AVERT: AVERAGE-DEPTH VARIABLE EXECUTION VIA REQUEST-LOCAL TOKEN ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Layer skipping can reduce large language model inference costs, but a request budget does not specify how to allocate layer execution across generated tokens. We introduce AVERT, which treats a continuous request budget as a target for mean executed depth. At each dynamic layer, a request-local router uses the current hidden state and that request's budget to decide whether to execute or skip the Transformer block, allowing token depths to vary without coupling decisions across requests. Skipped tokens bypass the whole Transformer block, omit their layer-local key/value (K/V) entries, and use a lightweight residual-repair adapter. A single checkpoint serves requests with different budgets in the same batch. Across three LLaMA backbones, six quality metrics, and 13 requested budgets, AVERT is best or tied for best in 206 of 234 comparisons. Across homogeneous and mixed-budget batches and five off-grid targets, the mean executed-layer fraction for each target differs from the requested budget by at most 1.15 percentage points. When splitting requests by budget leaves batches underfilled, mixing budgets reduces decode time per output token (TPOT) by 4.3–21.2%.

1 INTRODUCTION

Large language model (LLM) serving systems batch requests to improve utilization (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024). Some systems place prefill and decoding on separate GPU resources, making decode optimization independently useful (Zhong et al., 2024). A request may also specify how much inference compute it can use. Such a budget fixes an aggregate target, but leaves the allocation across generated tokens open: some tokens may need more layers than others. We study this question for *interior layer skipping*, where a token can bypass a Transformer block and still execute later blocks.

Existing budgeted methods impose different restrictions. In D-LLM, Jiang et al. (2024) route tokens progressively through the layers, but train a separate model for each budget and penalize each token for deviating from that budget's target depth. In BUDDY, Zhou et al. (2026) accept a request-specific continuous-valued budget from one model; its explicit-budget path obtains the total executed-layer count by rounding the budget-scaled layer count. The selected layer identities can change across tokens, but their executed counts cannot vary at a fixed budget. In GateSkip, Laitenberger et al. (2026) use one model across budgets, yet threshold attention and feed-forward network (FFN) sublayer gates against the current batch-wide token pool. Its decisions for a request therefore depend on the other requests in that batch.

AVERT treats the request budget Ω_b as a target for the *mean executed depth over that request's routed decode positions*. Its objective allows one token to execute more layers and another to execute fewer while keeping their request average near Ω_b. At each dynamic layer, a router reads the current token state and its request's budget, without ranking tokens from different requests against each other. Selected tokens execute the Transformer block; skipped tokens take a residual repair path and do not write to that layer's compact K/V cache. Because routing is conditioned on each request's Ω_b and the checkpoint is trained across the full budget interval, the same checkpoint can decode requests with different budgets in one batch. The compensation adapter follows the skip path in FlexiDepth as described by Luo et al. (2025a); physical K/V omission builds on earlier skipping with K/V eviction (Jiang et al., 2024).

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

On three LLaMA backbones, six quality metrics, and 13 requested budgets, AVERT is best or tied for best in 206 of the 234 comparisons. On five off-grid targets, the maximum absolute difference between each requested target and the mean executed fraction for requests assigned that target is 0.90 percentage points; the largest difference in heterogeneous-budget batches is 1.15 points. In controlled workloads, mixing budgets has comparable decode TPOT when batches separated by budget remain full; it lowers TPOT by 4.3–21.2% when separation creates underfilled batches.

Our contributions are (i) a request-average objective that allows token depths to vary within a request, (ii) a request-local physical decode design with compact per-layer caches and residual repair, and (iii) a systematic evaluation of quality, budget tracking, and heterogeneous-budget batching efficiency.

2 Related Work

Budgeted conditional computation. Several conditional computation methods take a resource budget as input. In CCT, Bapna et al. (2020) associate input symbols with budgets and impose a cost penalty aggregated over a batch, so examples can compensate for one another. In L2A, Chen et al. (2026) feed a continuous resource signal to hidden-state-dependent layer and head gates and define expected compute as a mini-batch average. In SOL, Akhauri & Abdelfattah (2026) choose among attention sparsity, FFN-channel pruning, and quantization actions under an average budget over short decode episodes. In contrast, AVERT matches the mean executed-layer fraction separately for each request before averaging the resulting errors across a batch.

Shared layer skipping. Shared or structured policies reuse a path or schedule across multiple tokens rather than choosing an independent mask for every layer–token pair. Fan et al. (2020) train reduced-depth subnetworks with LayerDrop. Del Corro et al. (2023) and Zhu et al. (2025) use position- or sequence-length-dependent schedules in SkipDecode and Hierarchical Skip Decoding, respectively. In Unified Layer Skipping, Liu et al. (2024) select a balanced input-independent pattern for a target speedup. In FiRST, Jain et al. (2025) choose a path from the input and reuse it during decoding, whereas in AdaSkip, He et al. (2025b) derive a context-dependent set of attention and FFN sublayers to bypass. In DASH, Yang et al. (2025) model input-aware layer execution states, including skipping and quantized computation. Early-exit and self-speculative methods instead change where or how a token is produced rather than selecting arbitrary interior blocks (Bae et al., 2023; Fan et al., 2025; Zhang et al., 2024; Elhoushi et al., 2024; Xia et al., 2025).

Token-wise layer skipping. Token-wise methods allow different tokens at the same layer or sublayer to take different execute/skip actions. In Mixture-of-Depths, Raposo et al. (2024) fix a token capacity at each layer. In D-LLM, Jiang et al. (2024) instead draw each token's executed depth toward a target; both train a separate model for each budget. Jaiswal et al. (2024) and Luo et al. (2025b) skip FFN computation while retaining attention in FFN-SkipLLM and DiffSkip, respectively. In FlexiDepth, Luo et al. (2025a) combine token-wise layer routers with a skip path adapter for representation alignment; our SCA follows this adapter design. In GateSkip, Laitenberger et al. (2026) gate attention and FFN sublayers, while in BUDDY, Zhou et al. (2026) select an integer number of layers under a request-level budget and can update the selection during decoding. He et al. (2025a) study router-only adaptation in Router-Tuning. In Duo-LLM, Alizadeh et al. (2024) route tokens among differently sized FFN modules, and in SkipGPT, Zhao et al. (2025) allocate attention and FFN computation across tokens. AVERT belongs to this token-wise family, with a runtime request budget used as a target for request-average executed depth. Existing methods provide different subsets of request-local routing, average-budget control, token-varying compute, and continuous budgets; AVERT combines these properties for whole-block layer skipping.

3 Method: AVERT

Within each request, AVERT allocates executed depth across routed decode positions under a request-average target. Figure 1 shows this allocation, layerwise gather–execute–scatter, and compact K/V omission. Skipped rows use the Skip Compensation Adapter (SCA), described below.

Request-local, budget-conditioned routing. The first two Transformer layers always execute. Here, b indexes requests, t indexes request-local decode positions, and ℓ indexes Transformer layers.

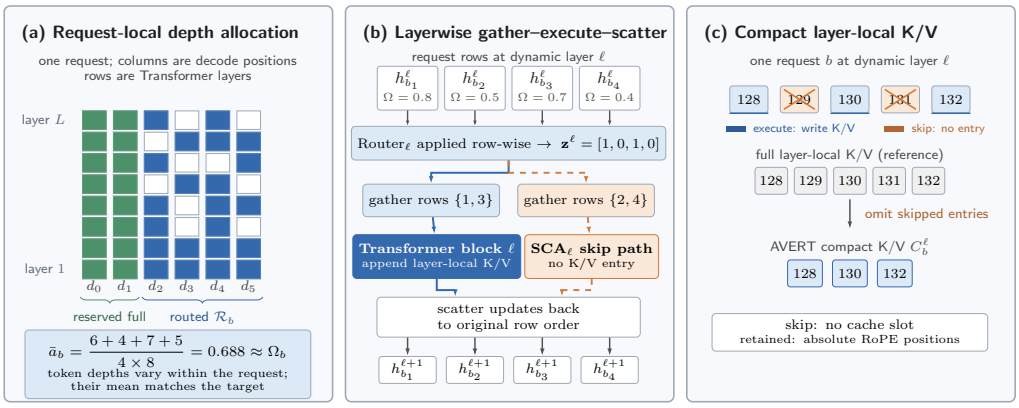

Figure 1: **Overview of AVERT.** (a) Token depths may vary within a request while their routed-position average tracks Ω_b; the first two request-local decode positions execute fully and are excluded from the objective and routing analyses. (b) At each dynamic layer, request-local routers gather selected rows for Transformer execution, send skipped rows through SCA, and scatter both paths back to the original batch order. (c) Only executed layer–token pairs append compact K/V entries. The first two Transformer layers always execute.

Table 1: Budget-handling properties in the 13-target LLaMA evaluation. AdaSkip and GateSkip skip sublayers; the other dynamic methods route whole Transformer blocks.

Method	Training runs	Budget set per	Execution rule at a fixed budget	Heterogeneous-budget batch
AdaSkip	0 additional	inference run	shared sublayer skip set	no
GateSkip	1	batch	batch-coupled sublayer threshold	no
D-LLM	13	trained model	per-token depth target	no
BUDDY	1	request	exact integer layer count per token	yes
AVERT	**1**	**request**	**request-average executed depth**	**yes**

At each subsequent layer ℓ, the router receives the current hidden state $h^\ell_{b,t}$ and the target Ω_b attached to request b:

$$\mathbf{u}^\ell_{b,t} = W^\ell_h h^\ell_{b,t} + \mathbf{b}^\ell_h + W^\ell_\Omega \Omega_b, \qquad \mathbf{s}^\ell_{b,t} = W^\ell_o \, \phi\big(\mathrm{RMSNorm}(\mathbf{u}^\ell_{b,t})\big) + \mathbf{b}^\ell_o \in \mathbb{R}^2, \tag{1}$$

where W^ℓ_h, W^ℓ_Ω, and W^ℓ_o are learned hidden-state, Ω-conditioning, and output projections, and the two logits score $\{\text{skip}, \text{execute}\}$. During training, a straight-through Gumbel–Softmax estimator with temperature $\tau = 1$ yields a hard one-hot forward decision and soft gradients (Jang et al., 2017). At inference, the router deterministically selects the higher-probability action, with ties resolved in favor of execution. The decision for request b depends only on $(h^\ell_{b,t}, \Omega_b)$ and never on another request's score. Unlike batch-wide ranking, this local policy cannot match its target by selecting a global top fraction from the current token pool; it must learn which layers to spend on each token while respecting the supplied target.

Execute and K/V-omitting skip paths. Let $z^\ell_{b,t} \in \{0, 1\}$ be the routing decision, where $z^\ell_{b,t} = 1$ denotes execution and $z^\ell_{b,t} = 0$ denotes skipping. The residual update is

$$h^{\ell+1}_{b,t} = \begin{cases} \mathrm{Block}_\ell(h^\ell_{b,t}), & z^\ell_{b,t} = 1, \\ h^\ell_{b,t} + \mathrm{SCA}_\ell(h^\ell_{b,t}), & z^\ell_{b,t} = 0. \end{cases} \tag{2}$$

On the execute path, attention and FFN run normally and the token writes its layer-local K/V entry. On the skip path, the entire block—including attention—is bypassed, so no entry is produced. Future tokens therefore attend only to the shorter compact K/V history at that layer. Beyond the current block skip, omission avoids the cache write and reduces both cache footprint and later attention work. It may also remove information useful to later tokens, which motivates the quality comparison with retained K/V.

