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 achieves up to 1.70× speedup with less than 3.5% accuracy loss, outperforming prior training-free layer skipping methods.

1 Introduction

Decomposing a complex problem into smaller subproblems often makes it easier to understand, reason about, and solve (Wang and Chiew, 2010). This principle has been widely adopted in recent Large Language Models (LLMs), particularly for reasoning tasks (Xue et al., 2024; Dai et al., 2024; Xu et al., 2024; Zhou et al., 2024). Beyond guiding models to generate reasoning paths within a single inference, prompting sub-questions and addressing them sequentially across multiple inferences provides explicit reasoning structures (see Figure 1(a)). This strategy is especially effective for small LLMs, whose limited capacity often prevents them from producing high-quality reasoning paths in a single pass (Piao and Park, 2024; Li et al., 2024).

Despite lightweight models being inherently efficient, relying on such multi-stage inference, however, inevitably increases latency (Kim et al., 2024).

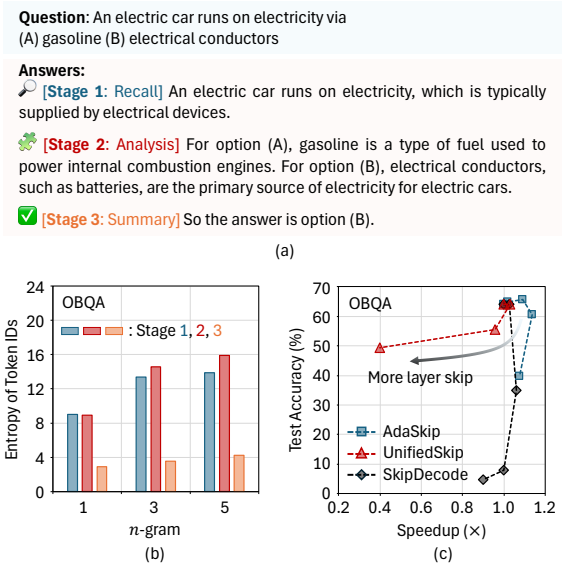

Figure 1: **Multi-stage Reasoning.** (a) Example of multi-stage reasoning introduced in TinyThinker (Piao and Park, 2024). The process consists of three stages, where each inference plays a distinct yet complementary role, jointly leading to a stronger conclusion. Stage 1 (Recall) generates an initial idea for solving the problem, Stage 2 (Analysis) evaluates each option with reasoning, and Stage 3 (Summary) delivers the final conclusion. (b) Entropy of token IDs generated across the three stages. Higher entropy indicates greater diversity in the generated tokens. (c) Accuracy-speedup trade-off in recent training-free layer-skipping techniques. Speedup is normalized to the latency of the full-layer model.

To preserve both efficiency and reasoning capability, one might consider applying existing acceleration techniques for LLMs. Nevertheless, multi-stage inference introduces unique challenges that limit the effectiveness of such methods. In particular, it often exhibits highly varying numbers of decoding steps and different degrees of token diversity across reasoning stages (see Figure 1(b)) (Dai et al., 2024). This implies that information density is non-uniform across stages: while some stages tolerate compression well, others are highly sensitive, ultimately bounding the achievable accuracy by compression techniques (**varying sensitivity**).

Hence, compression techniques that can readily adjust the efficiency level across reasoning stages are required. Adaptive computation in LLMs has been widely explored through *layer skipping* (Raposo et al., 2024; Men et al., 2024; He et al., 2025), where redundant layers are skipped to save computation. However, beyond the accuracy concern, we observe that such compression does not always ensure speedup. This is mainly because even though the per-token latency decreases, the approximated decoding frequently leads the models to generate more tokens, eventually growing the end-to-end latency. Figure 1(c) highlights this trade-off: excessive layer skipping often results in sub-1.0× speedup, meaning even slower inference than the full-layer baseline (**extra output tokens**).

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 from the slowest stage to the fastest for skip configurations that minimize latency within an accuracy threshold. This stage-wise allocation assigns each stage a tailored *layer budget*, *i.e.*, the number of layers to skip, based on its sensitivity, preventing accuracy collapse in skip-sensitive stages while further accelerating skip-insensitive ones. 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. Our key contributions are as follows:

- We present **LiteStage**, a novel latency-aware layer-skipping framework for efficient multi-stage reasoning, tackling *critical* yet *under-explored* challenges: varying stage sensitivity and the generation of extra output tokens.
- LiteStage introduces 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).
- Extensive experiments on three benchmarks demonstrate that LiteStage consistently outperforms prior techniques in both accuracy and latency, achieving up to 1.16-1.70× inference speedup with 0.4-3.4% accuracy loss.

