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

Anonymous ACL submission

Abstract

Multi-stage reasoning has emerged as an effective strategy for improving 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) extra generation of 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 redundant decoding. Experiments on three benchmarks, *e.g.*, OBQA, CSQA, and StrategyQA, show LiteStage outperforms prior training-free layer skipping methods.

1 Introduction

Recent research on reasoning spans a broad spectrum, ranging from deep reasoning with long-horizon generation (Hongru et al., 2025; Jin et al., 2024) 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 few structured stages such as knowledge recall, local inference, and decision aggregation (Piao and Park, 2024) (see Figure 1(a)), allowing them to be solved without long, deeply nested derivations.

This form of *short multi-stage reasoning* is particularly common in small language models, whose limited capacity often prevents reliable single-step deep reasoning (Li et al., 2024). However, executing multiple reasoning stages incurs non-trivial latency (Kim et al., 2024), so inference can remain slow even for lightweight models. Applying existing acceleration techniques therefore appears natural, but 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, limiting achievable accuracy. Effective acceleration thus requires adapting efficiency to the distinct computational demands of each reasoning stage rather applying uniform compression.

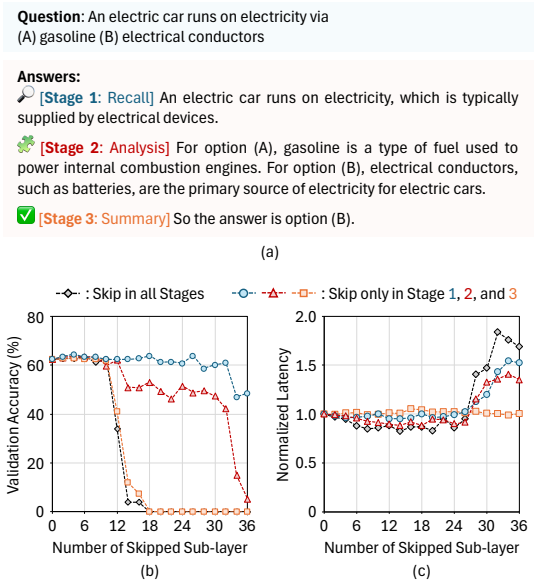

Figure 1: **Multi-stage Reasoning.** (a) An example of multi-stage reasoning introduced in TinyThinker (Piao and Park, 2024). The process comprises three stages: Stage 1 (*Recall*) generates an initial solution idea, Stage 2 (*Analysis*) evaluates candidate options through explicit reasoning, and Stage 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.

Adaptive computation in LLMs has been widely explored through *layer skipping* (Raposo et al., 2024; Men et al., 2024; He et al., 2025), which

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

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 result in 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 for multi-stage reasoning in small LLMs. LiteStage comprises two complementary components, an offline configuration and an online adjustment, that jointly address the key challenges. In the **offline phase**, LiteStage searches for *layer budget*, *i.e.*, the number of layers to skip, for each stage to minimize latency under an accuracy constraint. This search proceeds from the longest stage to the shortest, effectively accelerating slow stages while preventing accuracy collapse. In the **online phase**, LiteStage addresses the underexplored side effect of layer skipping, longer generation, by monitoring token confidence during decoding and *terminating generation early* when confidence falls below a threshold, avoiding redundant generation. LiteStage's key features are summarized in Table 1.

Our technical novelty lies in addressing the timely problem of accelerating reasoning in small LLMs. In multi-stage reasoning, we show stage-heterogeneous sensitivity that breaks the uniform-compression assumption of prior work. We also reveal an efficiency paradox where aggressive skipping increases decoding length, limiting end-to-end speedup. These observations motivate stage- and generation-aware acceleration for reasoning.

2 Related Works

Multi-stage Generation. Recent works on reasoning tasks employ various forms of multi-stage inference. TinyThinker (Piao and Park, 2024) introduces a deductive recall–analysis–summary reasoning cycle, showing progressive accuracy gains. DeAR (Xue et al., 2024) adopts a decomposition–analysis–rethinking process, refining intermediate answers across stages. CasCoD (Dai et al., 2024) distills decomposed chain-of-thoughts to enhance small-model reasoning. Self-Discover (Zhou et al., 2024) dynamically organizes reasoning structures for each problem. LLaVA-CoT (Xu et al., 2025) extends this idea to multi-modal settings. However, these works focus on reasoning quality rather than computational efficiency.

Layer Skip. Layer-skipping methods fall into: (1) *training-based* and (2) *training-free* approaches. Early works like LayerSkip (Elhoushi et al., 2024), DeeBERT (Xin et al., 2020), and EE-LLM (Chen et al., 2023) perform early exit from intermediate layers (Fan et al., 2024). Recent router-based methods including Mixture-of-Depth (Raposo et al., 2024) skip layers dynamically but require additional training, while layer studies (Luo et al., 2025b; He et al., 2024; Luo et al., 2025a; Bae et al., 2025) reduce cost by fine-tuning only routers.

Multi-stage reasoning models are often distilled from curated reasoning paths generated by larger models, which are not always available. Thus, *training-free* acceleration is more practical in our setting. Representative methods include SkipDecode (Del Corro et al., 2023), which progressively skips more layers toward deeper decoding layers; Unified Skipping (Liu et al., 2024), which periodically skips layers (*e.g.*, layers 3, 6, 9); and ShortGPT (Men et al., 2024), which uses cosine similarity to estimate block importance. AdaSkip (He et al., 2025) further estimates importance at the sub-layer level, but these methods typically apply temporally uniform skipping policies, leading to

suboptimal accuracy-efficiency trade-offs.

Generation Early Exit. Prior works mainly address redundant explanations in long reasoning traces. Zhang et al. (2025) trains probing heads to estimate confidence on intermediate answers, while training-free methods typically rely on heuristics. ES-CoT (Mao et al., 2025) stops when answers repeat, and logit-based methods (Yang et al., 2025b; Wang et al., 2025b) monitor confidence or entropy when *</think>* token appears. These approaches focus on deep reasoning models and do not address extra generation caused 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:

$$\arg\min_{\mathbb{L}} \mathbb{E}_{d\sim\mathbb{D}}\left[\mathcal{T}(\mathcal{M}_{\mathbb{L}}(d))\right] \text{ s.t. } \mathcal{A}(\mathcal{M}_{\mathbb{L}}) \geq \mathcal{A}(\mathcal{M}) - \epsilon \quad (1)$$

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

The details of each component in LiteStage are described in the following sections, and an overview is shown in Figure 2. The framework consists of two phases: (1) an *offline configuration* that determines the optimal set of sub-layers to skip at each stage, and (2) an *online adjustment* that compensates for inefficiencies during generation.

In the offline phase, we first estimate layer importance using cosine similarity to define priority of removable layers (**Step 1**). We then perform a greedy search that determines the *layer budget* (*i.e.*, the number of layers to skip) for each stage, from the longest reasoning stage to the shortest, to minimize latency within an accuracy threshold (**Step 2**). In the online phase, we address inefficiencies caused by extended generation. We observe that layer skipping often produces extra tokens with lower confidence, and thus combine it with confidence-based generation early exit (**Step 3**).

3.2 Step 1: Estimate Layer Importance

We first need a layer skipping policy for selecting which layers to skip given a target skip count. We adopt cosine similarity as a proxy for layer importance (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:

$$I^{(j)}_{\text{MHSA}} = 1 - \frac{1}{N}\sum_{n=0}^{N-1} \cos(\text{MHSA}^{(j)}(x) + x, x) \quad (2)$$

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

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 2 (Step 1), during prefilling and average over N validation samples. Equations (2) and (3) 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.

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. This process is conducted offline and only once per dataset. We report I_{MHSA} and I_{FFN} on evaluation datasets in Figure 10 (Appendix), where we observe little variation across datasets.

