage 1**: Recall] The term for the offspring of a wildlife, focusing on the biological and ecological process of producing new life.
🧩 **[Stage 2**: Analysis] For option A, \"raise children\" refers to the process of bringing forth offspring, typically seen in viviparous animals, but does not specifically describe the biological origin of new life. For option B, \"have children\" is a general term for producing offspring, but it doesn't specifically refer to the biological process of transferring water, which is central to the question. For option C, \" photo copy\" refers to a digital version of a document or image, which is unrelated to the biological process of producing new life. For option D, \"offspring\" is the term used to describe the biological and ecological process of producing new life, specifically in the context of wildlife reproduction. For option E, \"accidentally got pregnant suddenly\" refers to an unintended pregnancy, which is not the biological or ecological primary term for wildlife offspring.
✅ **[Stage 3**: Summary] So the answer is option D.

(a)

Layer Skip without Generation Early Exit

🔎 **[Stage 1**: Recall] The term used to describe the process of wildlife reproduction, focusing on the biological term that accurately describes the outcome.
🧩 **[Stage 2**: Analysis] For option A, \"raise children\" refers to the biological process of producing offspring, not the specific term used for the act of reproduction itself. For option B, \"have children\" is a general term that could apply to many contexts (e.g., in animals or as a metaphor for a bad character or condition, and does not specifically describe the biological process of reproduction. For option C, \" photo copy\" refers to a visual representation of a person, which is unrelated to the biological process of producing offspring. For option D, \"offspring\" is the term used to describe the biological process of producing offspring, which aligns with the concept of wildlife reproduction. For option E, \"accidentally getting pregnant\" refers to an unintended and unintended outcome, which is not the primary focus of the question. The correct term for the process of producing the necessary quantity of offspring from a wild source of water.
✅ **[Stage 3**: Summary] So the answer is option D.

(b)

Layer Skip with Generation Early Exit

🔎 **[Stage 1**: Recall] The term used to describe the process of wildlife reproduction, focusing on the biological term that accurately describes the outcome.
🧩 **[Stage 2**: Analysis] For option A, \"raise children\" refers to the biological process of producing offspring, not the specific term used for the act of reproduction itself. For option B, \"have children\" is a general term that could apply to. many fields, not specifically the biological process of reproduction. For option C, \" photo copy\" refers to a visual representation of a. For option D, \"offspring\" is the term used to describe the biological process of producing offspring, which aligns with the concept of wildlife reproduction. For option E, \"accidentally getting pregnant\" refers to an unintended and unintended outcome, which is not the primary focus of the question. The correct term for the process of producing the necessary quantity of.
✅ **[Stage 3**: Summary] So the answer is option D.

(c)

Figure 11: **Example of Generation Early Exit.** (a)-(c) the three-stage reasoning outcomes from models with full-layer, with layer skipping but without generation early exit, and with layer skipping and generation early eixt in a CSQA test sample.