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). Despite their effective stage structuring, they largely prioritize reasoning quality over 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 often depend on carefully curated reasoning data from large LLMs, 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.*, 1st, 4th, 7th); 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. Although they require neither additional training data nor retraining, these methods typically apply uni-

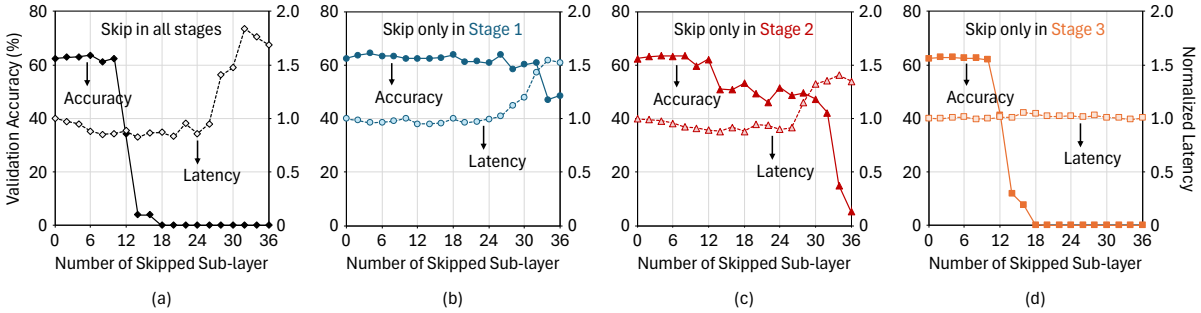

Figure 2: **Motivational Study.** (a) Accuracy and latency when the same number of layers are skipped across all reasoning stages. (b)-(d) Accuracy and latency when layer skipping is independently applied to a single stage, from Stage 1 through Stage 3, respectively. Notably, the accuracy trend in (d) closely aligns with that in (a). Furthermore, greater layer skipping does not always guarantee lower latency. Results are obtained on the TinyThinker OBQA dataset using AdaSkip-based layer importance in TinyLlama-1.1B.

form skipping policies, leading to suboptimal efficiency–accuracy trade-offs.

Generation Early Exit. Prior studies on generation early exit primarily address inefficiencies in long reasoning models that generate redundant explanations. For example, 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, treating it as convergence, while logit-based approaches (Yang et al., 2025; Wang et al., 2025) monitor confidence or next-token entropy when a *</think>* token is appended to 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 Motivation

Our key assumption is that the sensitivity to layer skipping varies substantially across reasoning stages. To validate this, we perform per-stage skipping experiments, in which layers are skipped only within a single stage, while all other stages are executed at their full depth.

Why Non-uniform Layer Skip? Figure 2(a) shows the validation accuracy by layer skipping uniformly applied across stages, *i.e.*, the same number of layers are skipped regardless of stage. Notably, this approach causes the accuracy to drop sharply after about ten sub-layers (*e.g.*, multi-head self-attention or feed-forward network) are skipped. In contrast, Figures 2(b)-(d) demonstrate the accuracy when only one of Stage 1, 2, and 3, respectively, is under layer skipping. We observe that the accuracy profile in Figure 2(d) closely resembles that of Figure 2(a), implying the uniform layer skipping's accuracy is bounded by this Stage 3.

This result motivates the question: what if we could protect the model from layer skipping during Stage 3 while allowing more layers to be skipped in Stages 1 and 2? Given that Stage 1 exhibits much greater robustness, such a strategy could potentially alleviate overall accuracy degradation. However, this raises a subsequent question: *how many layers should be skipped in each stage?* Data complexity is often used as a proxy for determining a model's compression budget (Liu et al., 2022; Lin et al., 2017). However, our entropy analysis (Figure 1(b)) reveals an interesting contradiction, where the token entropy in Stage 3 is the lowest, suggesting lower diversity and, in principle, less computational demand. Nevertheless, Stage 3 is empirically the most sensitive to layer skipping. This counterintuitive finding highlights the need to validate layer skipping configurations against actual accuracy profiles, such as Figures 2(b)–(d)), rather than relying solely on heuristic proxies.

Why Latency-aware Layer Skip? Given that accuracy degradation is most robust in Stage 1, one might allocate more skipped layers to this stage. However, this alone would not yield substantial inference speedup. As shown in Figure 2(b)-Latency, the latency gain from skipping layers in Stage 1 is minimal because most decoding steps take place in Stage 2. A notable observation is that latency can even increase as layer skipping progresses, despite maintaining a similar level of accuracy. This happens because layer skipping often leads to longer generation, amplifying both the decoding overhead within a stage and the prefill cost in later stages. These results suggest that accuracy alone does not

adequately capture the effectiveness of skip configurations. However, this adverse latency behavior has been largely overlooked in prior studies.