3.3 Step 2: Search Layer Budget

Our key contribution lies in how we search for the optimal layer budget. 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. This prioritizes accelerating the longest stage, which most effectively reduces the overall end-to-end latency. We then progressively explore layer skipping in the remaining stages. Please refer to Appendix F for a discussion on the robustness of our validation-based search method under test sets with mild distribution shifts.

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

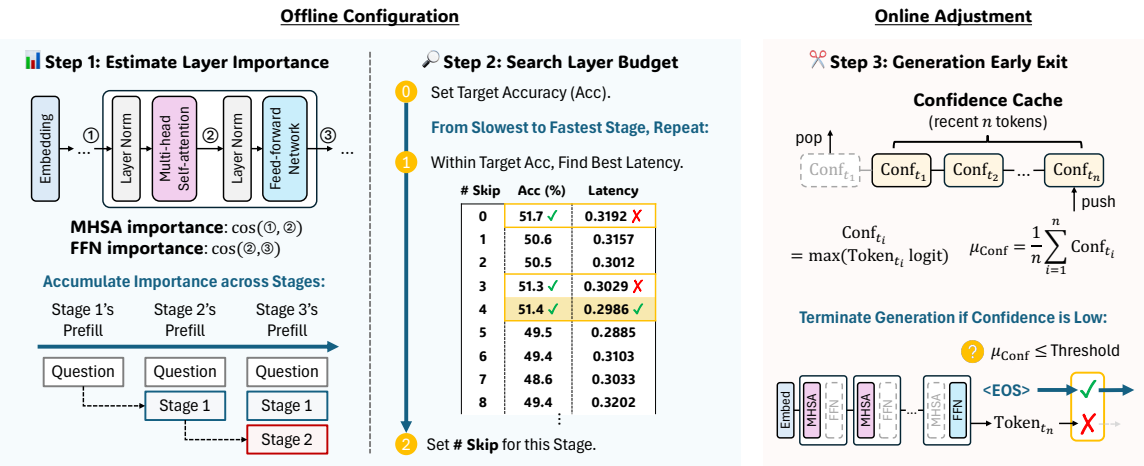

Figure 2: **Overview of LiteStage.** The proposed method consists of an *offline configuration* (Steps 1–2) and an *online adjustment* (Step 3). In the offline phase, (1) layer importance is estimated at the sub-layer level (MHSA and FFN) and accumulated across stages, followed by (2) a search for the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

stage, on CSQA with a TinyLlama 1.1B model (see Table 4 (Appendix) for data statistics). Let us consider an accuracy threshold (ϵ) 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. This example is for illustration only and does not imply that ϵ is fixed to 1%; in practice, we use multiple ϵ values.

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

(2) *interaction between stages*: the profile ensures that interactions between stages are explicitly considered during the budget search. We perform stage-wise budget allocation sequentially from the longest stage (Stage 2) to the shortest (Stage 3). When searching for Stage 2's layer budget, the other stages use full layers. Once Stage 2's budget is fixed, Stage 1's budget is searched while Stage 2 operates under skipping. Finally, Stage 3's budget is determined while both Stage 1 and Stage 2 use their selected skipping policies. This is different from an "independent search", where Stage n's budget would be determined while all other stages remain 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 offline search overhead. Finding the optimal setting re-

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

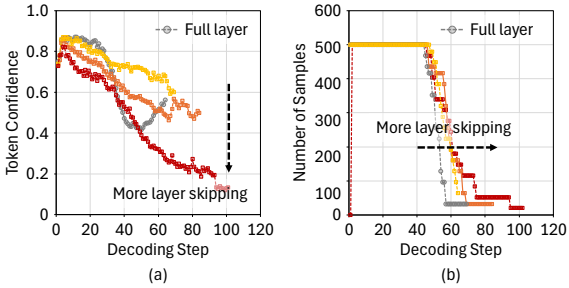

Figure 3: **Confidence of Output Tokens.** (a) Average token-level confidence across decoding steps under different layer-skipping levels. (b) Number of remaining samples during decoding (not yet exited), where more aggressive layer skipping leads to earlier exits due to confidence decay. Results are 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.

quires exploring combinations of skipped layers and reasoning stages, but this one-time search is performed on a single GPU and typically finishes within a few hours, *e.g.*, 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. The selected configuration is then reused for all inference, incurring no additional cost.

3.4 Step 3: Generation Early Exit

While Step 2 determines a balanced layer budget, it leaves long outputs unchanged. To further improve speedup, we reduce redundant output tokens to recover the latency gains from layer skipping. Our hypothesis is that tokens induced by skipping contribute little to the final reasoning outcome.

Extra Tokens may not be Useful. Figure 3 shows how token confidence evolves during decoding under four skip configurations, where confidence is defined as the maximum logit of each generated token. Confidence trajectories differ with layer skipping: skipped models show steadily decreasing confidence, whereas the full-layer model partially recovers in later steps. However, a common pattern remains that high confidence (>0.5) mostly occurs in early decoding steps. This becomes clearer as more layers are skipped; for example, the red curve

Algorithm 1 LiteStage: Stage-wise Layer Skipping with Early Exit

Input: Multi-stage pipeline $\{S_1, S_2, S_3\}$, model M, validation set D_{val}, accuracy threshold ε, exit params (n, τ)
Output: Skip configuration $\{L_1, L_2, L_3\}$
Step 1: Layer-importance estimation
for each stage S_t **do**
  Estimate sub-layer redundancy on D_{val}
  Rank sub-layers by importance
end for
Step 2: Stage-wise budget search
Measure full-depth accuracy A_0, set $A_{\text{target}} = A_0 - \varepsilon$
for stages in longest $\rightarrow$ shortest order **do**
  for candidate skip k **do**
    Apply skip to current stage only
    Evaluate accuracy $A(k)$ and latency $T(k)$ on D_{val}
  end for
  Select fastest k s.t. $A(k) \geq A_{\text{target}}$
end for
Step 3: Online confidence-based early exit
while decoding **do**
  if moving-average confidence $< \tau$ for n tokens **then**
    stop generation
  end if
end while
return $\{L_1, L_2, L_3\}$

(20-layer skip) consistently exhibits lower confidence and longer generation than the yellow curve (10-layer skip). Based on this observation, we apply confidence-based early exit in generation, assuming that terminating these low-confidence tokens has little impact on accuracy.

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 to stop generation. Because confidence can fluctuate across decoding steps, relying on a single token may cause premature termination. To stabilize the decision, we maintain a *confidence cache* storing the most recent n confidence values, and from the n-th step, compute the mean confidence μ_{Conf} to compare with a threshold. We set n=5 and the confidence threshold to 0.5.

We also incorporate generation early exit into the

search process (Step 2) so that the accuracy-latency profile reflects real evaluation conditions. Without generation early exit, the effective search space can shrink when generation length becomes highly sensitive to skipping, causing latency gains to vanish beyond few skip levels. Generation early exit mitigates this, expanding the feasible search space and allowing more valid layer-skipping configurations.

We provide an algorithm-style description of LiteStage in Algorithm 1 to improve clarity and facilitate understanding of the overall procedure.

4 Experimental Results

Dataset and Implementation. 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 trained on TinyThinker's reasoning paths before applying layer skipping. During validation and testing, reasoning paths are not accessible. Our method is primarily evaluated 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. 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].

4.1 Comparison with Baselines

Figure 4 compares LiteStage with three baselines on OBQA, CSQA, and StrategyQA. Notably, LiteStage preserves reasoning quality even at high acceleration ratios, where prior methods often degrade significantly. Baselines are evaluated by progressively increasing the number of skipped layers until speedup saturates. LiteStage consistently outperforms them, extending the achievable performance boundary. For example, in OBQA (Figure 4(a)), the strongest baseline AdaSkip matches our accuracy up to 1.10× speedup, but quickly collapses afterward, dropping to nearly 0% accuracy, while LiteStage remains stable, achieving 1.49× speedup with 60.6% accuracy. The achievable speedup depends on the user-specified ϵ, which controls the accuracy–efficiency trade-off. In practice, we tune ϵ on a validation set (*e.g.*, 1%–3% for Qwen2.5-0.5B), and select the configuration that best matches the accuracy-latency balance.