Following the FlexiDepth design of Luo et al. (2025a), SCA addresses representation misalignment on the bypass path. Our SCA is a width-256 bottleneck MLP whose up-projection is zero-initialized: it begins as an identity residual path and then learns a correction. Low-rank adaptation (LoRA) updates the executed attention/FFN path (Hu et al., 2022). Sequence-level cross-entropy supplies indirect training signal through later token predictions as well as the current prediction; the router does not explicitly estimate future cache utility. For request b, let C_b^ℓ denote the compact K/V history stored at layer ℓ. Appendix A gives the decode procedure.

The decision at layer ℓ uses $h_{b,t}^\ell$ after all preceding execute or SCA updates at the same decode step. Routing is therefore progressive rather than precomputed from one hidden state at the start of the dynamic-layer stack. The budget Ω_b conditions each local decision; it does not impose an exact executed-layer count on an individual token. Equation 3 constrains only the request average.

Each layer maintains a separate compact K/V history for every request. Batched attention temporarily pads the executed subset's histories and masks the padded tail; the padding is not retained. Rotary position embeddings (RoPE) use absolute token positions before compaction, which does not renumber tokens.

Prefill, decode, and memory accounting. For the main comparisons, prefill executes every layer. Decoding begins after prefill; its first two request-local input positions execute every layer, and learned routing applies from the third decode position onward. The request-average budget objective and all routing/depth analyses exclude these two positions; wall-clock decode measurements still include their actual execution. Each dynamic block processes only its selected request rows. Fewer block executions, omitted cache writes, and shorter future K/V histories can reduce decode TPOT; routing, gather/scatter, SCA, and subset-kernel overhead limit the realized speedup. Layer weights remain resident. Peak decode GPU memory (VRAM) includes weights, allocated cache capacity, padded temporaries, and workspaces; it excludes prefill-only peaks. Cache buffers grow geometrically with stored-token counts, so physical savings need not match the bytes of omitted entries; see Appendix L. The separately trained PIQA prefill probe computes dense projections before masking skipped positions, so it supports no prefill wall-clock or memory-efficiency claim.

Training objective and overhead. We freeze the backbone and train LoRA, introduced by Hu et al. (2022), on attention and FFN projections, the routers, and SCA using 51,564 Alpaca examples (Taori et al., 2023). Let L be the number of Transformer layers and $z_{b,t}^\ell = 1$ for an executed block, including both always-executed initial layers. For request b, let $\mathcal{T}_b$ be the set of valid decode-input positions at which learned routing is active, excluding the first two decode inputs after prefill, which execute fully. Define $T_b = |\mathcal{T}_b|$, $a_{b,t} = \frac{1}{L}\sum_{\ell=1}^{L} z_{b,t}^\ell$, and $\bar{a}_b = \frac{1}{T_b}\sum_{t\in\mathcal{T}_b} a_{b,t}$ for $T_b > 0$. Here $a_{b,t}$ is the all-layer execution fraction at t, and $\bar{a}_b$ is its average over routed positions. With $\mathcal{B}_+ = \{b : T_b > 0\}$, the loss is

$$\mathcal{L}_{\text{act}} = \frac{1}{|\mathcal{B}_+|}\sum_{b\in\mathcal{B}_+} |\bar{a}_b - \Omega_b|\,, \tag{3}$$

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_{\text{act}}\mathcal{L}_{\text{act}}. \tag{4}$$

This gives equal weight to every request with $T_b > 0$. Token errors may cancel *within* a request; taking absolute values before averaging requests prevents cross-request cancellation. The loss penalizes deviations from Ω_b in each request average without imposing a hard compute cap or an exact per-token layer count. Training samples continuous-valued targets from $[0.4, 1.0]$, warms λ_{act} to 10.0 during the first epoch, and uses rank-16 LoRA. Appendix L specifies sampling and optimization; Appendix J accounts for the non-mergeable router and SCA parameters.

4 Experiments

4.1 Setup

Models and baselines. We evaluate LLaMA-2-7B (multi-head attention, MHA; 32 layers) from Touvron et al. (2023), LLaMA-3.1-8B (grouped-query attention, GQA; 32 layers) from Grattafiori et al. (2024), and LLaMA-3.2-1B (GQA; 16 layers) from Grattafiori et al. (2024). Vanilla is the Alpaca-finetuned full-compute reference; it uses rank-16 LoRA on the same Q/K/V and FFN

up/down/gate projections as AVERT. In D-LLM, Jiang et al. (2024) train one specialist per budget. In its released implementation, later-token attention is computed over the full materialized history before the eviction mask is applied, so masked positions neither shorten attention nor reduce decode-cache storage. For system measurements, we preserve its checkpoint and routing policy but physically omit masked K/V entries. GateSkip, proposed by Laitenberger et al. (2026), is evaluated in two adaptation regimes: GateSkip-FFT uses its original full-parameter fine-tuning, while GateSkip-LoRA matches our LoRA setting. Its released Hugging Face gate runs after the Transformer computation, while the authors' physical-skipping patch does not support cached autoregressive attention. Accordingly, GateSkip-FFT's reported decode TPOT reflects its released cached-autoregressive path rather than the physical-skipping patch. BUDDY (reimpl.), based on Zhou et al. (2026), reconstructs the paper's explicit-budget integer Top-k path because its official code was unavailable; the reported runs load no static prior, and the remaining implementation choices are documented in Appendix L. AdaSkip, proposed by He et al. (2025b), applies a fixed attention/FFN sublayer skip set during decode, with its online window disabled and full-compute prefill. The reduced Qwen2.5-3B evaluation is in Appendix P.

Metrics. Alpaca uses 1,000 held-out examples to test open-ended instruction-following generation. We treat Prometheus-2 (1–5) as the primary Alpaca metric because lexical similarity to one reference need not imply a correct or useful open-ended response (Kim et al., 2024). The judge receives the instruction, generated response, reference answer, and a fixed rubric, then returns feedback and an integer score; Appendix L gives the scale. ROUGE-L, described by Lin (2004), and BLEU-4, described by Papineni et al. (2002), provide complementary lexical-overlap measurements. GSM8K uses 8-shot prompting on 1,319 questions and reports flexible answer-extraction accuracy (Cobbe et al., 2021). Its multi-step reasoning chains stress whether insufficient computation at an intermediate token propagates through later reasoning. Official zero-shot IFEval evaluates 541 prompts and 834 instructions; its strict prompt and strict instruction accuracies test whether skipping preserves explicit constraints throughout generation using rule-based checks independent of the Alpaca LLM judge (Zhou et al., 2023). PIQA (1,838 examples) is a separate zero-shot prompt-log-likelihood probe using a separately trained prefill-skipping checkpoint (Bisk et al., 2020).

Budgets and efficiency. We evaluate 13 targets $\Omega \in \{0.40, 0.45, \ldots, 1.00\}$. For each method, Ω denotes the requested keep rate at that method's native routing granularity: whole Transformer blocks for AVERT, D-LLM, and BUDDY, and attention/FFN sublayers for GateSkip and AdaSkip. *Realized activation* is the corresponding measured fraction of native routing units that execute. For AVERT, it averages all L blocks, including the first two always-executed blocks, over routed decode positions; the first two full-execution decode positions are excluded. For each target, we average realized activation over requests assigned that target; *calibration error* is the absolute difference between this mean and Ω. Thus $\Omega = 0.4$ denotes 60% nominal omission at the method's native granularity, not 60% measured speedup. Unless stated otherwise, all methods use full-compute 128-token prefill and 128 decode-produced tokens to measure TPOT and peak *decode* VRAM at batch size (BS) 16. CUDA memory statistics are reset after prefill; the peak includes allocated K/V capacity and decode temporaries. Appendix L gives timing details and Appendix Q varies prompt length.

Evaluation protocol. All six main quality metrics and system measurements use full-compute prefill across the compared methods and skip only during decode. The PIQA prefill probe is the sole quality exception. Alpaca evaluation examples are excluded from training; GSM8K and IFEval are not used for AVERT fine-tuning. D-LLM uses 13 specialists; GateSkip-FFT and GateSkip-LoRA assess two adaptation regimes. At requested Ω, the main GateSkip comparison focuses on quality and TPOT; Appendix M compares TPOT and peak decode VRAM at each method's measured realized activation. To test request-local calibration, homogeneous and heterogeneous batches both use BS=32. The controlled mixed-versus-split study uses identical requests, budgets, checkpoints, and inference implementation in both modes; only batch packing changes. It processes 128 requests per workload across four budgets, forces 128 output tokens per request, and averages 10 runs. Budget counts, exact batch packing, and timing are in Appendix E.

Table 2: Main results at the near-full target ($\Omega = 1.0$) and the aggressive target ($\Omega = 0.4$). IFEval-SP/SI are strict prompt/instruction accuracies; GSM8K-Flex is flexible answer-extraction accuracy. Best dynamic result per setting is in **bold**, using unrounded values.

		Prom.		ROUGE-L		BLEU-4		GSM8K-Flex		IFEval-SP		IFEval-SI	
Model	Method	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4	1.0	0.4
LLaMA-2-7B (MHA, 32L)	Vanilla	3.39	–	.278	–	.116	–	.116	–	.246	–	.354	–
	GateSkip-FFT	3.08	1.10	.261	.159	.104	.022	.024	.001	.142	.059	.252	.182
	GateSkip-LoRA	3.32	1.00	.255	.000	.109	.000	.033	.002	.202	.046	.301	.133
	D-LLM	**3.40**	1.51	.281	.190	.125	.036	.120	.011	.255	.081	.366	.177
	AdaSkip	3.38	1.01	.277	.034	.116	.002	.116	.003	.244	.061	.352	.131
	BUDDY (reimpl.)	3.12	1.39	**.291**	.170	.122	.034	.064	.015	.196	.068	.302	.157
	AVERT (Ours)	**3.40**	**2.38**	.279	**.221**	**.126**	**.080**	**.121**	**.017**	**.275**	**.159**	**.398**	**.255**
LLaMA-3.1-8B (GQA, 32L)	Vanilla	3.66	–	.280	–	.123	–	.428	–	.325	–	.436	–
	GateSkip-FFT	3.00	1.02	.251	.113	.096	.019	.083	.006	.196	.056	.330	.160
	GateSkip-LoRA	3.58	1.05	.281	.140	.124	.014	.317	.005	.257	.048	.384	.132
	D-LLM	**3.66**	1.62	.284	.173	.124	.050	.422	.005	.359	.052	.460	.146
	AdaSkip	**3.66**	1.01	.281	.009	.124	.001	**.428**	.000	.324	.061	.438	.091
	BUDDY (reimpl.)	3.29	1.48	**.298**	.170	**.126**	.035	.176	.016	.242	.070	.354	.163
	AVERT (Ours)	3.65	**2.02**	.281	**.199**	.124	**.061**	.425	**.020**	**.362**	**.092**	**.471**	**.189**
LLaMA-3.2-1B (GQA, 16L)	Vanilla	2.93	–	.250	–	.098	–	.044	–	.128	–	.224	–
	GateSkip-FFT	2.66	1.06	.240	.065	.085	.006	.023	.001	**.142**	.070	**.249**	.145
	GateSkip-LoRA	1.95	1.08	.214	.061	.069	.005	.026	.003	.103	.052	.199	.144
	D-LLM	2.93	1.26	.247	.166	.098	.045	**.045**	.011	.120	.074	.222	.152
	AdaSkip	2.92	1.00	.249	.001	.097	.000	.044	.000	.126	.061	.225	.134
	BUDDY (reimpl.)	2.33	1.17	**.250**	.133	.091	.018	.018	**.021**	.085	.081	.186	**.165**
	AVERT (Ours)	**2.94**	**1.44**	.250	**.187**	**.098**	**.057**	.045	.017	.133	**.085**	.239	.158

4.2 Main quality results

At $\Omega = 1.0$, AVERT remains close to Vanilla and the strongest baselines. This endpoint is not the unmodified backbone: AVERT uses trained LoRA, router, and SCA parameters, and its realized activation can remain slightly below one. AdaSkip changes the inference path least at this endpoint and remains close to Vanilla. At $\Omega = 0.4$, AVERT is best or tied for best in 16 of the 18 model–metric cells in Table 2. Its Prometheus gain over D-LLM is $+0.87$ on LLaMA-2-7B, $+0.40$ on LLaMA-3.1-8B, and $+0.18$ on LLaMA-3.2-1B; the margins vary by architecture. BUDDY is stronger on LLaMA-3.2-1B GSM8K-Flex and IFEval-SI at this target. Robustness to aggressive layer skipping is backbone-dependent. LLaMA-3.1-8B can degrade more steeply than LLaMA-2-7B at low budgets despite higher full-compute quality, so we compare methods within each backbone rather than interpreting cross-backbone differences as effects of model size or architecture.