These findings motivate a stage-wise, latency-aware approach to layer skipping. Instead of treating all stages equally, skipping policies should balance stage sensitivity (accuracy) with inference cost (latency), ensuring that skipping truly delivers reliable efficiency gains.

4 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.

It is worth noting that other methods have focused on which layers to skip using heuristics, such as periodic skipping (Liu et al., 2024) and consine similarity (Men et al., 2024). As we already observe that more layer skipping does not always lead to speedup (Figure 2(b)), such latency-agnostic proxy is unlikely to provide the optimal layer choices. While brute-force search-based approaches can ensure the globally optimal solution, the search space in our case is extremely large, *e.g.*, TinyLlama-1.1B with 44 sub-layers (22 decode blocks), each reasoning stage corresponds to 2^{44} cases. This poses a challenging problem: how to efficiently allocate the optimal layer budget across stages.

4.1 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 3.

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 slowest reasoning stage to the fastest to effectively reduce the latency within an accuracy threshold (**Step 2**). (2) an *online adjustment*, which addresses inefficiencies that may occur due to unexpectedly extended generation. We observe that the skipping lengthens generation but the confidence level diminishes. Considering that, we jointly apply generation early exit with layer skipping (**Step 3**).

4.2 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 the 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 *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:

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

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

where $I^{(j)}_{\text{MHSA}}$ and $I^{(j)}_{\text{FFN}}$ 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 3 (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.

Although this estimation is first performed using prompts, the recursive nature of multi-stage inference ensures that outputs generated at one stage

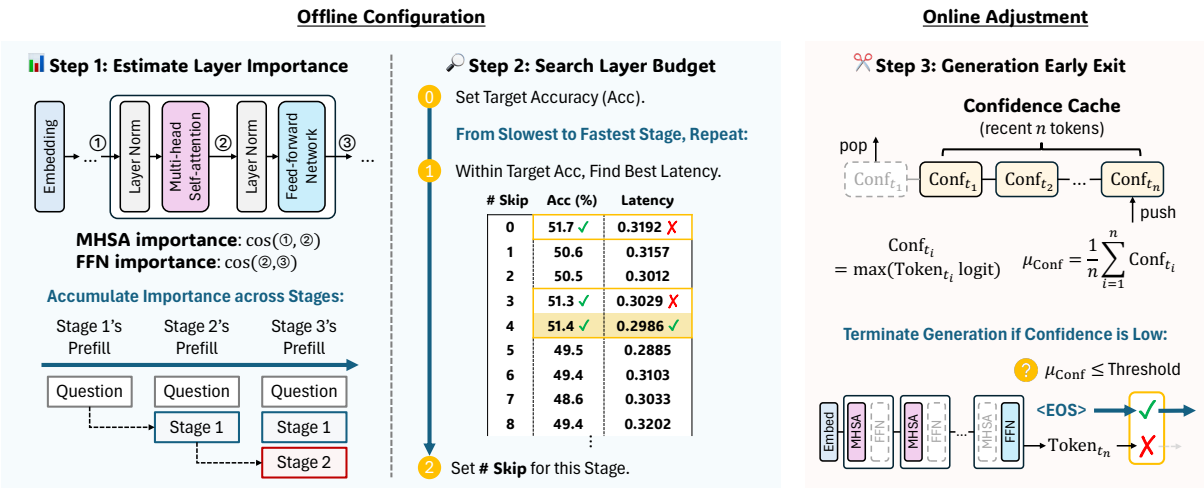

Figure 3: **Overview of LiteStage.** The proposed method consists of an *offline configuration* (Steps 1–2) and an *online adjustment* (Step 3). In the offline phase, (1) layer importance is estimated at the sub-layer level (MHSA and FFN) and accumulated across stages, followed by (2) a search for the 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.

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 for each dataset.

4.3 Step 2: Search Layer Budget

The key contribution of LiteStage 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 an accuracy-latency profile by varying the layer budget in the slowest reasoning stage using validation data, while keeping all other stages full-layer.

Skipping from the Slowest Stage. Figure 3 (Step 2) illustrates this process using the actual accuracy-latency profile of Stage 2, which is the slowest stage, on the CSQA dataset with a TinyLlama 1.1B model. 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 slowest 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). This ensures that the interaction between reasoning stages under different layer budgets are accurately reflected in the search. 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 slowest stage will be effectively accelerated.