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

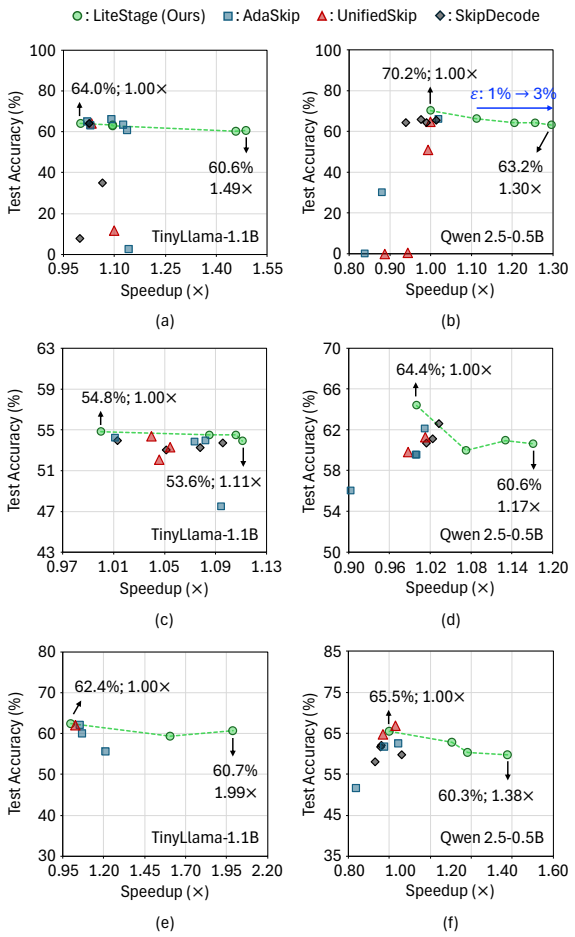

Figure 4: **Comparison with Baselines.** Accuracy–speedup trade-offs on the OBQA (row-1), CSQA (row-2), and StrategyQA (row-3) datasets, respectively. Results are experimented using TinyLlama-1.1B (col-1) and Qwen2.5-0.5B (col-2) models. The performance of the full-layer model is marked in the upper-left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

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

4.2 Benefits of Non-uniform Layer Budget

Protection from Accuracy Collapse. Table 3 presents layer budgets assigned by LiteStage for

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

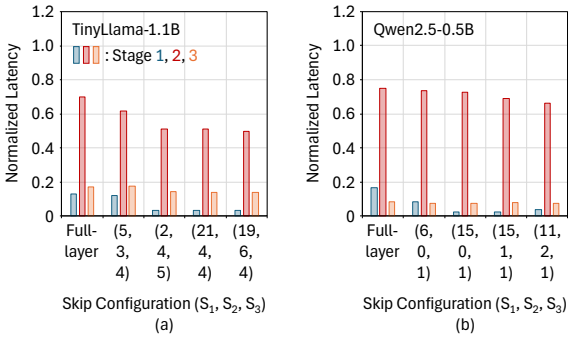

Figure 5: **Stage-wise Latency.** (a)-(b) show the normalized latency of each stage in TinyLlama-1.1B and Qwen2.5-0.5B, respectively, on OBQA. In the full-layer configuration, the sum of stage-wise latencies is normalized to 1.0. S_1, S_2, S_3 on the x-axis denotes the number of skipped layers at Stages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

each stage. LiteStage consistently skips no more than five (TinyLlama) and three (Qwen) layers in Stage 3, reflecting its ability to limit compression in sensitive stages while accelerating more robust ones. This is because Stage 3 is notably more sensitive to skipping and often determines the overall accuracy (see Figure 15-20, Appendix). By allocating fewer skipped layers to such stages, LiteStage prevents accuracy collapse, demonstrating the effectiveness of stage-wise non-uniform skipping.

Layer Budget Distribution. Beyond protecting accuracy-sensitive stages, LiteStage adapts the layer budget across datasets. As shown in Figures 1(b)-(c), Stage 1 is relatively robust to skipping, whereas Stage 2 dominates latency. Accordingly, when accuracy must be preserved, LiteStage allocates more skipping to Stage 1, as Stage 2 has little room for further skipping under the accuracy constraint. For example, on OBQA (Table 3), most variation appears in the Stage 1 budget, increasing from 5→21 (TinyLlama) and 6→15 (Qwen).

As further speedup is required, LiteStage shifts

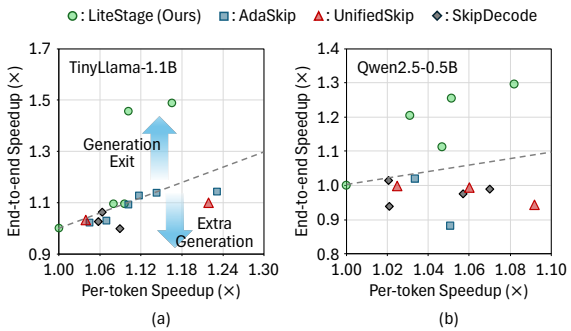

Figure 6: **Per-token and End-to-End Speedup.** (a)-(b) show the per-token and end-to-end speedup of TinyLlama-1.1B and Qwen2.5-0.5B, respectively, on the OBQA dataset. The full-layer baseline is normalized to a per-token and end-to-end speedup of 1.0. The gray dotted line denotes the $y{=}x$ reference.

budget to Stage 2 while reducing Stage 1. In the most aggressive OBQA setting (row 3 in Table 3), budgets change from 21, 4→19, 6 (TinyLlama) and 15, 1→11, 2 (Qwen) for Stages 1 and 2. Thus, LiteStage first reduces Stage 1 latency and then Stage 2 when larger reductions are needed (Figure 5). Similar trend appears on StrategyQA, while on CSQA Stage 2 is more sensitive, so larger speedups come from further accelerating Stage 1.

4.3 Benefits of Generation Early Exit

Per-token vs End-to-End Speedup. Extended generation hinders end-to-end speedup, even though layer skipping reduces per-token latency. Figure 6 compares per-token and end-to-end speedup, where the ideal case follows the dotted line ($y{=}x$), *i.e.*, per-token speedup translates to end-to-end speedup. For the baselines, per-token speedup keeps increasing, but end-to-end speedup saturates or even drops, especially under aggressive skipping. In contrast, with generation early exit, LiteStage improves end-to-end speedup, often exceeding per-token speedup and fully benefiting from layer skipping.

With and Without Generation Early Exit. Figure 7 compares layer skipping with and without generation early exit under the same skip budgets, revealing two regimes. (1) *Mild skipping*. When skipping is small (up to 12 sub-layers for TinyLlama and 2 for Qwen), accuracy, latency, and decoding steps change little, as output confidence remains high and decoding length stays close to the full-layer baseline. In this regime, layer skipping alone provides speedup, so early exit gives limited benefit. (2) *Aggressive skipping*. With larger skips, decoding length grows without early exit, while early exit shortens the sequence and reduces end-