The quality sweep aligns requested targets; these need not produce equal realized compute across methods. Figure 3 instead plots Prometheus-2 directly against measured decode TPOT for the 13 requested-budget operating points of each method. Quality and TPOT are measured under their respective evaluation protocols and paired by method, backbone, and requested target. The curves compare measured quality–TPOT trade-offs rather than assuming that a shared requested target gives equal compute.

For example, on LLaMA-3.1-8B, AVERT at requested $\Omega = 0.55$ scores 2.89 Prometheus with 3.519 ms/token, while AdaSkip at $\Omega = 0.85$ scores 2.85 with 3.942 ms/token. Appendix N gives matched-quality counts across all existing operating points, and Appendix G reports the AdaSkip pairs and peak decode VRAM.

At $\Omega = 0.7$, AVERT is best or tied for best in all 21 model–metric comparisons, including the auxiliary PIQA probe. Across all 13 targets, the six main metrics yield 206 best-or-tied results among 3 architectures $\times$ 6 metrics $\times$ 13 targets $= 234$ comparisons; including the auxiliary PIQA probe gives 240/273 across seven reported quality metrics. These cell counts summarize the measured sweep and are not statistical-significance tests.

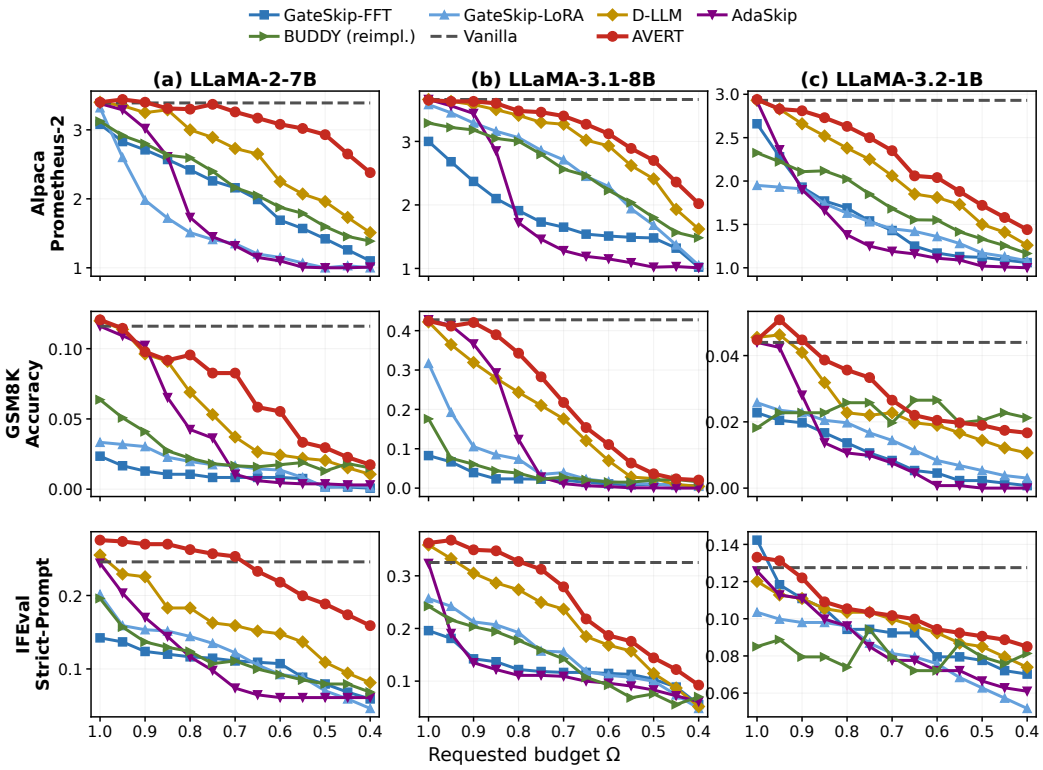

Figure 2: **Quality across the full 13-point budget sweep.** Columns are backbones; rows report Prometheus-2 on Alpaca, GSM8K flexible accuracy, and IFEval strict-prompt accuracy. Each curve connects the 13 requested targets, and dashed lines mark Vanilla. Remaining metrics are in Appendix B.

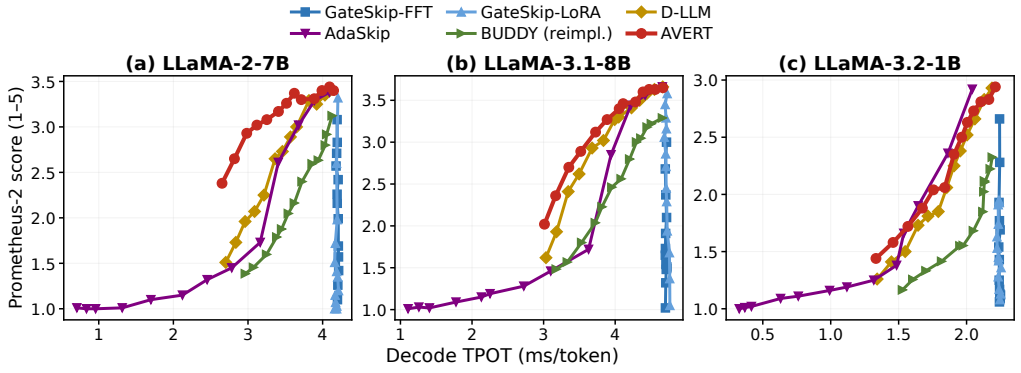

Figure 3: **Prometheus-2 versus measured decode TPOT.** Each curve connects the 13 evaluated operating points for one method on a fixed backbone. Lower TPOT and higher Prometheus-2 are better. Points are joined in requested-budget order; equal requested targets need not imply equal realized activation.

GSM8K flexible accuracy approaches the floor at low Ω. We therefore treat $\Omega = 0.4$ as an aggressive endpoint that exposes the controlled trade-off range, not as a quality-preserving operating point; practical budgets are task dependent and generally higher. Within each backbone, AVERT usually retains the strongest quality while using one checkpoint and preserving request-level control.

Five independent training runs on LLaMA-3.1-8B show small absolute variation across seeds relative to the changes across budgets, as reported in Appendix R.

Routing behavior beyond a per-token Top-k plan. At a fixed target, higher dense-reference uncertainty quartiles receive greater mean execution depth. Across targets on the same teacher-forced text, lower-budget layer sets need not be subsets of higher-budget sets. Appendix O reports both analyses. BUDDY selects a complete Top-k layer set before execution within each decode step, whereas AVERT routes progressively, one layer at a time. The observations show token-depth variation and layer sets that can change across budgets, but they do not establish that either behavior alone causes the quality gains. Non-nesting indicates that the selected layer set changes with budget rather than being obtained by truncating a single fixed layer ranking.

4.3 Request-local control under batching

Single-budget curves do not test whether heterogeneous batchmates perturb a request's target. With BS=32, the maximum absolute difference between each requested target and the mean realized activation for requests assigned that target is 0.85, 0.97, and 0.94 percentage points in homogeneous batches and 1.06, 1.15, and 0.89 in mixed batches, for LLaMA-2-7B, LLaMA-3.1-8B, and LLaMA-3.2-1B, respectively. Each mixed batch contains four requests at each of eight targets; see Appendix E. On five additional off-grid targets within the trained interval, the corresponding largest differences are 0.85, 0.90, and 0.78 percentage points; Appendix F shows quality, TPOT, VRAM, and calibration together. These are differences of per-target means, not worst-case errors over individual requests. For a 32-layer BUDDY path, budgets 0.940, 0.945, and 0.950 all round to the same 30 executed layers per token. BUDDY uses the same executed-layer count at every token, so its fixed-budget activation granularity is $1/32$. AVERT accepts a continuous-valued target; over T routed decode positions, one request's realized activation has granularity $1/(32T)$.

GateSkip-FFT behaves differently because its released control rule derives a sublayer threshold from the current batch-wide live-token distribution. On LLaMA-3.1-8B at requested $\Omega = 0.4$, realized activation is 1.000 at BS=1, 0.632 at BS=2, approximately 0.298 at BS=8/16, and 0.289 at BS=32. Prometheus over the same sweep ranges from 3.01 to 1.02, a maximum variation of 1.99 points. Quantile thresholding also does not explicitly retain an exact integer k, so even the batch-wide realized fraction need not match the requested target. AVERT remains stable across its own BS sweep; Section 4.4 reports the corresponding quality and efficiency ranges, and Appendix D provides the full batch-size results.

BUDDY itself supports heterogeneous budgets; this study does not model a BUDDY limitation. It isolates the batching cost imposed when a budget is fixed per batch or checkpoint. The study assigns 128 requests among four budgets, $\{1.0, 0.8, 0.6, 0.4\}$, under four workloads: balanced, moderate skew, strong low-budget skew, and strong high-budget skew. Mixed-versus-split execution uses the same AVERT checkpoint on both sides, so the only intervention is whether requests with different budgets may share the same physical batch or must be partitioned into homogeneous batches. The balanced workload is a no-fragmentation control for batching efficiency and timing: equal request counts across the four budgets let both modes form four full BS=32 batches, and mean TPOT differs by at most 0.15% (total wall-clock by at most 0.24%). Across models the small difference changes sign, showing no measurable material overhead from heterogeneous-budget mixing itself. Peak VRAM can still differ because split execution contains homogeneous high-budget batches and reports the largest per-batch peak. Under moderate skew, splitting requires five batches and lowers average fill to 80%; under either strong skew, it requires six batches and lowers fill to 66.7%, while mixed execution still uses four full batches. These underfilled split batches process fewer concurrent requests and therefore use batching capacity less efficiently. Across the three models, mixed execution reduces TPOT—and, because every workload generates the same 16,384 output tokens, total decode time—by 4.3–21.2%; total wall-clock time falls by 4.1–20.9%. The gain comes from avoiding underfilled homogeneous batches; model weights and routing policy are identical in the two modes. Peak decode VRAM is workload- and model-dependent rather than uniformly lower under mixed execution; exact values are in Appendix E.