Protection from Latency Degradation. We observe that beyond the "# Skip" of 5, the latency gain does not appear, and additional layer skipping even worsens the overall inference speed. This phenomenon occurs because, although per-token latency decreases, the total number of generated tokens increases, leading to higher end-to-end latency. Since our search process jointly considers both accuracy and latency, such configurations are automatically excluded even when the target accuracy is set to lower values.

4.4 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.

Extra Tokens may not be Useful. Figure 4 illustrates how token confidence evolves over decoding steps under four different skip configurations. Here, token confidence is defined as the maximum logit

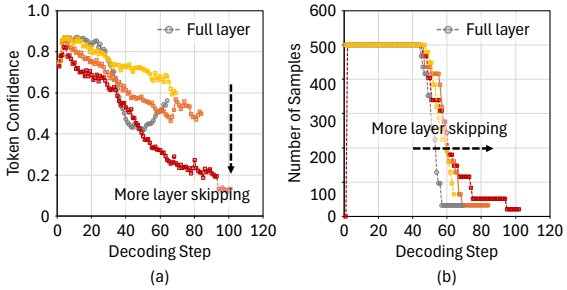

Figure 4: **Confidence of Output Tokens.** (a) Average token confidence over decoding steps with varying layer skipping levels. (b) Number of remaining samples during decoding, where more samples are associated with generation early exit in larger skip budgets due to confidence decay. Results include skipping of 10 (yellow), 15 (orange), and 20 (red) sub-layers in Stage 1 of OBQA validation using TinyLlama-1.1B.

value of each generated token. The confidence trajectories differ between models with and without layer skipping, *e.g.*, the layer-skip models exhibit a monotonic 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 3 (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.

5 Experimental Results

Datasets. 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.

Implementation Details. We primarily evaluate our method using TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2024). Our training and evaluation procedures follow TinyThinker (Piao and Park, 2024), with adjusted hyperparameters for the TinyLlama architecture. We train 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 protocol with 10 iterations.

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.

5.1 Comparison with Baselines

Figure 5 provides a comprehensive comparison between our proposed LiteStage and three baseline methods across the OBQA, CSQA, and StrategyQA datasets. Each method is 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 5(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 1.32× speedup with 60.0% accuracy.

We highlight two key observations from these results: (1) how LiteStage mitigates severe accuracy degradation (0% → 60%) through its **Non-uniform Layer Budget**, and (2) how it extends the achievable latency limit (1.10× → 1.32×) via **Generation Early Exit**.

5.2 Benefits of Non-uniform Layer Budget

To evaluate the effectiveness of our non-uniform layer skipping compared to uniform baselines, Ta-

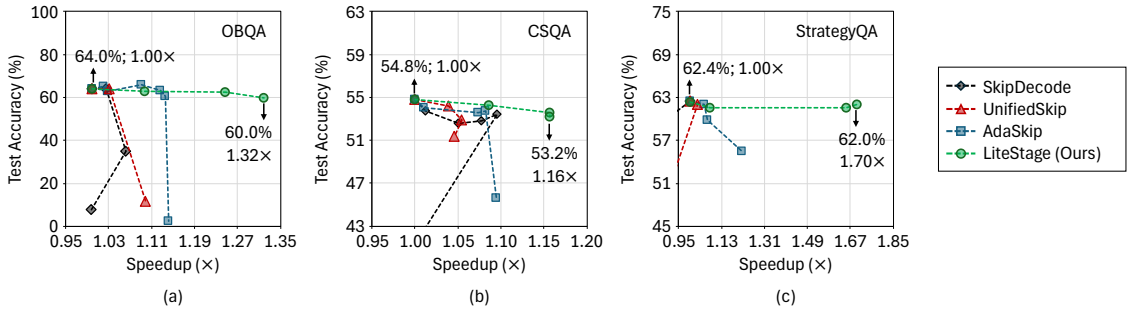

Figure 5: **Comparison with Baselines.** (a)–(c) present the accuracy–speedup trade-offs on the OBQA, CSQA, and StrategyQA datasets, respectively. The performance of the full-layer model is marked in the upper-left region of each plot (*i.e.*, 64.0% for OBQA, 54.8% for CSQA, and 62.4% for StrategyQA), and the speedup is normalized with respect to its latency. The highest achievable speedup by our method, along with its corresponding accuracy, is indicated in the lower-right region. All results are obtained using TinyLlama-1.1B models.