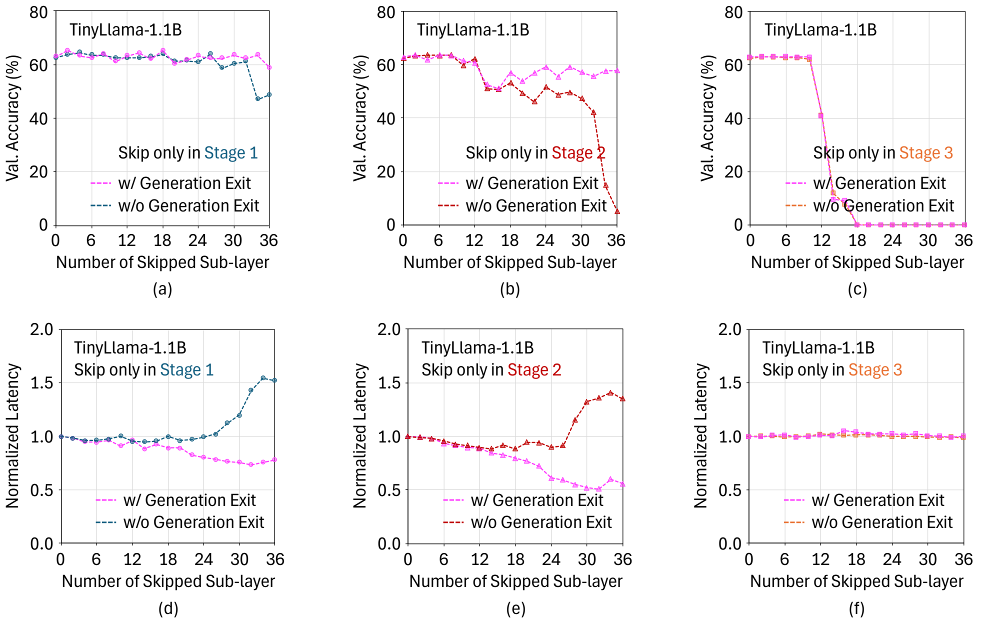


Figure 12: **Ablation: TinyLlama-1.1B on OBQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on OBQA, comparing configurations with and without generation early exit.