4.4 Efficiency and component ablations

Layer skipping and K/V omission affect decode efficiency in different ways, as shown in Figure 8; see Appendix C for the full results. Retaining K/V for skipped positions preserves more attention-visible history and can improve quality, but it gives up the peak decode VRAM benefit. It is also generally slower at aggressive budgets. The omission path avoids current block work and cache writes, and it shortens later attention history. Both TPOT and VRAM savings remain below the nominal $1 - \Omega$ block-compute saving because fixed costs and hardware utilization do not scale linearly.

At numerically equal keep rates, AdaSkip has lower TPOT and peak decode VRAM at all nine evaluated settings, but its quality falls faster as the rate decreases. Before timing, we selected three quality-matched pairs per architecture from the existing Prometheus-2 sweeps (absolute score gaps: 0.00–0.06); matching AVERT's quality requires AdaSkip to retain more sublayers. Across the nine pairs, AVERT has lower TPOT in 7/9 cases and lower peak decode VRAM in 3/9; in the other six memory comparisons, the gap is at most 5.7%. Appendix G reports the pairs and measurement protocol.

Equal requested budgets need not yield equal executed compute for token-level baselines. Appendix M compares decode TPOT and peak VRAM against measured activation for AVERT, D-LLM, and GateSkip-FFT/LoRA at 13 points per backbone, without interpolation. Equal activation need not imply equal FLOPs for GateSkip, which routes attention/FFN sublayers rather than whole blocks. BUDDY uses integer Top-k at each decode step, fixing executed layer counts up to rounding without post-hoc matching.

For the SCA ablation, we compare quality at similar mean activation. At $\Omega = 0.4$ on LLaMA-3.1-8B, the full model and a separately trained no-SCA model realize 0.4067 and 0.4078 activation, yet Prometheus falls from 2.02 to 1.41 without SCA; D-LLM reaches 1.62. The no-SCA model has slightly higher mean activation, so fewer executed layers on average do not explain its quality gap; the result indicates that residual repair matters in the single-checkpoint multi-budget setting. Detailed results are provided in Appendix H.

Appendix I directly ablates the request-average objective and progressive routing. Relative to the per-token alternative, the request-average objective gives higher Prometheus-2 at 11 of 13 targets, ties at one, and is lower at one; it gives higher ROUGE-L and BLEU-4 at all 13 targets. Progressive routing gives higher Prometheus-2 at 12 targets and higher BLEU-4 at 10, while ROUGE-L is higher at 6 targets and lower at 7; the full one-shot results are therefore kept in the appendix.

Across BS $\in \{1, 2, 4, 8, 16, 32\}$ on LLaMA-3.1-8B, AVERT varies by at most 0.15 Prometheus, 0.0135 ROUGE-L, and 0.0119 BLEU-4. Larger batches lower TPOT but raise peak decode VRAM. Appendix K finds lower local skip-versus-execute target-rank drift for skipped layer–token pairs; this probe does not measure how omitted K/V affects future tokens.

Across 128–2048-token full-compute prefills and a fixed 128-token decode, lowering Ω reduces TPOT and peak decode VRAM at every tested length. The absolute TPOT reduction generally grows with prompt length, whereas the increase in peak-memory savings is small; see Appendix Q.

5 Limitations and Conclusion

AVERT can lose full-compute accuracy at aggressive budgets, particularly on GSM8K. The five off-grid tests stay within the trained interval $[0.4, 1.0]$; extrapolation and budgets changing across stages, tokens, or layers remain untested.

Our PyTorch decode benchmark excludes stochastic arrivals, queueing, and online continuous batching, so mixed-budget TPOT gains do not predict deployment latency. Peak-memory gains vary by workload. PIQA tests prefill quality without reducing prefill compute; the reduced Qwen2.5-3B appendix sweep provides only a limited architecture check beyond three LLaMA models.

AVERT combines request-local progressive routing, a request-average depth target, and compact K/V caches to allocate whole-block execution across tokens within independently budgeted requests using one checkpoint.

Reproducibility Statement

The method and objective are specified in Section 3. Appendix L records training, budget sampling, and evaluation settings. The anonymous supplementary code provides AVERT training and evaluation scripts, the Alpaca train/evaluation split, GSM8K, IFEval, and PIQA evaluation code, and defaults needed to instantiate AVERT from separately obtained base checkpoints. It also contains our documented BUDDY reimplementation. Other baseline implementations are taken from public releases and are not redistributed. Appendix G describes the static omission of AdaSkip's selected decode sublayers; the corresponding baseline patch is not redistributed. If reproduction difficulties arise, please contact the authors through OpenReview. The anonymous code repository is available at `https://anonymous.4open.science/r/AVERT-1F8B`.

AI Use Statement

Generative AI tools assisted with drafting and editing portions of the manuscript, locating and summarizing related work, discussing research ideas and experimental design, and interpreting experimental results. They also assisted with code development and debugging for training and evaluation, and with preparing figures. The authors selected the final methods and experiments, ran the reported evaluations, checked the cited references and reported results, and reviewed the final text, analyses, figures, and code. The authors take responsibility for the content of this submission.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134. USENIX Association, 2024.

Yash Akhauri and Mohamed S. Abdelfattah. Compute where it counts: Self optimizing language models. In *Forty-third International Conference on Machine Learning*, 2026.

Keivan Alizadeh, Iman Mirzadeh, Hooman Shahrokhi, Dmitry Belenko, Frank Sun, Minsik Cho, Mohammad Hossein Sekhavat, Moin Nabi, and Mehrdad Farajtabar. Duo-LLM: A framework for studying adaptive computation in large language models. In *Proceedings of the 4th NeurIPS Workshop on Efficient Natural Language and Speech Processing*, 2024.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. FREE: Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5910–5924. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.362.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Controlling computation versus quality for neural sequence models, 2020.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

Yuhang Chen, Jinhao Duan, Ruichen Zhang, Mingfu Liang, Xiaohan Wei, Yunchen Pu, Fei Tian, Chonglin Sun, Parish Aggarwal, Frank Shyu, Luke Simon, Sandeep Pandey, Tianlong Chen, and Xi Liu. End-to-end dynamic sparsity for resource-adaptive LLM inference, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. SkipDecode: Autoregressive skip decoding with batching and caching for efficient LLM inference, 2023.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, and Yequan Wang. Not all layers of LLMs are necessary during inference. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 5083–5091. International Joint Conferences on Artificial Intelligence Organization, 2025. doi: 10.24963/ijcai.2025/566.

Aaron Grattafiori et al. The Llama 3 herd of models, 2024.

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, and Dong Yu. Router-Tuning: A simple and effective approach for dynamic depth. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1925–1938. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.emnlp-main.99.

Zhuomin He, Yizhen Yao, Pengfei Zuo, Bin Gao, Qinya Li, Zhenzhe Zheng, and Fan Wu. AdaSkip: Adaptive sublayer skipping for accelerating long-context LLM inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24050–24058, 2025b. doi: 10.1609/aaai.v39i22.34579.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Akriti Jain, Saransh Sharma, Koyel Mukherjee, and Soumyabrata Pal. FiRST: Finetuning router-selective transformers for input-adaptive latency reduction. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21957–21975. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.1197.

Ajay Kumar Jaiswal, Bodun Hu, Lu Yin, Yeonju Ro, Tianlong Chen, Shiwei Liu, and Aditya Akella. FFN-SkipLLM: A hidden gem for autoregressive decoding with adaptive feed forward skipping. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16943–16956. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.941.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

Yikun Jiang, Huanyu Wang, Lei Xie, Hanbin Zhao, Chao Zhang, Hui Qian, and John C. S. Lui. D-LLM: A token adaptive computing resource allocation strategy for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1725–1749. Curran Associates, Inc., 2024. doi: 10.52202/079017-0055.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4334–4353. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.248.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165.

Filipe Laitenberger, Dawid Jan Kopiczko, Cees G. M. Snoek, and Yuki M. Asano. What layers when: Learning to skip compute in LLMs with residual gates. In *The Fourteenth International Conference on Learning Representations*, 2026.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81. Association for Computational Linguistics, 2004.

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

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained LLMs. In *Second Conference on Language Modeling*, 2025a.

Xuan Luo, Weizhi Wang, and Xifeng Yan. DiffSkip: Differential layer skipping in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 7221–7231. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-acl.377.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318. Association for Computational Linguistics, 2002. doi: 10.3115/1073083.1073135.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. GitHub repository, 2023.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *International Conference on Learning Representations*, 2025.

Ning Yang, Fangxin Liu, Junjie Wang, Tao Yang, Kan Liu, Haibing Guan, and Li Jiang. DASH: Input-aware dynamic layer skipping for efficient LLM inference with Markov decision policies, 2025.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538. USENIX Association, 2022.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.607.

Anhao Zhao, Fanghua Ye, Yingqi Fan, Junlong Tong, Jing Xiong, Zhiwei Fei, Hui Su, and Xiaoyu Shen. SkipGPT: Each token is one of a kind. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 77844–77863. PMLR, 2025.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210. USENIX Association, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023.

Yuhua Zhou, Shaoqi Yu, Shichao Weng, Changhai Zhou, Mingze Yin, Fei Yang, and Aimin Pan. BUDDY: BUdget-Driven DYnamic depth routing for adaptive large language model inference, 2026.

Yunqi Zhu, Xuebing Yang, Yuanyuan Wu, and Wensheng Zhang. Hierarchical skip decoding for efficient autoregressive language model. In *Advances in Information Retrieval*, pp. 219–228. Springer Nature Switzerland, 2025. doi: 10.1007/978-3-031-88714-7_20.

A PROGRESSIVE DECODE WITH COMPACT K/V CACHES

At a single-token decode step, active request b has request-local decode-input index r_b and absolute sequence position p_b. Index $r_b = 0$ is the first single-token step after prefill: the token produced by prefill is the input to this step. The reserved positions are $r_b = 0$ and $r_b = 1$, the first two *decode* inputs rather than prompt positions. Algorithm 1 also handles ragged batches, where different active requests may have different r_b values.

Algorithm 1 Request-local progressive decode with compact layer-local K/V

Input: active requests $\mathcal{B}$; hidden rows h_b^1; budgets Ω_b; decode-input indices r_b; absolute positions p_b; persistent caches C_b^ℓ.

1. For layers $\ell = 1, \ldots, L$, form the forced-execute set $F^\ell = \mathcal{B}$ if $\ell \leq 2$, and $F^\ell = \{b \in \mathcal{B} : r_b < 2\}$ otherwise.
2. For each $b \in \mathcal{B} \setminus F^\ell$, compute $z_b^\ell = \mathrm{Router}_\ell(h_b^\ell, \Omega_b)$ independently. Set $E^\ell = F^\ell \cup \{b : z_b^\ell = 1\}$ and $S^\ell = \mathcal{B} \setminus E^\ell$.
3. Gather rows in E^ℓ. Apply RoPE at each absolute position p_b, attend to the compact history C_b^ℓ, execute attention and FFN, and append the new K/V entry to C_b^ℓ.
4. For every $b \in S^\ell$, set $h_b^{\ell+1} = h_b^\ell + \mathrm{SCA}_\ell(h_b^\ell)$ and leave C_b^ℓ unchanged.
5. Scatter both paths back to the original request order before processing layer $\ell + 1$.

Output: apply final normalization and the LM head to h_b^{L+1} for every active request.

Only executed layer–token pairs append persistent K/V, and each router observes the hidden state produced by all preceding decisions at the same decode step. The objective and routing/depth analyses exclude positions with $r_b < 2$; wall-clock decode measurements include them.

B COMPLETE QUALITY RESULTS ACROSS ALL REPORTED METRICS

We give the complete 13-point Alpaca/GSM8K sweeps, the auxiliary PIQA prefill-skipping probe, and the full IFEval strict sweeps for each architecture.