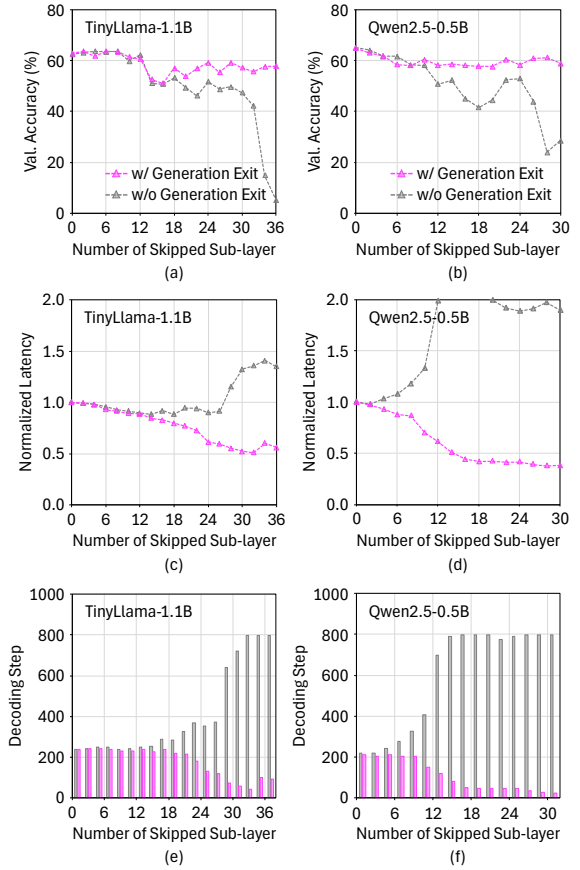

Figure 7: **Layer Skipping w/ and w/o Generation Exit.** (a)–(b) validation accuracy, (c)–(d) normalized end-to-end latency, and (e)–(f) decoding steps as a function of the number of skipped sub-layers for TinyLlama-1.1B and Qwen2.5-0.5B on the OBQA dataset. Results apply layer skipping only at Stage 2.

to-end latency (Figures 7(c)-(f)). The effect appears once decoding steps increase, beyond roughly 16 skipped sub-layers for TinyLlama and 4 for Qwen, indicating that early exit is most beneficial in the high-acceleration regime.

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 increase end-to-end latency but can also harm final predictions.

4.4 Evaluation in Other Models

Larger Models (4B/7B). We further evaluate Qwen3-4B-Instruct-2507 and Qwen2.5-7B-Instruct to verify that our findings generalize beyond small models (<1.1B). As shown in Figure 8, LiteStage consistently outperforms the full-layer baseline and AdaSkip on OBQA, CSQA, and StrategyQA, achieving higher end-to-end speedup while

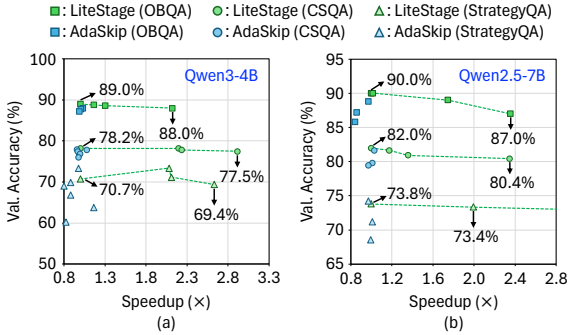

Figure 8: **Evaluation in 4B and 7B models.** (a) and (b) show the accuracy-speedup trade-off for Qwen3-4B and Qwen2.5-7B, respectively. Speedup is normalized by the end-to-end latency of the full-layer baseline.

maintaining similar accuracy. The gain is more pronounced in these models, where combining layer skipping with generation early exit yields substantial latency reduction without harming answer quality. Detailed results are provided in Appendix E.

Deep Reasoning Models. While LiteStage is designed for short multi-stage reasoning, we also examine whether its strategies extend to *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 to 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.

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 for truly efficient multi-stage reasoning.

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.

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. 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*.

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 ⟨/Think⟩ 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). We include additional analyses on deep reasoning tasks to examine the behavior of LiteStage under long-horizon generation settings (Appendix D). We further provide additional evaluations on larger models to verify scalability beyond small LMs, as well as experiments under mild distribution shifts to examine the robustness of the validation-selected skip configuration (Appendix E, Appendix F).

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 on the complete OBQA, CSQA, and StrategyQA datasets 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}. Training uses the AdamW optimizer (default in `SFTTrainer`) with weight decay 0.0, β_1=0.9, β_2=0.999, and ϵ=10^{-8}. Evaluation follows TinyThinker's self-consistency (*i.e.*, majority voting) protocol with 10 iterations. All experiments are conducted on a single NVIDIA A6000 GPU (48GB).

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

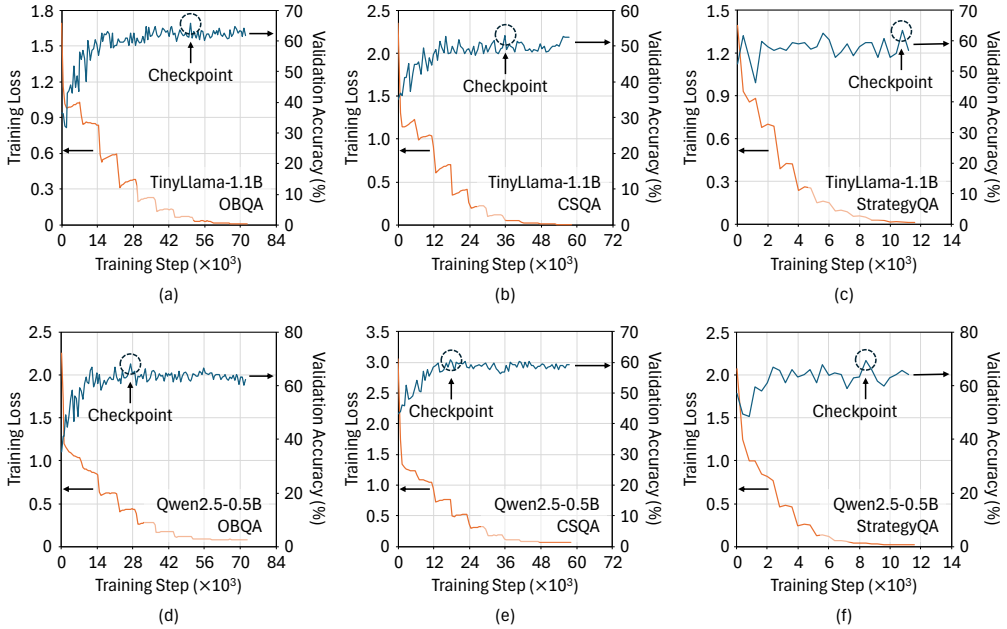

Figure 9: **Training Dynamics.** Training loss and validation accuracy over training steps for TinyLlama-1.1B and Qwen2.5-0.5B on OBQA, CSQA, and StrategyQA. The top row shows results for TinyLlama-1.1B, and the bottom row shows results for Qwen2.5-0.5B. In each plot, training loss is shown on the left y-axis and validation accuracy on the right y-axis. Circles denote the selected checkpoints used for evaluation.

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, 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 9 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 14 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 14(a), while Figures 14(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

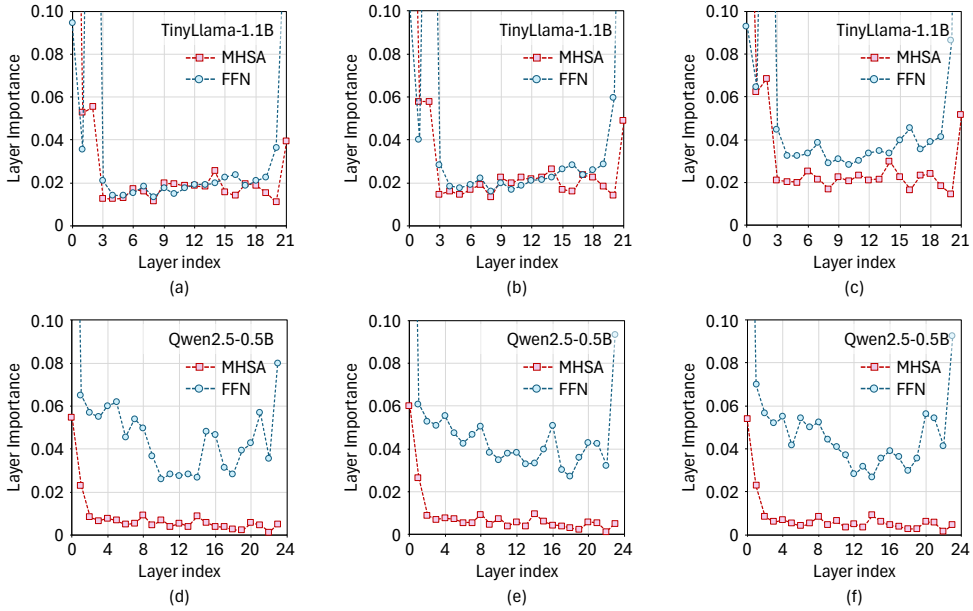

Figure 10: **Layer Importance.** Sub-layer-wise importance estimates for TinyLlama-1.1B (a)–(c) and Qwen2.5-0.5B (d)–(f) across OBQA, CSQA, and StrategyQA. Importance is computed using cosine similarity between sub-layer inputs and outputs, separately for multi-head self-attention (MHSA) and feed-forward network (FFN) layers.