Table 1: **Comparison at Given Layer Budget.** Each configuration is evaluated under a similar number of skipped full layers (two sub-layers), denoted as # Skip. †In CSQA, while LiteStage's accuracy is lower than AdaSkip's, it already achieves 1.09× speedup with 54.3% accuracy at "# Skip": 2.3. Due to its dynamic mechanism, SkipDecode's # Skip represents the max number of skipped layers, seemingly achieving less speedup.

Method / Benchmark	OBQA			CSQA			StrategyQA		
	# Skip	Acc (%)	Speedup	# Skip	Acc (%)	Speedup	# Skip	Acc (%)	Speedup
Full-layer	0	64.0	1.00×	0	54.8	1.00×	0	62.4	1.00×
SkipDecode (Del Corro et al., 2023)	4.0	64.0	1.03×	2.0	51.8	1.05×	1.0	62.4	0.93×
UnifiedSkip (Liu et al., 2024)	4.0	55.6	0.96×	2.0	53.4	1.04×	1.0	62.4	0.94×
AdaSkip (He et al., 2025)	4.0	63.2	1.13×	2.0	53.6	1.07×	1.0	62.0	1.06×
LiteStage (Ours)	3.7	62.8	1.10×	2.3	54.3	1.09×	0.7	61.6	1.08×
SkipDecode (Del Corro et al., 2023)	6.0	62.4	1.03×	3.0	51.8	1.05×	9.0	59.4	0.73×
UnifiedSkip (Liu et al., 2024)	6.0	11.8	1.10×	3.0	47.0	1.05×	9.0	0	0.54×
AdaSkip (He et al., 2025)	6.0	39.8	1.08×	3.0	53.7	1.08×	9.0	0	0.61×
LiteStage (Ours)	6.3	62.4	1.25×	3.0	53.0†	1.11×	9.3	61.6	1.65×
SkipDecode (Del Corro et al., 2023)	7.0	62.6	1.04×	4.0	52.6	1.05×	10.0	61.1	0.67×
UnifiedSkip (Liu et al., 2024)	7.0	11.8	1.10×	4.0	39.8	0.76×	10.0	0	0.53×
AdaSkip (He et al., 2025)	7.0	2.4	1.14×	4.0	54.0	1.09×	10.0	0	1.12×
LiteStage (Ours)	7.0	60.6	1.32×	4.3	53.6†	1.16×	10.3	62.0	1.70×

ble 1 reports the accuracy-speedup trade-offs under the same number of skipped layers. The accuracy gap between LiteStage and the baselines becomes increasingly pronounced as more layers are skipped. This is primarily because Stage 3 is substantially more sensitive to layer skipping than other stages, thus upper-bounding the overall accuracy, as shown in Figure 2 (Motivation). By assigning fewer skipped layers to such sensitive stages, our method maintains accuracy even when baselines collapse after more than one stage fails, highlighting the effectiveness of our non-uniform layer skipping.

Beyond merely protecting sensitive stages, our approach adapts the layer budget distribution across datasets. Table 2 presents the number of layers allocated to each stage by LiteStage. In the OBQA dataset, skipping layers in Stage 1 is less influential in final accuracy, whereas in StrategyQA, LiteStage allocates a larger skip budget to Stage 2. Meanwhile, in the CSQA dataset, the layer distribution is relatively uniform across stages. We observe that skipping more than four layers in CSQA causes either a substantial increase in generation length or noticeable accuracy degradation in both Stages 1 and 2. This indicates that CSQA exhibits uniform sensitivity to layer skipping, resulting in a reduced performance gain from LiteStage.

Notably, LiteStage consistently avoids skipping more than five layers in Stage 3, highlighting its ability to adaptively constrain aggressive compression in sensitive stages while intensively accelerating more robust ones. In summary, LiteStage not only protects vulnerable reasoning stages but also strategically maximizes efficiency where performance degradation is minimal.

Table 2: **Non-uniform Layer Budget.** The number of skipped layers allocated to each stage by LiteStage and the baselines across the three benchmarks. Results illustrate how the layer budget varies by dataset.

	Method	# Skip	Stage 1	Stage 2	Stage 3
OBQA	Baselines	7.0	7	7	7
	Ours	3.7	2	4	5
	Ours	6.3	11	4	4
	Ours	7.0	11	6	4
CSQA	Baselines	4.0	4	4	4
	Ours	2.3	0	3	4
	Ours	3.0	0	4	5
	Ours	4.3	4	4	5
Stra-QA	Baselines	10.0	10	10	10
	Ours	0.7	0	2	0
	Ours	9.3	8	19	1
	Ours	10.3	8	20	3

5.3 Benefits of Generation Early Exit

Figures 6(a)-(c) present ablation studies analyzing the effect of applying generation early exit along with layer skipping. When only a few layers are skipped (*e.g.*, 4.0 layers in OBQA), the difference in the number of decoding steps between models with and without generation early exit is negligible (*e.g.*, -0.5%). This indicates that the layer-skip models still generate tokens confidently, with decoding lengths comparable to the full-layer baseline. Thus, the speedup at this level of skipping primarily arises from the non-uniform layer budget, for example, by skipping more layers in Stage 2.