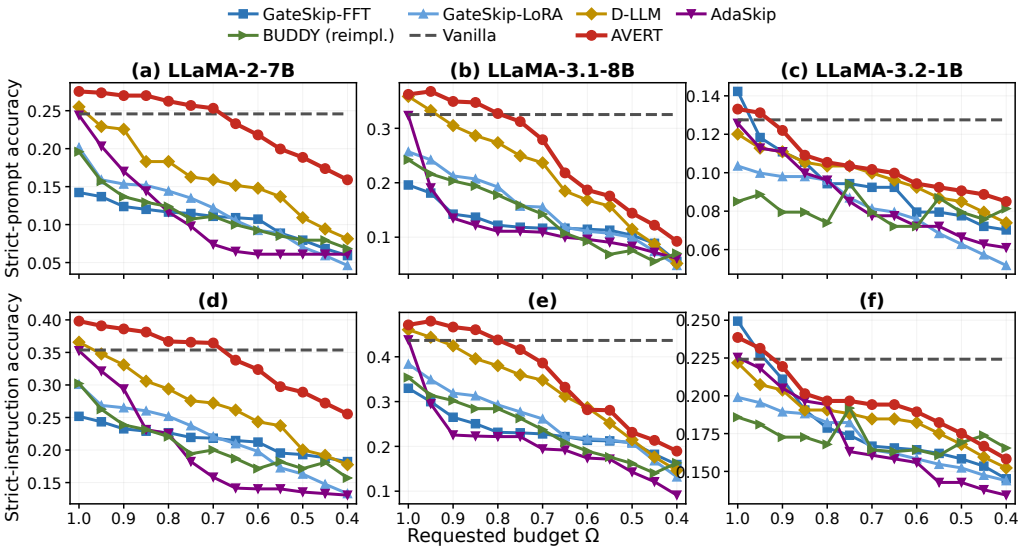

Figure 4: Full IFEval strict prompt/instruction sweeps.

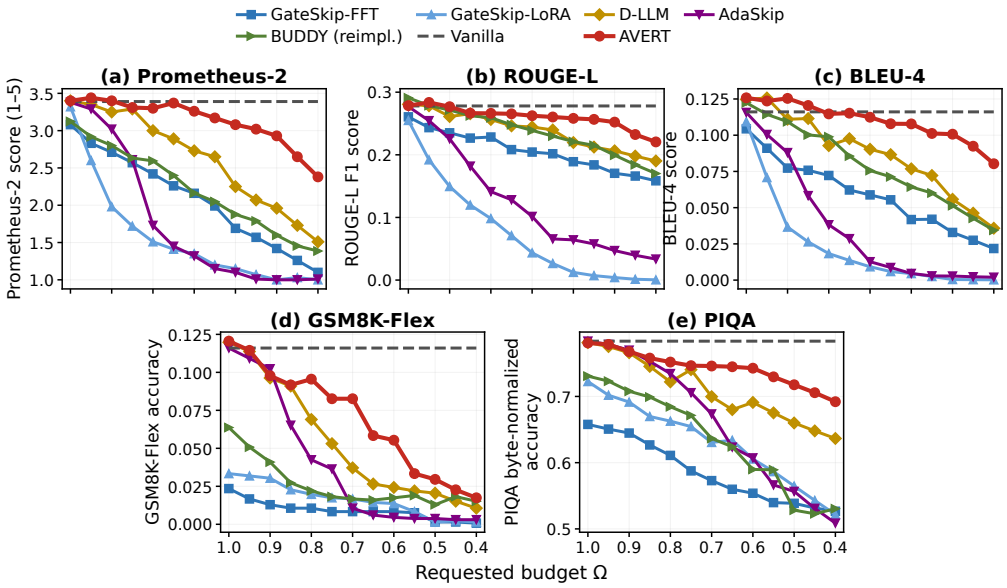

Figure 5: **LLaMA-2-7B (MHA), Alpaca/GSM8K plus auxiliary PIQA metrics.**

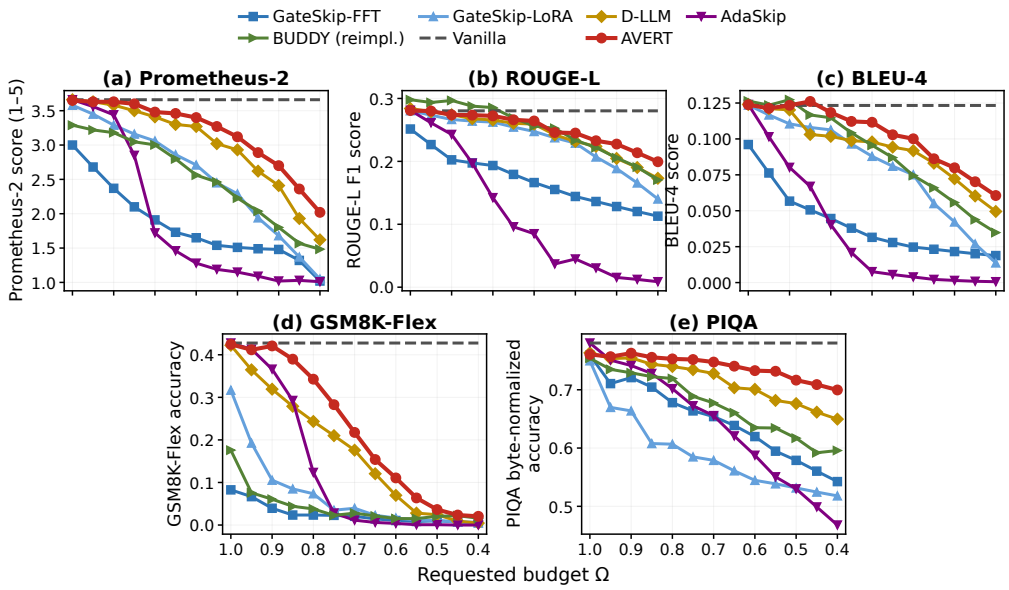

Figure 6: **LLaMA-3.1-8B (GQA), Alpaca/GSM8K plus auxiliary PIQA metrics.**

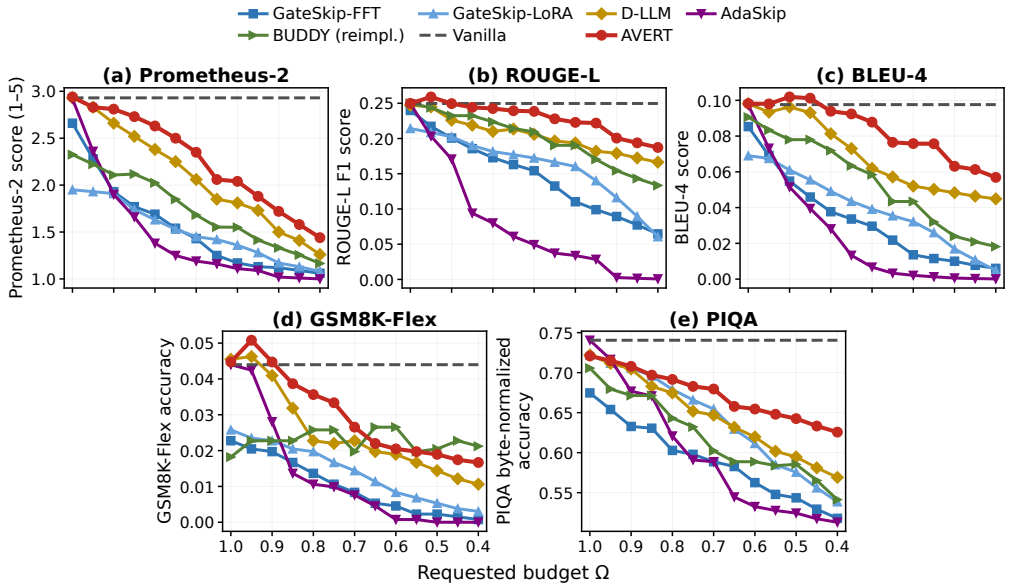

Figure 7: **LLaMA-3.2-1B (GQA, 16L), Alpaca/GSM8K plus auxiliary PIQA metrics.**

C Full K/V omission-versus-retention ablation across architectures

The complete comparisons report Prometheus, ROUGE-L, BLEU-4, decode TPOT, and peak decode VRAM. In the retention ablation, a router-skipped layer–token pair computes only that layer's K and V projections and appends them to the cache; it then follows the SCA bypass without computing the attention output or FFN.

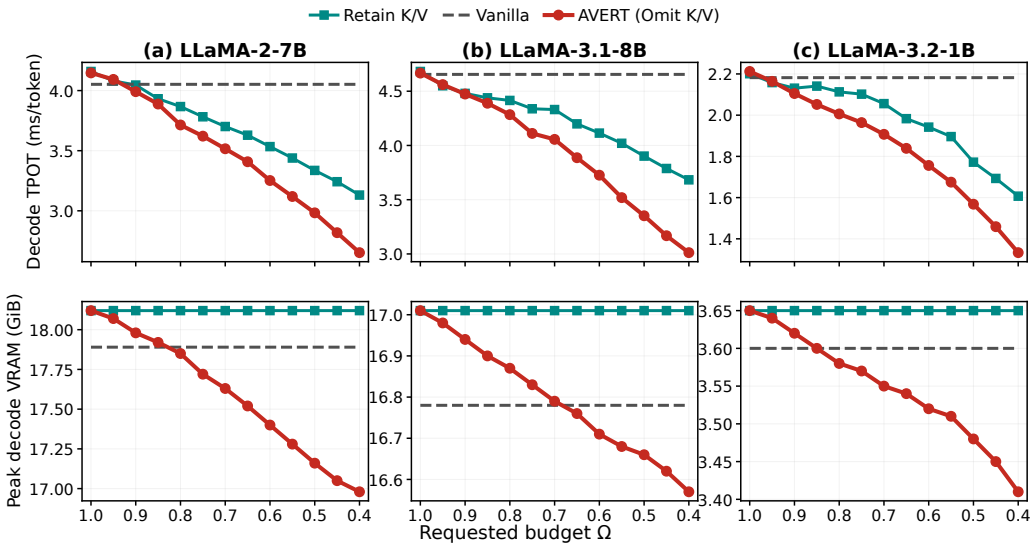

Figure 8: **Measured TPOT and memory effects of K/V omission.** Layer skipping reduces TPOT in both variants; omission avoids cache writes, shortens future attention histories, and reduces peak decode VRAM. Retention is generally slower at aggressive budgets and keeps peak decode VRAM near its $\Omega = 1.0$ level.

Retention therefore preserves attention-visible history and keeps the cache footprint approximately constant over Ω, whereas omission lowers peak decode VRAM as the target decreases. Near $\Omega = 1.0$, the TPOT difference between the two paths is small; the additional benefit from K/V omission becomes clearer at lower budgets.