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
```

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 10 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 15-17 report results for TinyLlama-1.1B, and Figures 18-20 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 15(b) and 18(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 15(a), skipping 18 sub-layers achieves higher validation accuracy than skipping 16 sub-layers, while in Figure 15(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 Live-

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

CodeBench 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 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

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

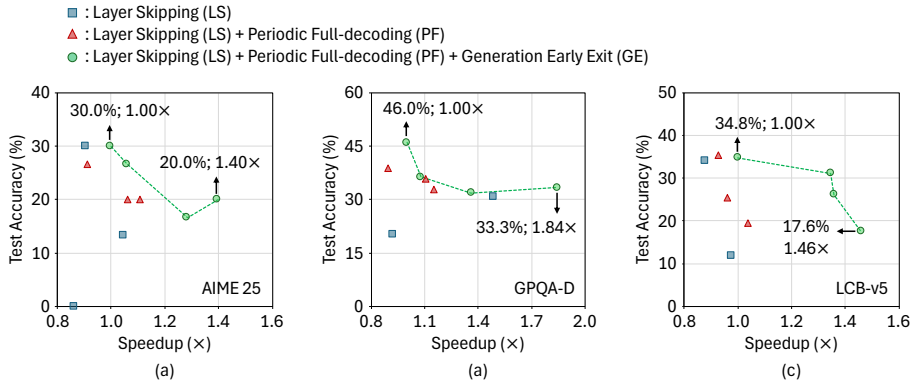

Figure 11: **Deep Reasoning Performance.** Test accuracy and end-to-end speedup are compared between the three configurations (LS, LS+PF, and LS+PF+GE) on the three benchmarks, (a) AIME 25, (b) GPQA-D, and (c) LCB-v5. Results are experimented using Qwen3-1.7B.

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× (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 generation 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 11 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 Evaluation in Larger Models

To evaluate whether LiteStage generalizes beyond small models (<1.1B), we additionally test Qwen3-4B-Instruct-2507 and Qwen2.5-7B-Instruct, following exactly the same multi-stage reasoning workflow used for TinyLlama-1.1B and Qwen2.5-0.5B.

Across OBQA, CSQA, and StrategyQA, LiteStage consistently provides larger end-to-end speedups than both the full-layer baseline and AdaSkip while maintaining comparable accuracy. Tables 6–11 report full results including accuracy, end-to-end speedup, and decoding steps.

Results on 7B. AdaSkip again shows negligible or even negative speedups due to increased generation length (*e.g.*, 0.842×-0.974× on OBQA, 0.972×-1.032× on CSQA, and 0.466×-1.012× on StrategyQA). In contrast, LiteStage achieves substantial acceleration (1.016×-2.356× on OBQA, 1.178×-2.347× on CSQA, and 1.995×-4.444× on StrategyQA) while preserving accuracy close to

Table 6: Layer-skipping results in Qwen2.5-7B on OBQA. "# Skip (Stg1 / Stg2 / Stg3)" denote the number of skipped layers for Stage 1, Stage 2, and Stage 3. "Acc." and "Dec. Steps" represent the validation accuracy and the total number of decoding steps.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	90.00%	1.000×	285.2
AdaSkip	2 / 2 / 2	88.80%	0.974×	298.5
AdaSkip	4 / 4 / 4	87.20%	0.860×	354.0
AdaSkip	6 / 6 / 6	85.80%	0.842×	346.0
LiteStage	0 / 2 / 10	90.00%	1.016×	285.3
LiteStage	0 / 30 / 2	89.00%	1.748×	81.8
LiteStage	22 / 30 / 10	87.00%	2.356×	39.6

Table 7: Layer-skipping results in Qwen2.5-7B on CSQA.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	82.98%	1.000×	279.4
AdaSkip	2 / 2 / 2	81.65%	1.032×	283.7
AdaSkip	4 / 4 / 4	79.77%	1.012×	301.8
AdaSkip	6 / 6 / 6	79.44%	0.972×	312.7
LiteStage	0 / 0 / 2	81.65%	1.178×	236.0
LiteStage	26 / 0 / 2	80.92%	1.360×	208.4
LiteStage	28 / 24 / 4	80.42%	2.347×	57.0

Table 8: Layer-skipping results in Qwen2.5-7B on StrategyQA.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	73.80%	1.000×	248.4
AdaSkip	2 / 2 / 2	68.56%	0.996×	247.6
AdaSkip	4 / 4 / 4	71.18%	1.012×	264.3
AdaSkip	6 / 6 / 6	74.24%	0.972×	277.9
AdaSkip	8 / 8 / 8	72.49%	0.683×	324.5
AdaSkip	10 / 10 / 10	63.76%	0.525×	499.0
AdaSkip	12 / 12 / 12	67.69%	0.466×	587.1
LiteStage	8 / 36 / 12	73.36%	1.995×	94.9
LiteStage	28 / 36 / 0	72.49%	4.343×	43.0
LiteStage	32 / 36 / 16	72.95%	4.444×	41.1

Table 9: Layer-skipping results in Qwen3-4B on OBQA.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	89.00%	1.000×	276.6
AdaSkip	2 / 2 / 2	88.00%	1.027×	284.9