As more layers are skipped, however, the gap in decoding steps increases consistently (*e.g.*, -19.6% in OBQA and -82.5% in StrategyQA), producing sequences shorter than those of the full-layer baseline and resulting in proportional latency gain. That is, generation early exit enables layer skipping to realize its true end-to-end latency gain, which would otherwise be masked by the extended generation. As a result, the speedup gap between LiteStage and the baselines becomes more pronounced at higher skip configurations, as shown in Table 1.

The corresponding accuracy results are shown in Figures 6(d)–(f). The accuracy with and without generation early exit remains comparable in most cases, suggesting that the tokens omitted by early termination contribute little to the final prediction. However, the variation slightly increases as a larger portion of decoding steps is truncated. For example, in OBQA, the accuracy drops by 2.2% when 19.6% decoding steps are reduced. Interestingly, in StrategyQA, aggressively reducing decoding steps in StrategyQA by nearly 80% improves accuracy by 2.6% and 3.9%. We observe that most of these terminated tokens occur in Stage 2, implying that

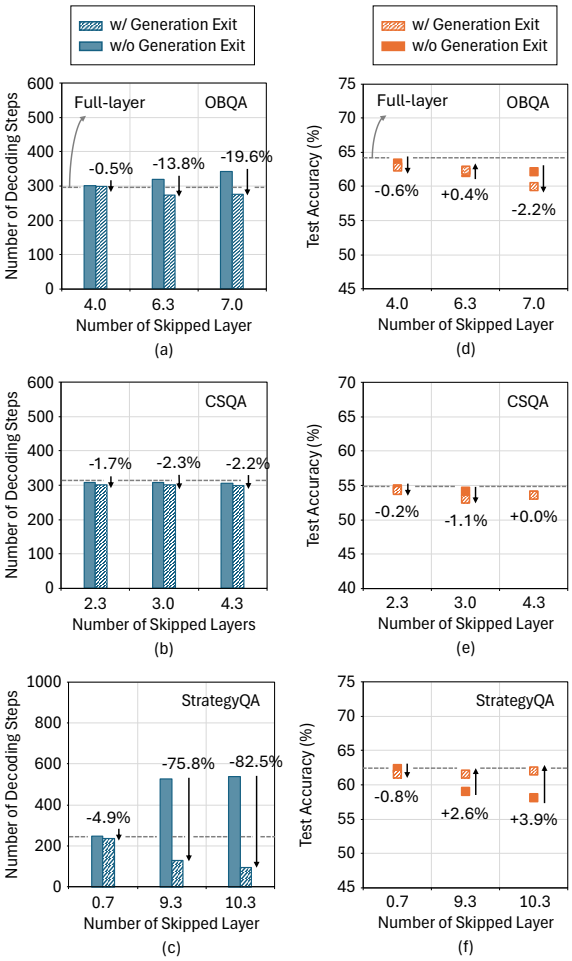

Figure 6: **Ablation Study.** (a)–(c) show the average number of decoding steps (*i.e.*, generation length) per question with and without generation early exit as the number of skipped layers increases. The decoding steps are summed across the reasoning stages. (d)–(f) present the corresponding test accuracy.

Stage 2 may generate redundant information that propagates to later stages, potentially confusing the model. Despite these minor variations in accuracy due to generation early exit, LiteStage consistently outperforms the baseline methods.

6 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 achieves up to 1.70× speedup with minimal accuracy loss, surpassing prior training-free methods. Our results highlight the importance of stage-aware optimization and adaptive decoding in realizing truly efficient multi-stage reasoning.

7 Limitations and Future works

LiteStage consists of two key components: *offline configuration* and *online adjustment*. The second step of the offline configuration—searching for the optimal layer budget—requires more computation than prior layer-skipping methods that only perform layer importance estimation (Step 1). However, this process occurs only once offline and enables LiteStage to achieve superior performance. The approximate evaluation time is provided in Appendix A.

Our experiments primarily focus on Llama-type models. Although we conducted a motivational study using Qwen2.5-0.5B (Appendix B), we observed that Qwen-based architectures are inherently sensitive to layer skipping, which substantially reduces the viable search space for LiteStage. Extending our framework to other small LLM architectures and further analyzing model-specific skip sensitivity remain promising directions for future work.