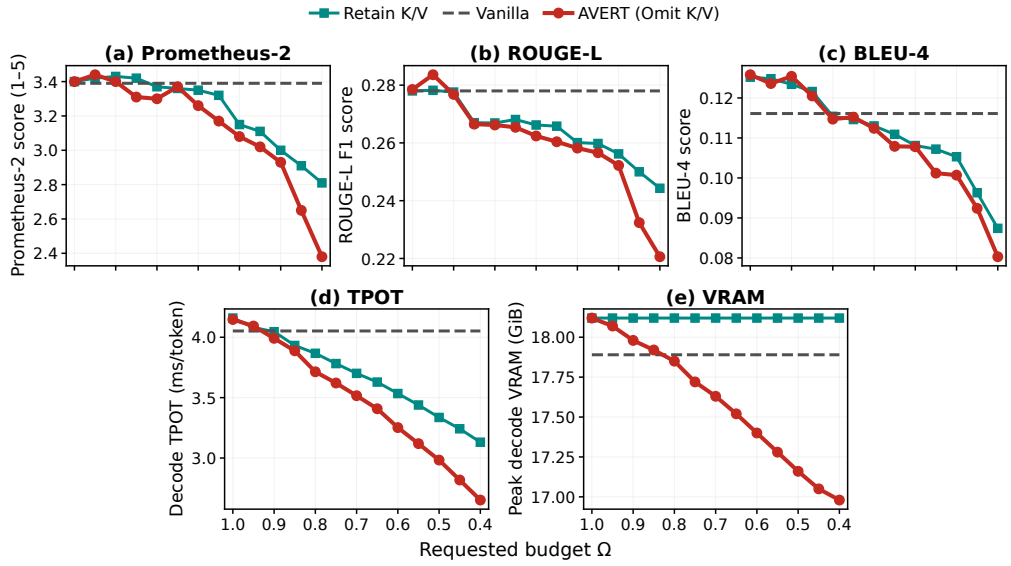

Figure 9: **K/V omission versus retention on LLaMA-2-7B (MHA).** MHA yields the largest cache-memory benefit from omitted K/V entries.

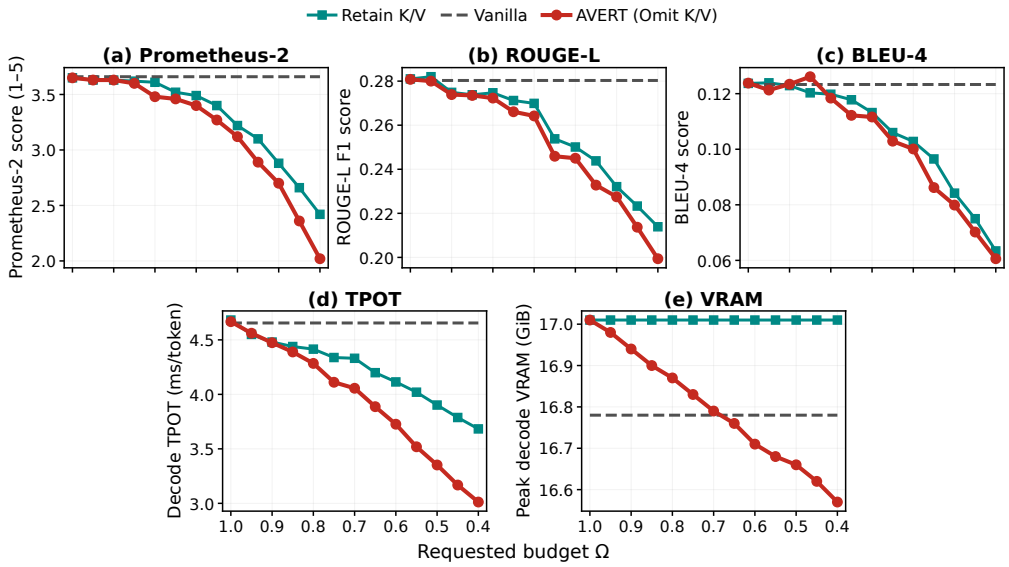

Figure 10: **K/V omission versus retention on LLaMA-3.1-8B (GQA).** GQA already compresses K/V, so the memory gap is smaller than for MHA.

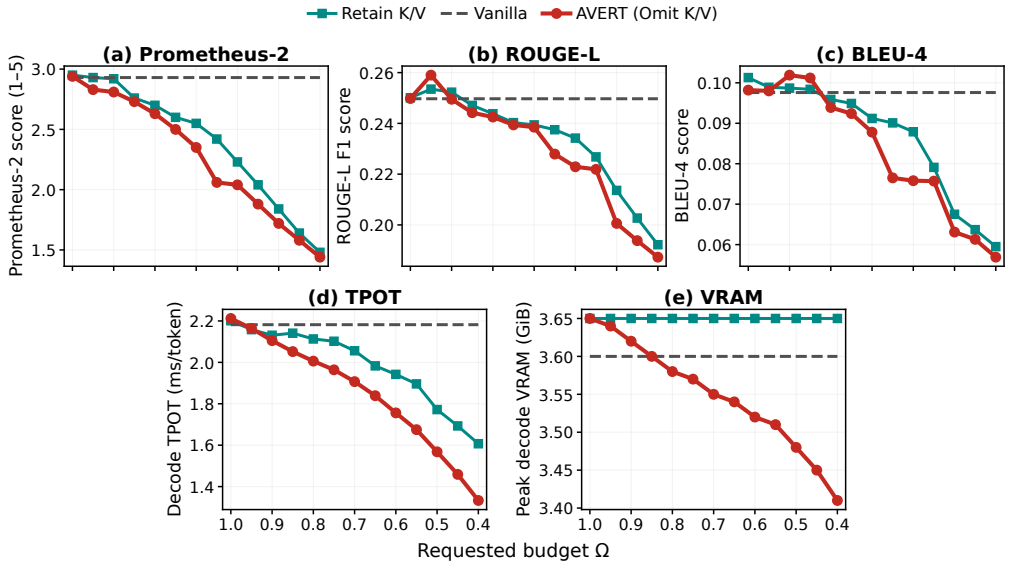

Figure 11: **K/V omission versus retention on LLaMA-3.2-1B (GQA, 16L).**

D BATCH-SIZE SENSITIVITY

We evaluate BS $\in \{1, 2, 4, 8, 16, 32\}$ on the 1,000-example Alpaca set with LLaMA-3.1-8B. The three Alpaca quality metrics use the standard generation protocol. TPOT and peak decode VRAM are measured separately with a fixed 128-token prefill and 128 decode-produced tokens at every batch size. AVERT quality remains stable as batching changes, while TPOT falls as GPU utilization improves and peak decode VRAM grows with concurrent requests.

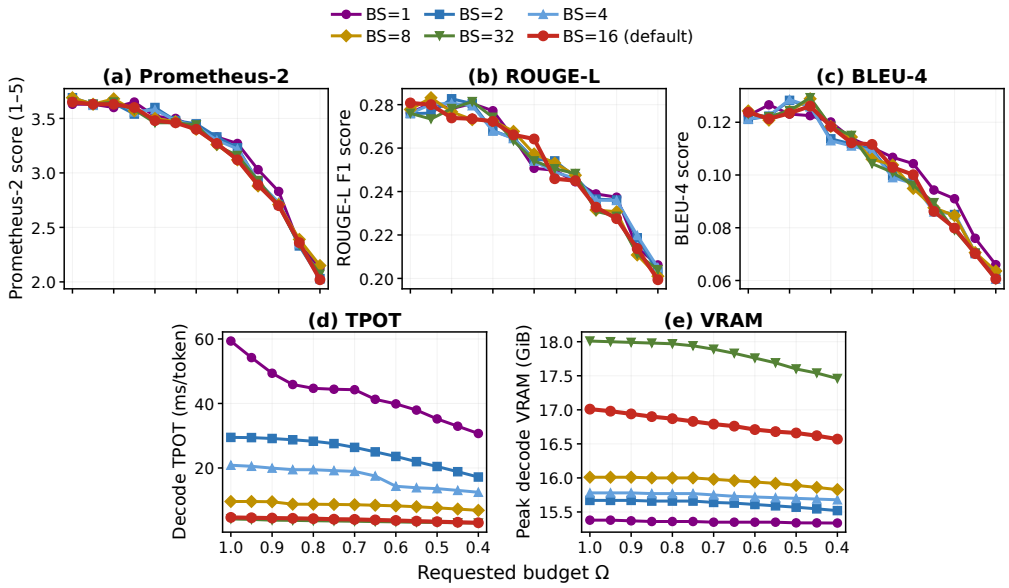

Figure 12: **AVERT batch-size ablation on LLaMA-3.1-8B.** Quality varies little across batch sizes, whereas TPOT benefits from batching.

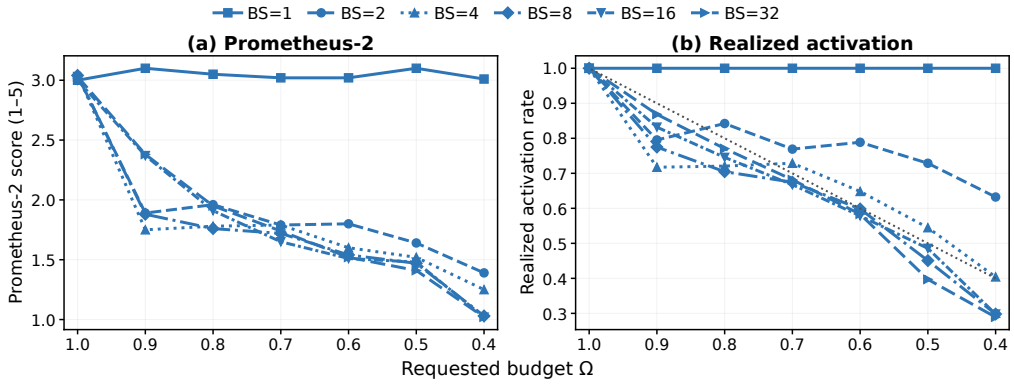

Figure 13: **GateSkip-FFT batch-size sensitivity on LLaMA-3.1-8B.** Left: Prometheus versus requested Ω. Right: realized activation versus requested Ω, with the identity line as the calibration target. Both quality and realized compute change sharply with BS because the selector uses a threshold derived from the current batch-wide live-token pool.

E MIXED-BUDGET CALIBRATION AND BUDGET-FRAGMENTATION EXECUTION

The calibration study uses BS=32 for both homogeneous and mixed evaluation. Homogeneous calibration runs a separate batch for each target; the maximum absolute difference between each requested target and the mean realized activation for requests assigned that target is 0.85, 0.97, and 0.94 percentage points for LLaMA-2-7B, LLaMA-3.1-8B, and LLaMA-3.2-1B. Each full mixed calibration batch contains four requests at each of eight targets, $\{1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.45, 0.4\}$; the corresponding maximum differences are 1.06, 1.15, and 0.89 percentage points. These values are not maxima over individual requests.

The execution study uses the same AVERT checkpoint for mixed and split modes and identical request–budget pairs within each workload. Budget counts at $\Omega = \{1.0, 0.8, 0.6, 0.4\}$ are $(32, 32, 32, 32)$ for balanced, $(16, 24, 32, 56)$ for moderate skew, $(8, 16, 24, 80)$ for strong low-budget skew, and

$(80, 24, 16, 8)$ for strong high-budget skew. Using tuple order $(1.0, 0.8, 0.6, 0.4)$, the four fixed mixed batches are $4 \times (8, 8, 8, 8)$ for balanced; $(8, 8, 8, 8), (8, 8, 8, 8), (0, 8, 12, 12), (0, 0, 4, 28)$ for moderate skew; $(8, 8, 8, 8), (0, 8, 12, 12), (0, 0, 4, 28), (0, 0, 0, 32)$ for strong low-budget skew; and $(8, 8, 8, 8), (12, 12, 8, 0), (28, 4, 0, 0), (32, 0, 0, 0)$ for strong high-budget skew. Split execution independently chunks each budget class into as many full BS=32 batches as possible plus one remainder batch, yielding 4, 5, 6, and 6 physical batches. The corresponding average split-batch fill ratios, relative to BS32 capacity, are 100%, 80%, 66.7%, and 66.7%. Every request uses a fixed 128-token prefill and is forced to generate exactly 128 output tokens (end-of-sequence stopping is disabled). For each run, total timing sums over the physical batches, whereas workload peak decode VRAM is the maximum of their per-batch peaks; Table 3 reports the 10-run means. Because all workloads generate 16,384 output tokens, relative TPOT and total decode-time reductions are identical. The balanced workload is a no-fragmentation control for batching efficiency and timing. Peak VRAM can still differ because split execution contains homogeneous high-budget batches and reports the maximum per-batch peak. Under skewed workloads, mixed execution avoids underfilled homogeneous batches and reduces TPOT. Peak decode VRAM is workload- and model-dependent and is not a universal benefit of mixed execution.