AdaSkip	4 / 4 / 4	87.80%	1.004×	288.3
AdaSkip	6 / 6 / 6	87.20%	0.987×	308.8
LiteStage	2 / 8 / 12	88.80%	1.165×	255.4
LiteStage	0 / 22 / 2	88.60%	1.303×	242.0
LiteStage	0 / 54 / 4	88.00%	2.118×	76.3

Table 10: Layer-skipping results in Qwen3-4B on CSQA.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	78.21%	1.000×	283.8
AdaSkip	2 / 2 / 2	76.74%	0.985×	285.6
AdaSkip	4 / 4 / 4	77.97%	0.958×	298.6
AdaSkip	6 / 6 / 6	77.48%	0.969×	287.8
AdaSkip	8 / 8 / 8	76.90%	0.996×	295.4
AdaSkip	10 / 10 / 10	77.89%	1.076×	287.8
AdaSkip	12 / 12 / 12	76.00%	0.977×	316.3
LiteStage	0 / 52 / 0	78.21%	2.197×	69.0
LiteStage	0 / 52 / 2	77.81%	2.232×	67.5
LiteStage	42 / 54 / 6	77.47%	2.911×	38.7

Table 11: Layer-skipping results in Qwen3-4B on StrategyQA.

Model	# Skip (Stg1 / Stg2 / Stg3)	Acc.	Speedup	Dec. Steps
Full-layer	0 / 0 / 0	70.74%	1.000×	232.4
AdaSkip	2 / 2 / 2	69.87%	0.880×	265.9
AdaSkip	4 / 4 / 4	69.00%	0.800×	254.0
AdaSkip	6 / 6 / 6	73.36%	0.975×	249.6
AdaSkip	8 / 8 / 8	66.81%	0.879×	260.7
AdaSkip	10 / 10 / 10	60.26%	0.823×	255.3
AdaSkip	12 / 12 / 12	63.76%	1.161×	231.9
LiteStage	0 / 36 / 0	73.36%	2.080×	85.0
LiteStage	0 / 36 / 14	71.18%	2.107×	85.0
LiteStage	26 / 36 / 0	69.40%	2.629×	67.1

the full-layer model.

Results on 4B. A similar trend is observed in Qwen3-4B. AdaSkip provides only marginal speedup (0.987×-1.027× on OBQA, 0.958×-1.076× on CSQA, and 0.800×-1.161× on StrategyQA), whereas LiteStage achieves significantly larger gains (1.165×-2.118× on OBQA, 2.197×-2.911× on CSQA, and 2.080×-2.629× on StrategyQA) with comparable accuracy.

Effect of Model Scale. Larger models show stronger robustness to layer skipping and generation early exit. Under aggressive skipping in Stage 2, LiteStage often produces lower output confidence earlier, leading to shorter decoding sequences (<100 steps), yet still maintaining accuracy close to the full-layer model. This suggests that larger models can produce reliable answers with fewer decoding steps once confident tokens are generated.

Overall, these results demonstrate that the effectiveness of LiteStage is not limited to small models. Even for larger 4B and 7B models, the same stage-wise skip strategy continues to provide consistent speedups with minimal accuracy degradation, indicating that LiteStage scales well with model size and can be applied as a general acceleration framework across different model scales.

F Evaluation under Distribution Shifts

Setups. To examine whether the validation-selected skip configuration generalizes across mild distribution shifts, we construct additional paraphrased test splits for OBQA, CSQA, and StrategyQA using an external LLM. This experiment tests whether the offline search configuration transfers beyond the original validation/test distribution.

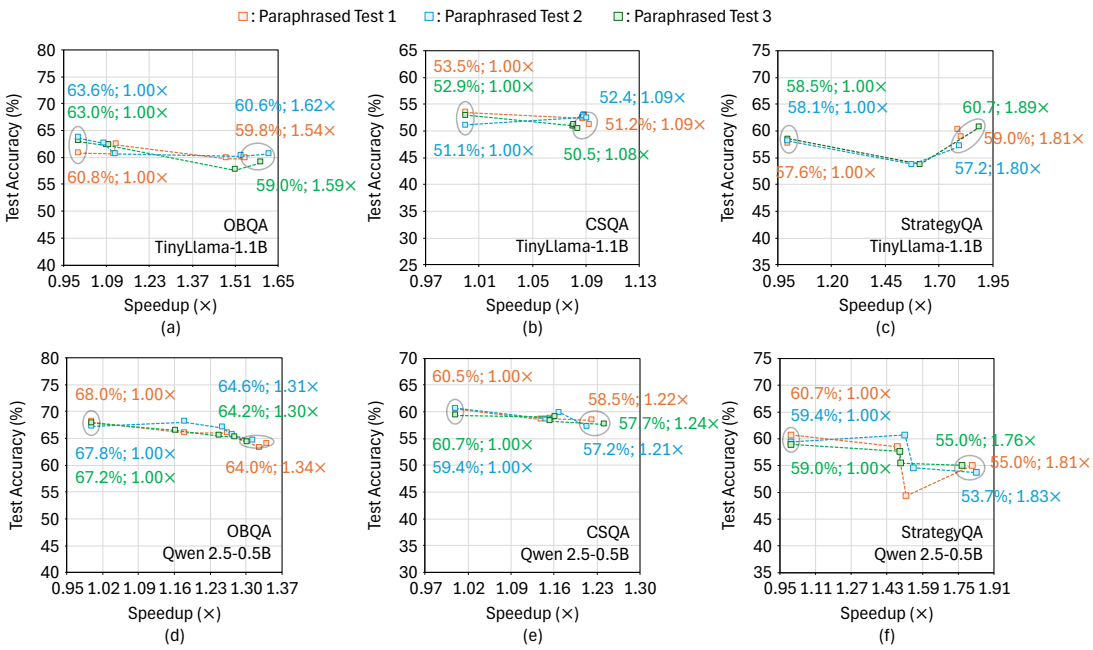

Figure 12: **Evaluation under Mild Distribution Shifts.** For each dataset, we evaluate three paraphrased versions of the original test set using the skip configuration selected on the original validation set without additional tuning. Results are shown for TinyLlama-1.1B and Qwen2.5-0.5B. Across all datasets and paraphrased splits, LiteStage maintains similar accuracy–speedup trade-offs, demonstrating that the validation-selected configuration remains effective under mild distribution shifts.

Paraphrased Test Split Generation. For each dataset, we generate three additional paraphrased versions of the original test set using GPT-4.1-mini. The paraphrasing is applied only to the question text, while the answer choices and ground-truth labels remain unchanged. We instruct the model to produce mild paraphrases that preserve the exact semantics of the original question.

The system prompt used for paraphrasing is shown below:

```
You rewrite dataset questions.

Task:
Generate 3 mild paraphrases of the question
text.

Rules:
- Keep meaning exactly the same
- Do NOT change answer meaning
- Do NOT include choices
- Do NOT include explanation
- Do NOT number them
- Output exactly 3 lines
- Each line must be one paraphrase
```

The user prompt template is:

```
Rewrite the following question into 3
different mild paraphrases.

Question:
{question}
```

Each original test sample is therefore expanded into three paraphrased variants, resulting in three additional test splits per dataset.

Construction of Evaluation Splits. For each dataset (OBQA, CSQA, StrategyQA), we create original test split and paraphrased split 1, 2, and 3. Each paraphrased split contains the same number of samples as the original test set, with identical answer choices and labels, but with paraphrased question text. Examples of the generated paraphrases are shown in Figure 13. As illustrated, the paraphrases modify wording while keeping the semantics and answer unchanged. For instance, the original question:

```
A person wants to start saving money so
that they can afford a nice vacation at
the end of the year. After looking over
their budget and expenses, they decide
the best way to save money is to ...
```

is rewritten into semantically equivalent variants such as:

```
A person plans to save money for a
nice vacation at the end of the year
and, after reviewing their budget and
expenses, concludes the best method to
save is to ...
```

while preserving the same answer.

Generalization under Mild Distribution Shifts. Figure 12 shows the results on three paraphrased test splits for OBQA, CSQA, and StrategyQA, where the skip configuration selected on the original validation set is applied without re-tuning. Overall, the accuracy–speedup trade-offs remain similar across paraphrased splits for both TinyLlama-1.1B and Qwen2.5-0.5B, indicating that the offline-selected configuration transfers well under mild distribution shifts.

On OBQA, even in the highest-speedup regime, the results remain close across paraphrased splits.For example, TinyLlama-1.1B achieves 60.6% at 1.62×, 59.8% at 1.54×, and 59.0% at 1.59×, while Qwen2.5-0.5B shows 64.6% at 1.31×, 64.0% at 1.34×, and 64.2% at 1.30×. CSQA shows similar stability, where TinyLlama-1.1B obtains 52.4% at 1.09×, 51.2% at 1.09×, and 50.5% at 1.08×, and Qwen2.5-0.5B achieves 57.2% at 1.21×, 58.5% at 1.22×, and 57.7% at 1.24×.

We observe relatively larger variation on StrategyQA, but the overall trend is still consistent. At the highest speedup point, TinyLlama-1.1B shows 60.1% at 1.89×, 59.0% at 1.81×, and 57.2% at 1.80×, while Qwen2.5-0.5B reports 55.0% at 1.76×, 55.0% at 1.81×, and 53.7% at 1.83×. This larger fluctuation is expected since StrategyQA has only two answer options, making the accuracy more sensitive to small prediction changes compared to multi-choice datasets.

Overall, these results suggest that under the domain-conditioned multi-stage reasoning setting considered in this work, the validation-selected skip configuration generalizes across splits and remains effective under mild distribution shifts, supporting the practicality of the offline search procedure.

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.

OBQA Test Example

”original question": "A person wants to start saving money so that they can afford a nice vacation at the end of the year. After looking over their budget and expenses, they decide the best way to save money is to\n(A) make more phone calls (B) quit eating lunch out (C) buy less with monopoly money (D) have lunch with friends\nRecall:"

”paraphrased question 1": "A person plans to save money for a nice vacation at the end of the year and, after reviewing their budget and expenses, concludes the best method to save is to\n(A) make more phone calls (B) quit eating lunch out (C) buy less with monopoly money (D) have lunch with friends\nRecall:"