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*.

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.

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.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. *Advances in neural information processing systems*, 30.

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*.

Zhenhua Liu, Yunhe Wang, Kai Han, Siwei Ma, and Wen Gao. 2022. Instance-aware dynamic neural network quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12434–12443.

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. Diff-skip: 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*.

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*.

Qwen, :, 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, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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*.

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

Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. 2025. Entropy after ⟨/Think⟩ for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*.

Yingxu Wang and Vincent Chiew. 2010. On the cognitive process of human problem solving. *Cognitive systems research*, 11(1):81–92.

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, Li Hao, Yibing Song, Lichao Sun, and Li Yuan. 2024. Llava-o1: Let vision language models reason step-by-step. *arXiv preprint arXiv:2411.10440*.

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.

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

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 Additional Details on LiteStage

A.1 Estimate Layer Importance

Figure 8 shows the sub-layer-wise importance estimated in Step 1. We use the TinyLlama-1.1B model and the valdation datasets in the three benchmarks of OBQA, CSQA, and StrategyQA. We observe that the overall trend of layer importance in both MHSA and FFN is consistent across the datasets. However, the level of importance slightly shifts. For example, in StartegyQA, the FFN's importance is relatively higher than the other datasets, making the layer skipping primarily occurs at self-attention layers.

A.2 Search Layer Budget

All experiments are conducted on a single NVIDIA A6000 GPU (48GB). Compared to other training-free layer-skipping methods, LiteStage involves additional computation in the *offline configuration* and *online adjustment* phases. In the offline phase, the model repeatedly evaluates the validation set while varying the layer budget; thus, search time scales with the number of configurations explored. For reference, a single validation pass with the full-layer model takes 2.6 minutes on OBQA, 7.2 minutes on CSQA, and 1.0 minute on StrategyQA. Assuming searching the layer budget in a full-layer interval (*i.e.*, every two sub-layers), this validation will repeat 21 times for each reasoning stage, which approximately incurs, 2.7 hours (OBQA), 7.6 hours (CSQA), 1.0 hour (StrategyQA) to search a layer skipping configuration. However, this search is performed only once to determine the optimal layer budget. The online adjustment maintains a confidence cache, storing batch size$\times$five floating-point values, and computes their mean, adding only negligible runtime overhead.

A.3 Generation Early Exit

Figure 7 illustrates a CSQA test example comparing reasoning outputs with and without generation early exit. For reference, the full-layer baseline prediction is also shown in Figure 7(a). Since generation early exit terminates decoding once token confidence drops below a threshold, the generated tokens remain identical up to that point. The truncated portion is highlighted in blue. For instance, the full reasoning "For option C, photo copy refers to a visual representation of a person, which is unrelated to the biological process of producing offspring." becomes "For option C, photo copy refers to a visual representation of a." when early exit is applied. This shows that the model maintains high confidence until "visual representation," while later tokens are less certain. Nonetheless, both models yield the same final conclusion in Stage 3, indicating that shorter reasoning was sufficient to reject option C.

B Additional Experimental Results

B.1 Training Results

TinyThinker (Piao and Park, 2024) is primarily implemented with the T5 (Raffel et al., 2020) architecture. As recent progress in reasoning tasks has been driven by decoder-only models such as Llama, we extend TinyThinker's framework to the Llama family. This includes fine-tuning our models on TinyThinker's benchmarks with three-stage reasoning paths. The largest TinyThinker model (T5-Large, 770M parameters) reports test accuracies of 65.4% (CSQA), 68.8% (OBQA), and 69.0% (StrategyQA), whereas our TinyLlama-1.1B achieves 54.8%, 64.0%, and 62.4% on the same datasets. We consider the possible reason for the performance gap, even with the larger-size TinyLlama, is related to training batch size. TinyThinker utilized four A100 GPUs for larger batches, while we used a single A6000 GPU. Figure 9 shows the training dynamics, including loss and validation accuracy.

B.2 Results from other Model

We also evaluate LiteStage on a model other than TinyLlama-1.1B. Qwen2.5-0.5B-Instruct (Qwen et al., 2025) is a widely used small LLM. We fine-tune it using the same training setup and hyperparameters as our TinyLlama-1.1B experiments across the three benchmarks. The full-layer model achieves 64.4% (CSQA), 70.2% (OBQA), and 65.5% (StrategyQA), consistently surpassing TinyLlama despite having roughly half the parameters. However, Qwen2.5-0.5B proves highly sensitive to layer skipping: skipping as few as 2–4 sub-layers can reduce accuracy to nearly 0% on OBQA (see Figure 10). This sensitivity leaves little room for effective optimization with LiteStage, suggesting an inherent characteristic of the Qwen architecture. For example, skipping even one or two sub-layers in Stage 2 often increases latency due to prolonged generation lengths. Consequently, we primarily report results using the TinyLlama model.