Table 3: **Mixed-budget versus homogeneous split execution.** Means over 10 runs using the same AVERT checkpoint and identical request–budget pairs. Batch counts shown as mixed/split; fill is the average split-batch fill ratio.

Workload	Model	Total wall-clock (s)		Decode TPOT (ms/token)		Peak decode VRAM (GiB)	
		Mixed	Split	Mixed	Split	Mixed	Split
Balanced 4/4 batches; 100% fill	LLaMA-2-7B	54.53	54.47	3.263	3.258	17.13	19.01
	LLaMA-3.1-8B	57.78	57.91	3.480	3.484	17.89	18.01
	LLaMA-3.2-1B	29.61	29.65	1.780	1.781	3.47	4.03
Moderate skew 4/5 batches; 80% fill	LLaMA-2-7B	50.28	53.72	3.003	3.214	17.04	18.13
	LLaMA-3.1-8B	54.57	56.88	3.280	3.428	17.88	17.77
	LLaMA-3.2-1B	27.08	28.34	1.625	1.702	3.47	3.65
Strong low-budget skew 4/6 batches; 66.7% fill	LLaMA-2-7B	46.79	59.14	2.791	3.544	17.03	17.55
	LLaMA-3.1-8B	52.15	60.60	3.133	3.643	17.89	17.60
	LLaMA-3.2-1B	24.85	26.82	1.489	1.608	3.47	3.50
Strong high-budget skew 4/6 batches; 66.7% fill	LLaMA-2-7B	61.72	77.15	3.702	4.643	19.00	19.01
	LLaMA-3.1-8B	62.61	71.34	3.779	4.307	18.10	18.02
	LLaMA-3.2-1B	32.59	34.86	1.961	2.100	4.04	4.04

F EVALUATION BETWEEN THE REPORTED BUDGET GRID POINTS

We evaluate five targets that are not part of the reported 0.05-spaced grid, using the same values on all three backbones: $\Omega \in \{0.4783, 0.6348, 0.7316, 0.8273, 0.9719\}$. Across these targets, the maximum absolute difference between a requested target and the mean realized activation for requests assigned that target is 0.85, 0.90, and 0.78 percentage points for LLaMA-2-7B, LLaMA-3.1-8B, and LLaMA-3.2-1B, respectively. Figure 14 compares these measurements with the 13 on-grid targets. Figure 15 overlays the same off-grid evaluations on the on-grid Prometheus-2, decode TPOT, and peak decode VRAM curves.

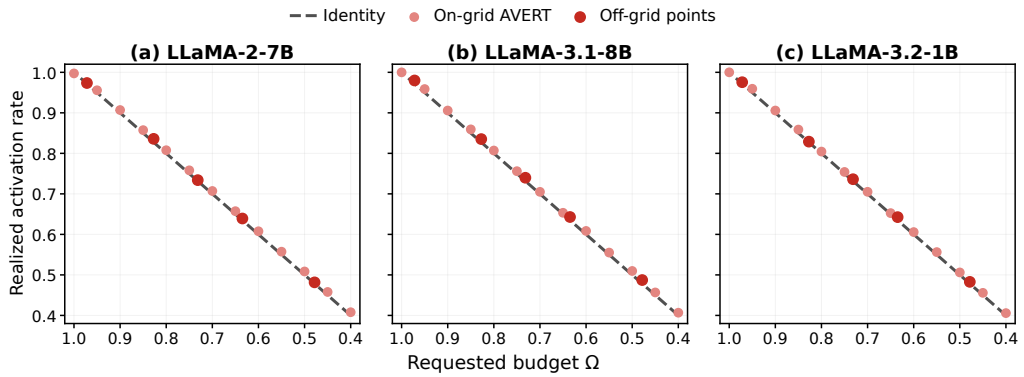

Figure 14: **On-grid and off-grid budget calibration.** Requested target versus realized all-layer activation. The identity line marks exact calibration; the five off-grid targets are overlaid on the 13-point on-grid sweep.

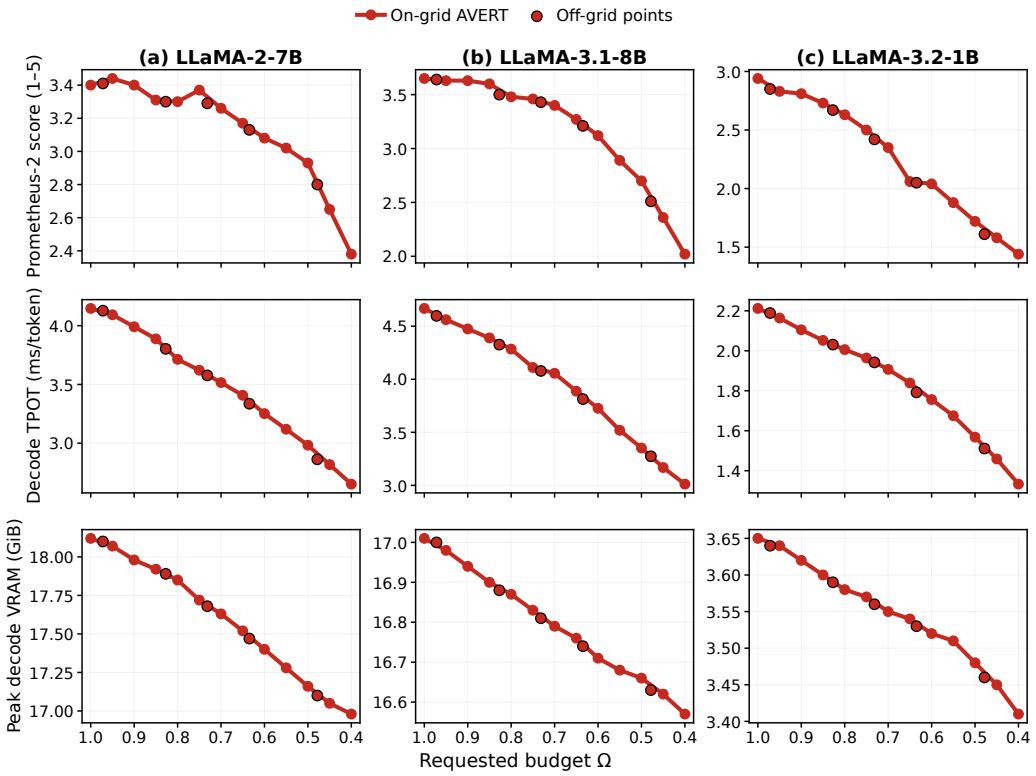

Figure 15: **Off-grid operating points within the trained budget interval.** The 13 on-grid AVERT points define the requested-budget curves; the five off-grid evaluations are overlaid for Prometheus-2, decode TPOT, and peak decode VRAM.

G Matched-quality efficiency against AdaSkip

We compare AVERT with AdaSkip at three Prometheus-2-matched operating points per architecture. The pairs were selected from the existing quality sweeps before TPOT and peak decode VRAM measurement; the absolute Prometheus difference is at most 0.06 in every pair. The numeric keep rates are not identical routing units: Ω is AVERT's mean Transformer-layer activation target, while

AdaSkip applies shared attention/FFN sublayer skipping. We therefore use Prometheus-2, rather than equal keep rate, to define the matched-quality comparison.

Both methods use the same timing protocol. We take the first 176 held-out Alpaca requests so every batch has BS=16. Batch 0 is a warm-up batch. Batches 1–10 contain the 160 timed requests; every request uses a fixed 128-token prefill and is forced to generate 128 output tokens, giving 20,480 timed output tokens in total. TPOT is the sum of decode time over those ten batches divided by $10 \times 16 \times 128$. Peak decode VRAM is measured in GiB with `torch.cuda.max_memory_allocated()/(1024**3)`. All measurements use the same NVIDIA A800 80GB GPU.

AdaSkip's online window is disabled. As for all compared methods, prefill executes every layer, and the selected attention/FFN skip set is applied only during decode.

At the AdaSkip keep rates used below, its fixed attention/FFN sublayer skip pattern gives lower TPOT and peak decode VRAM than AVERT at numerically equal keep rates. Under matched Prometheus-2 quality, AdaSkip must retain more sublayers. AVERT has lower TPOT in 7/9 pairs and lower peak decode VRAM in 3/9; in the remaining six memory comparisons, AVERT is higher by at most 5.7%.

Table 4: **Prometheus-2-matched efficiency against AdaSkip.** Each row compares two operating points with similar Prometheus-2 quality. TPOT is in ms/token and peak decode VRAM in GiB.

Model	Keep rate (AVERT/AdaSkip)	Prometheus (AVERT/AdaSkip)	TPOT (AVERT/AdaSkip)	Peak decode VRAM (AVERT/AdaSkip)
LLaMA-2-7B	.80/.95	3.30/3.29	3.714/3.891	17.85/17.43
LLaMA-2-7B	.55/.90	3.02/3.02	3.119/3.678	17.28/16.88
LLaMA-2-7B	.45/.85	2.65/2.61	2.818/3.406	17.05/16.13
LLaMA-3.1-8B	.85/.95	3.60/3.56	4.389/4.419	16.90/16.51
LLaMA-3.1-8B	.75/.90	3.46/3.44	4.111/4.217	16.83/16.23
LLaMA-3.1-8B	.55/.85	2.89/2.85	3.519/3.942	16.68/15.86
LLaMA-3.2-1B	.70/.95	2.35/2.36	1.907/1.863	3.55/3.56
LLaMA-3.2-1B	.55/.90	1.88/1.90	1.675/1.644	3.51/3.53
LLaMA-3.2-1B	.40/.80	1.44/1.38	1.333/1.485	3.41/3.45

H FULL SCA ABLATION RESULTS

The no-SCA model is trained without SCA from the start and uses an identity skip path during both training and inference. This is an architectural ablation with a separately trained model. We measure realized activation for both models rather than assuming that a shared requested target implies identical realized compute. On LLaMA-3.1-8B at $\Omega = 0.40$, full AVERT and no-SCA realize 0.4067 and 0.4078 activation, respectively, while Prometheus falls from 2.02 to 1.41. D-LLM reaches 1.62 at the same requested target.