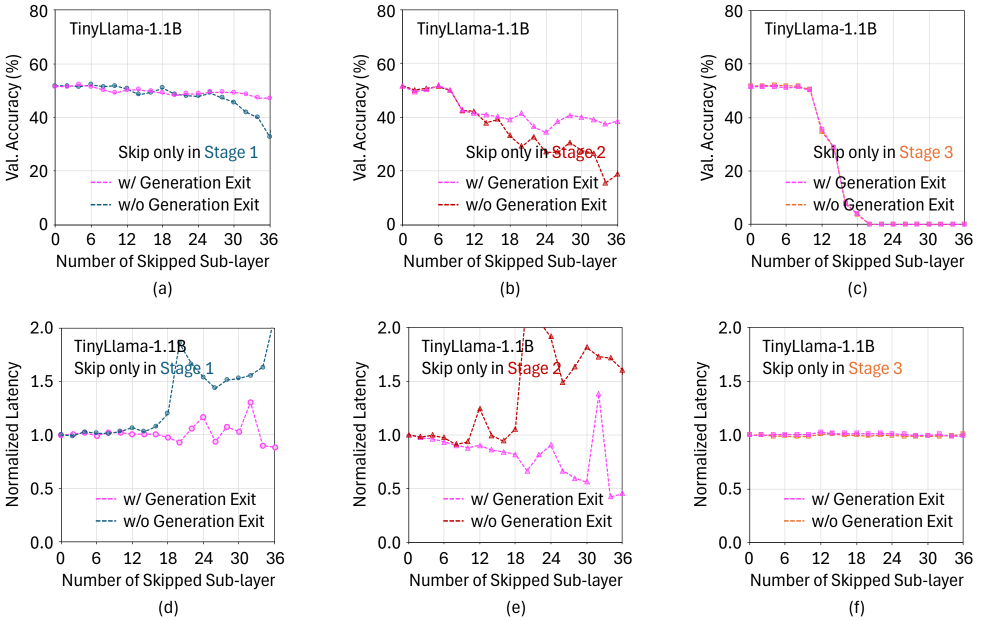
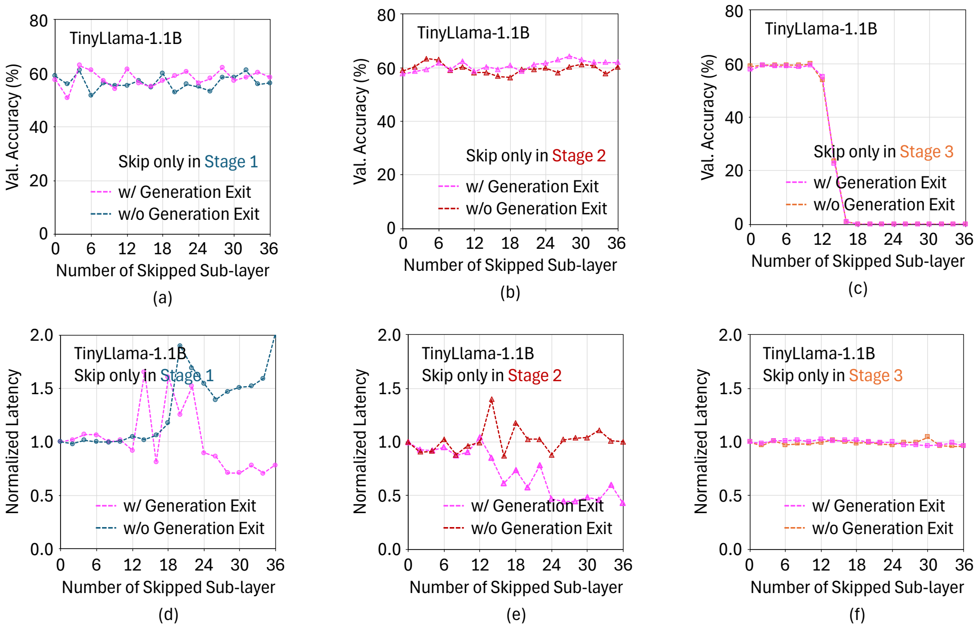
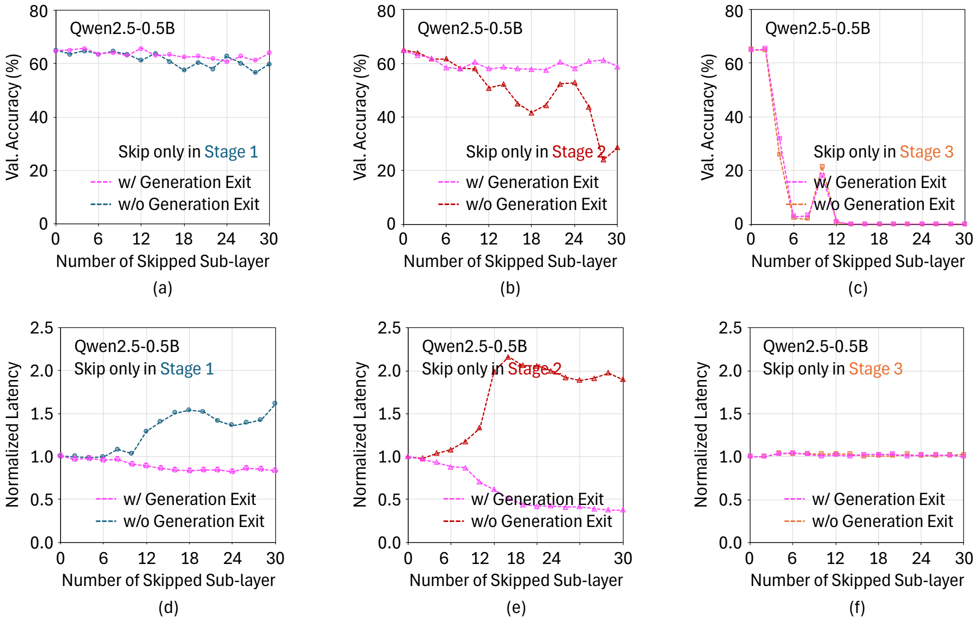


Figure 13: **Ablation: TinyLlama-1.1B on CSQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on CSQA, comparing configurations with and without generation early exit.



(a) (b) (c) (d) (e) (f)

Figure 14: **Ablation: TinyLlama-1.1B on StrategyQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on StrategyQA, comparing configurations with and without generation early exit.



(a) (b) (c) (d) (e) (f)

Figure 15: **Ablation: Qwen2.5-0.5B on OBQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on OBQA, comparing configurations with and without generation early exit.