C 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

Answers (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)

Answers (Layer Skip w/o 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)

Answers (Layer Skip w/ 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 7: **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.

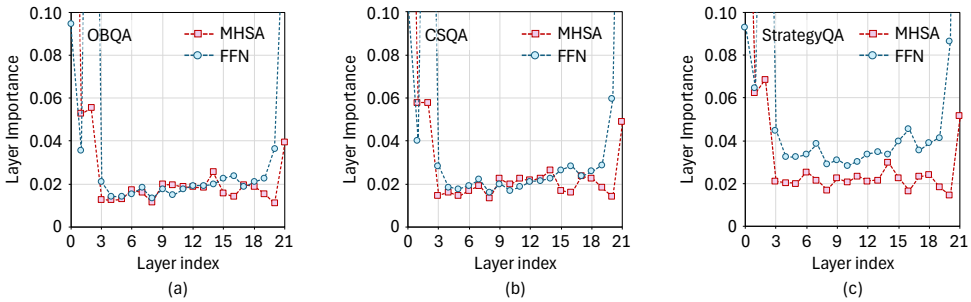

Figure 8: **Layer Importance.** (a)-(c) Layer importance estimated using cosine similarity between layer inputs and outputs across the OBQA, CSQA, and StrategyQA datasets, respectively.

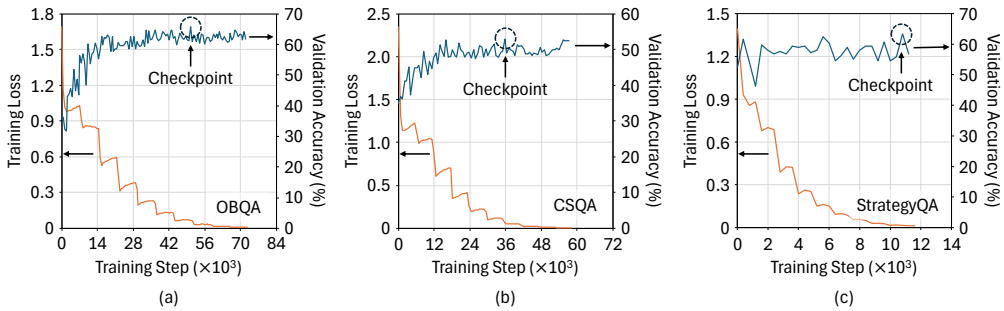

Figure 9: **Training Dynamics.** (a)-(c) represent the training loss and validation accuracy over the training steps in OBQA, CSQA, and StrategyQA, respectively. Results are obtained using TinyLlama-1.1B models.

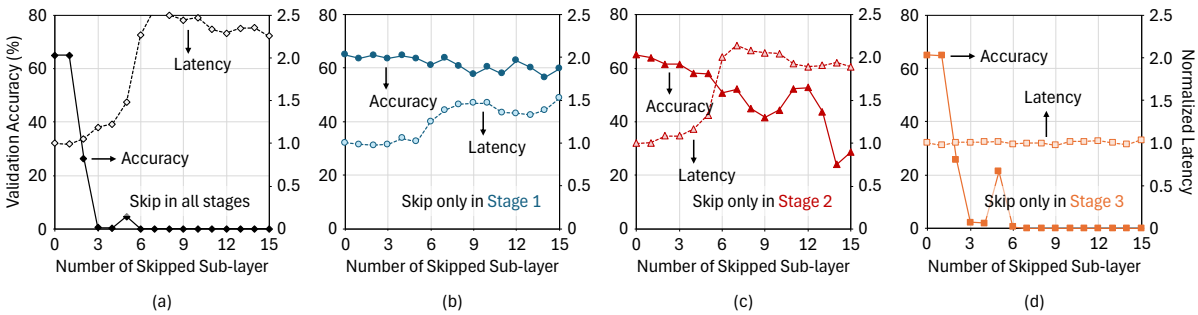

Figure 10: **Motivational Study.** (a) Accuracy and latency when the same number of layers are skipped across all reasoning stages. (b)-(d) Accuracy and latency when layer skipping is independently applied to a single stage, from Stage 1 through Stage 3, respectively. Results are obtained on the TinyThinker OBQA dataset using AdaSkip-based layer importance in Qwen2.5-0.5B.