”paraphrased question 2": "A person wants to set aside money for a pleasant vacation at year's end and, after examining their budget and expenses, determines the best approach to saving is to\n(A) make more phone calls (B) quit eating lunch out (C) buy less with monopoly money (D) have lunch with friends\nRecall:"

”paraphrased question 3": "A person aims to save funds for an enjoyable vacation at the end of the year and, after analyzing their budget and expenses, decides the most effective way to save is to\n(A) make more phone calls (B) quit eating lunch out (C) buy less with monopoly money (D) have lunch with friends\nRecall:"

"GT": "B"

(a)

CSQA Test Example

”original question": "A revolving door is convenient for two direction travel, but it also serves as a security measure at a what?\n(A) bank (B) library (C) department store (D) mall (E) new york\nRecall:"

”paraphrased question 1": "A revolving door is useful for two-way traffic, but it can also act as a security feature at what kind of place?\n(A) bank (B) library (C) department store (D) mall (E) new york\nRecall:"

”paraphrased question 2": "While a revolving door allows travel in both directions, it also functions as a security measure at what location?\n(A) bank (B) library (C) department store (D) mall (E) new york\nRecall:"

”paraphrased question 3": "A revolving door facilitates two-direction travel and also serves as a security device at what type of facility?\n(A) bank (B) library (C) department store (D) mall (E) new york\nRecall:”

"GT": "A"

(b)

StrategyQA Test Example

”original question": "Is Alistair Darling in favor of Scottish independence?\n(A) yes (B) no\nRecall:"

”paraphrased question 1": "Does Alistair Darling support Scottish independence?\n(A) yes (B) no\nRecall:"

”paraphrased question 2": "Is Alistair Darling opposed or in favor of Scottish independence?\n(A) yes (B) no\nRecall:"

”paraphrased question 3": "What is Alistair Darling's stance on Scottish independence?\n(A) yes (B) no\nRecall:"

"GT": "B"

(c)

Figure 13: **Example of Paraphrased Test Data.** For each original question in (a) OBQA, (b) CSQA, and (c) StrategyQA, the paraphrases modify only the wording of the question while preserving its semantics, allowing us to test whether the validation-selected skip configuration generalizes beyond the original data distribution.

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 14: **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.

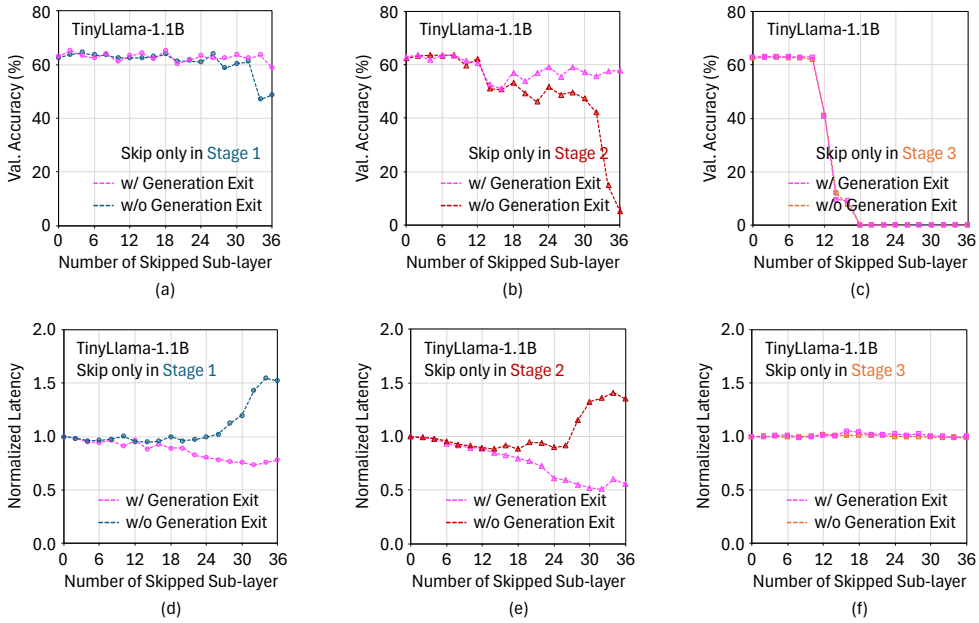

Figure 15: **Ablation: TinyLlama-1.1B on OBQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on OBQA, comparing configurations with and without generation early exit.

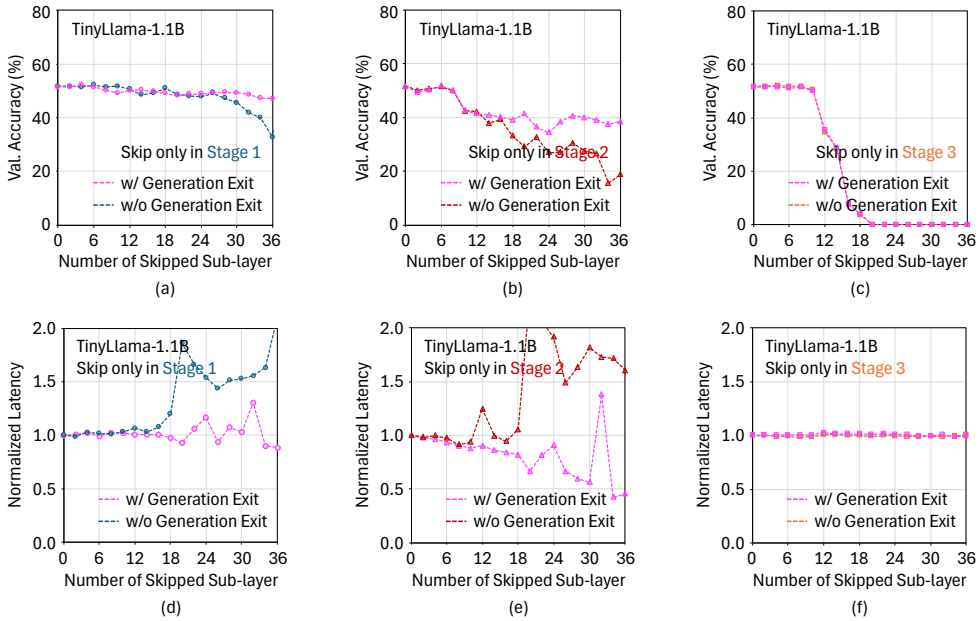

Figure 16: **Ablation: TinyLlama-1.1B on CSQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on CSQA, comparing configurations with and without generation early exit.

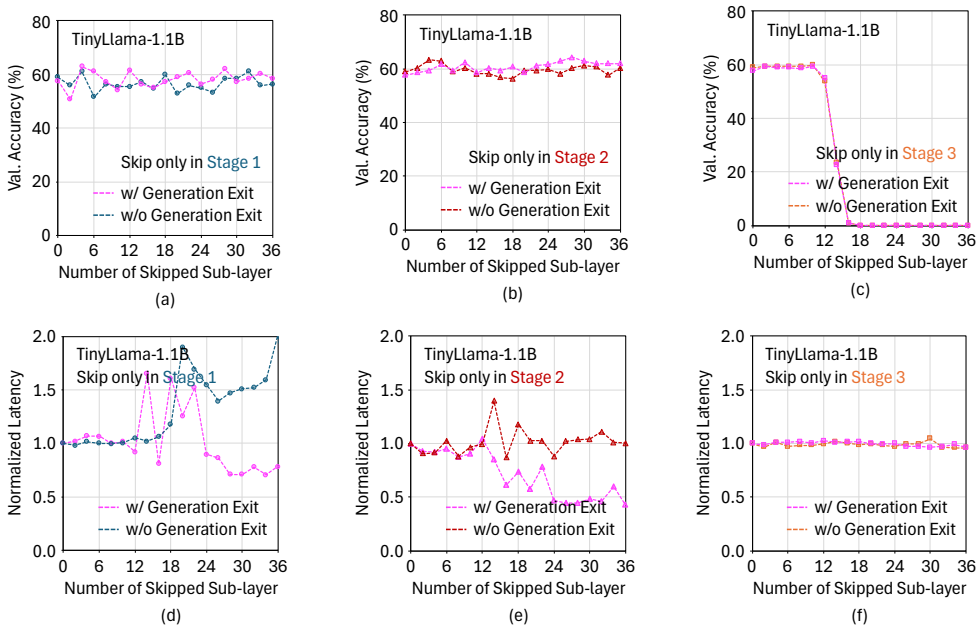

Figure 17: **Ablation: TinyLlama-1.1B on StrategyQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for TinyLlama-1.1B on StrategyQA, comparing configurations with and without generation early exit.

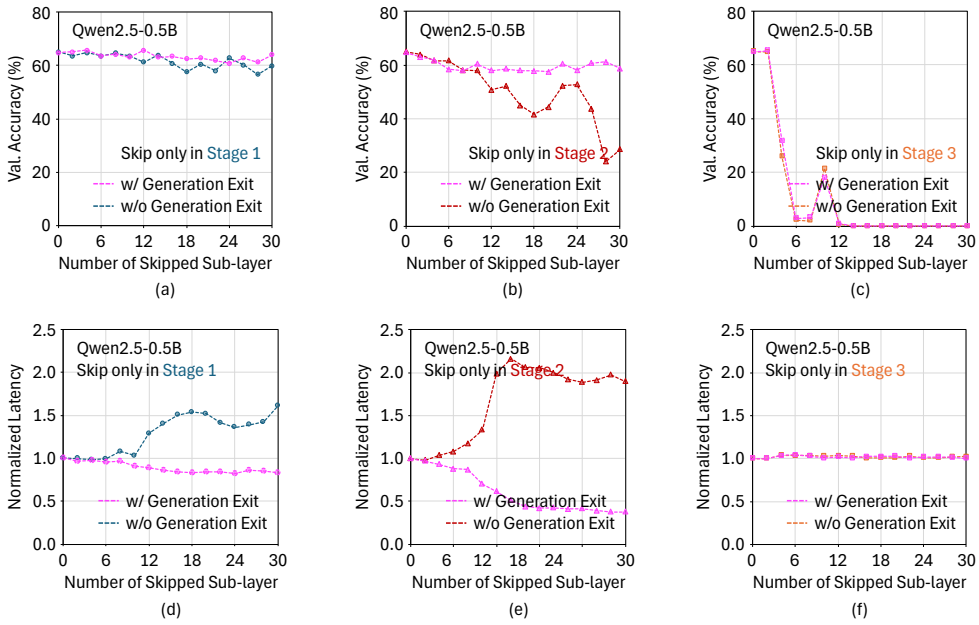

Figure 18: **Ablation: Qwen2.5-0.5B on OBQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on OBQA, comparing configurations with and without generation early exit.

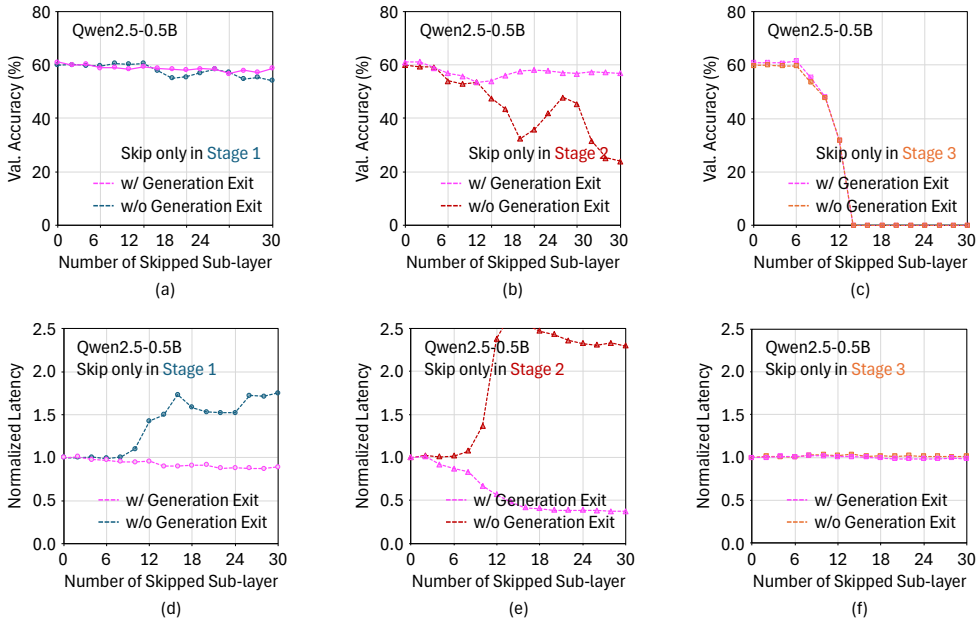

Figure 19: **Ablation: Qwen2.5-0.5B on CSQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on CSQA, comparing configurations with and without generation early exit.

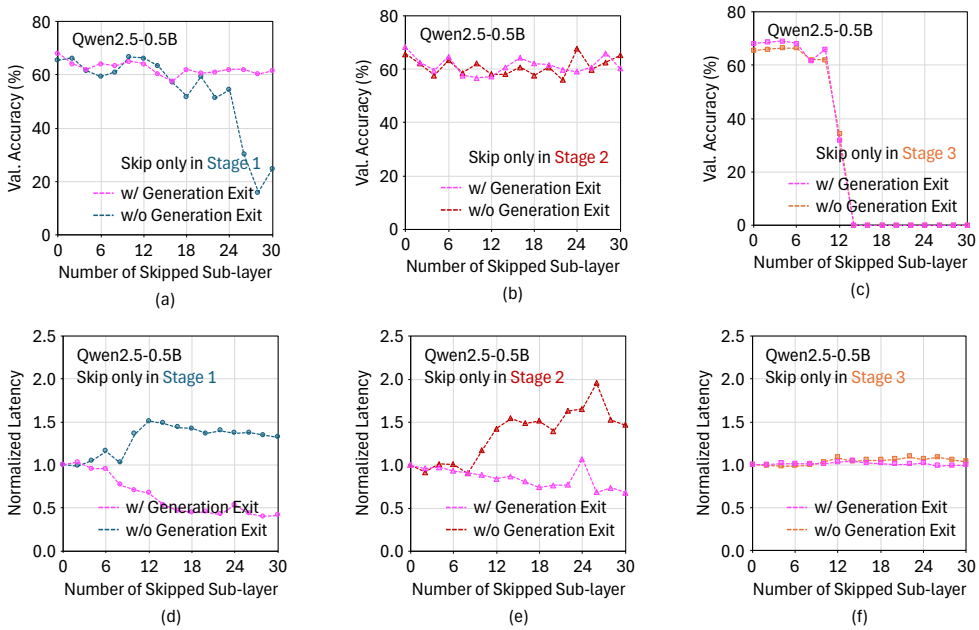

Figure 20: **Ablation: Qwen2.5-0.5B on StrategyQA.** Validation accuracy (top row) and normalized latency (bottom row) as a function of the number of skipped sub-layers when applying layer skipping to a single reasoning stage at a time (Stage 1, Stage 2, or Stage 3). Results are shown for Qwen2.5-0.5B on StrategyQA, comparing configurations with and without generation early exit.