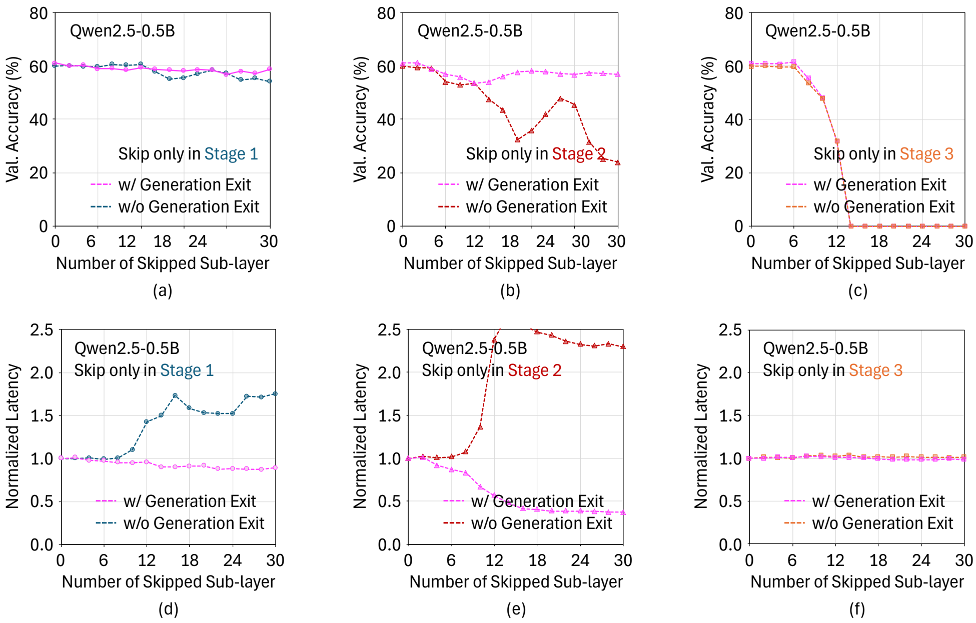


Figure 16: **Ablation: Qwen2.5-0.5B on CSQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on CSQA, comparing configurations with and without generation early exit.

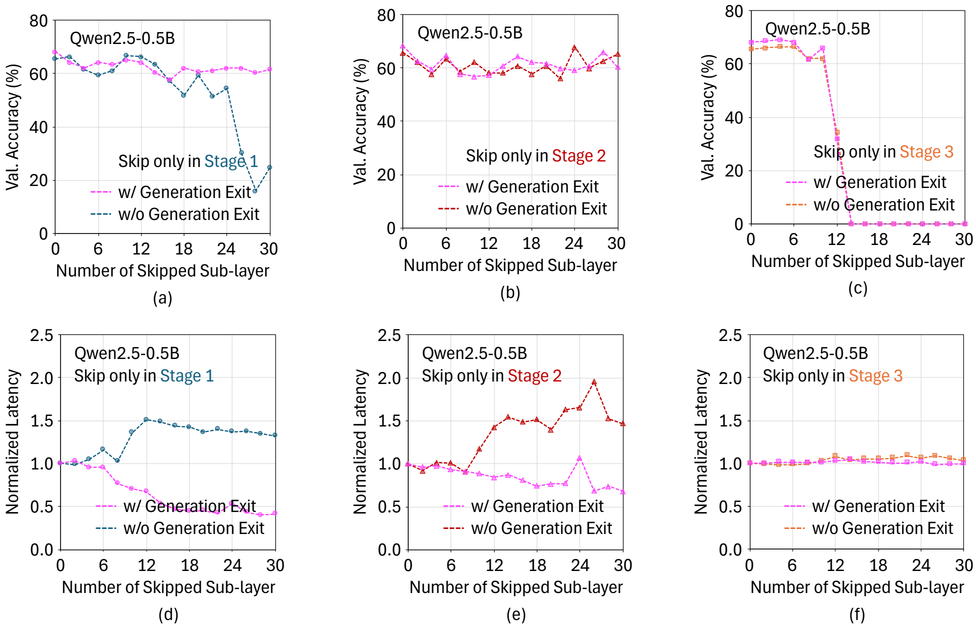


Figure 17: **Ablation: Qwen2.5-0.5B on StrategyQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on StrategyQA, comparing configurations with and without generation early exit.