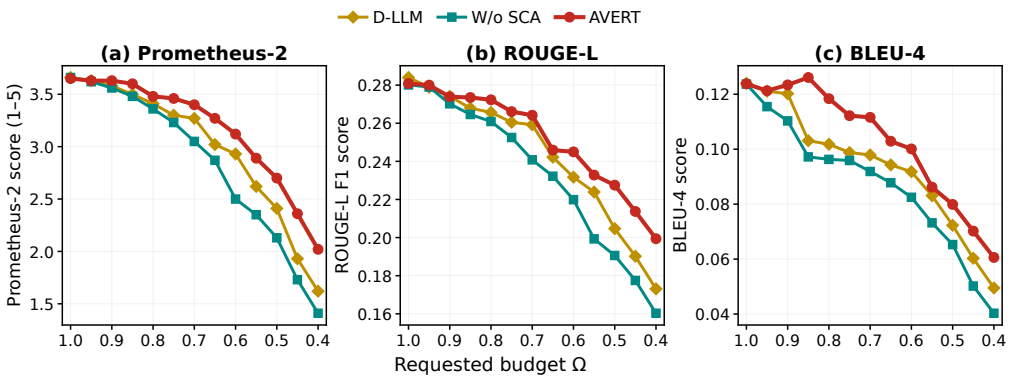

Figure 16: **SCA ablation on LLaMA-3.1-8B.** The quality gap between full AVERT and the separately trained no-SCA model widens at lower requested budgets.

I ROUTING-OBJECTIVE ABLATIONS

Request-average objective. The per-token alternative uses the same set of positions eligible for routing, the same all-layer execution-fraction definition and targets, and the same equal weighting across requests as Eq. 3, but moves the absolute value inside the within-request token average. Deviations above and below Ω_b therefore cannot offset one another, pushing each token toward the same depth. The largest realized-activation difference between the two models is 0.71 percentage points across the 13 targets. On LLaMA-3.1-8B, the request-average objective has higher Prometheus-2 at 11 targets, ties at one, and is slightly lower at one; it has higher ROUGE-L and BLEU-4 at all 13 targets. This comparison changes the objective without changing the architecture or inference path.

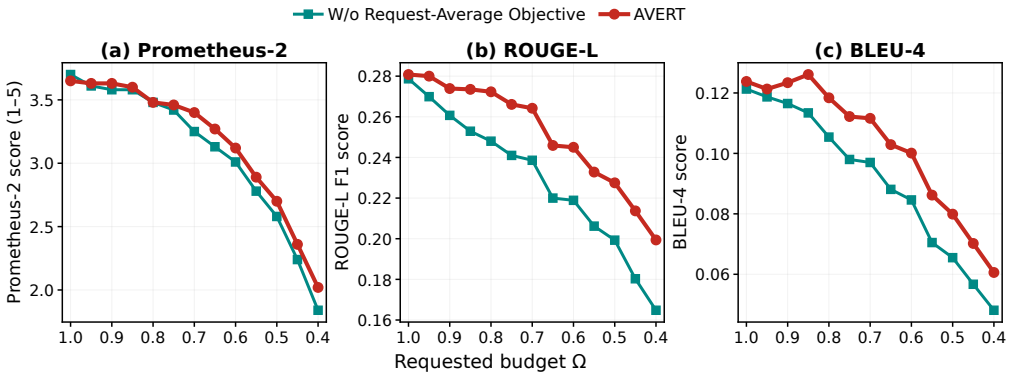

Figure 17: **Request-average objective ablation on LLaMA-3.1-8B.** "W/o Request-Average Objective" uses the per-token absolute-error objective described above.

Progressive versus one-shot routing. The one-shot ablation computes every dynamic-layer routing decision from the same hidden state at the start of the dynamic-layer stack. It retains the first two always-executed layers, the two full-execution decode positions, compact K/V omission, SCA, and the request-average objective; only the router-state source changes. The largest realized-activation difference between the two models is 0.92 percentage points across the 13 targets. Progressive AVERT has higher Prometheus-2 at 12 targets and higher BLEU-4 at 10. ROUGE-L is mixed: progressive routing is higher at 6 targets and one-shot routing at 7. The Prometheus-2 comparison favors progressive routing, whereas the lexical metrics do not do so uniformly.

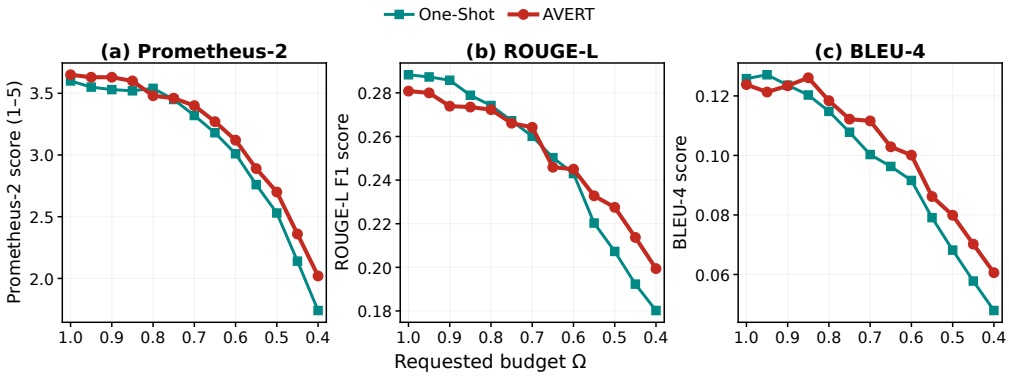

Figure 18: **Progressive-routing ablation on LLaMA-3.1-8B.** One-shot routing decisions are all computed from the same hidden state before the dynamic-layer stack; AVERT routes progressively from layer-updated states.

J PARAMETER AND STATIC OVERHEAD ACCOUNTING

Table 5 separates mergeable LoRA parameters from the router+SCA modules that remain at deployment. The MiB values assume fp16/bf16 storage. This is a static parameter footprint; measured decode peaks additionally include cache capacity and temporary allocations, so the peak difference from Vanilla need not equal the parameter overhead.

Table 5: **Static parameter overhead.** LoRA parameters are mergeable into the base model; router+SCA parameters remain at deployment.

Model	LoRA params (mergeable)	Router+SCA params	Non-mergeable overhead
LLaMA-2-7B	35.78M	125.91M	≈240 MiB
LLaMA-3.1-8B	37.75M	125.91M	≈240 MiB
LLaMA-3.2-1B	10.22M	29.40M	≈56 MiB
Qwen2.5-3B	27.57M	71.39M	≈136 MiB

K LOGIT-LENS PROBE OF LOCAL ROUTING STABILITY

We use a logit-lens-style counterfactual probe to analyze whether AVERT skips locally stable layer–token pairs. The first two request-local decode positions advance the actual trajectory at full depth but are excluded from this analysis. At each subsequent dynamic-layer position of the on-policy decode trajectory, we branch the current hidden state into execute and skip+SCA outcomes. We project both through the final normalization and LM head, and measure $|\log(r_{\text{skip}}+1)-\log(r_{\text{execute}}+1)|$ for the token selected by the actual routed decode path, with ranks starting at one. Only the router-selected branch advances and writes persistent K/V when it executes. This local current-token probe does not measure how omitted K/V affects future tokens.

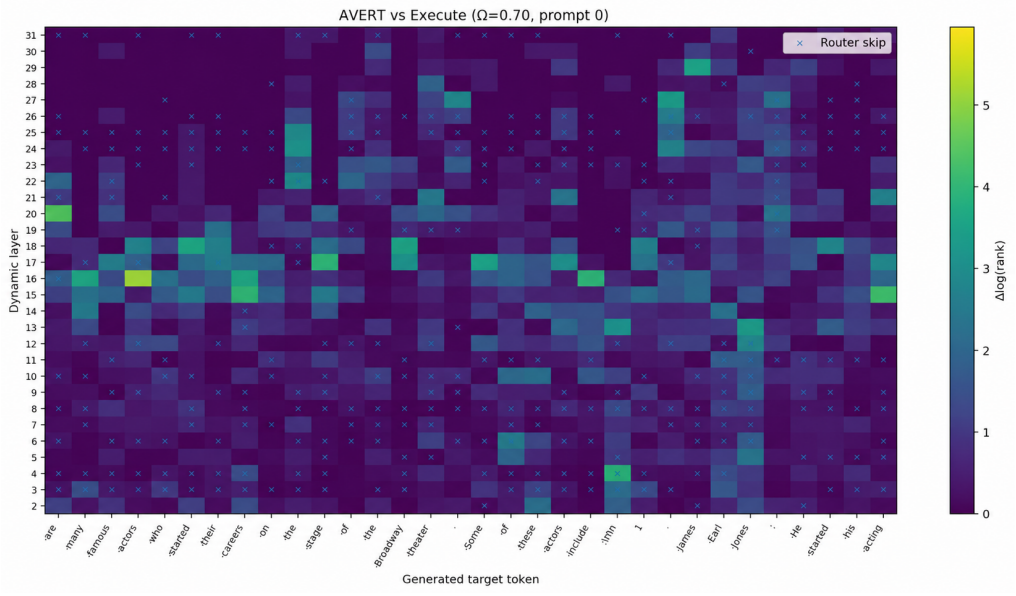

Figure 19: **Qualitative logit-lens probe on LLaMA-3.1-8B at** $\Omega = 0.70$**.** Color indicates target-token $\Delta\log(\text{rank})$ between skip and execute paths; crosses mark router-skipped pairs. Rows begin at zero-indexed layer 2 because the first two Transformer layers always execute and are not plotted. The first two full-execution decode inputs are also excluded.

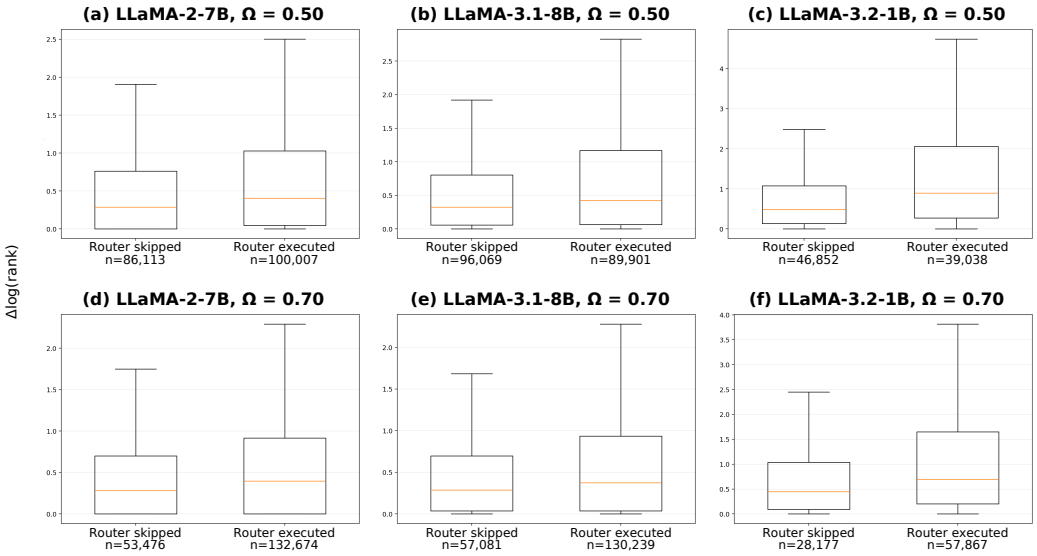

Figure 20: **Aggregate local rank drift by router decision.** Across all three architectures and two targets, router-skipped layer–token pairs show distributions of local skip-versus-execute target-rank drift concentrated toward lower values than executed pairs. This is a routing signal, not evidence that K/V omission is lossless.

L Reproducibility Details

Training data and optimization. All three main LLaMA AVERT models use the 51,564-example Alpaca training split included in the supplementary archive. The disjoint 1,000-example evaluation set contains the 805-example D-LLM test set plus 195 additional held-out Alpaca-format examples. The released training launchers use 5 epochs, batch size 1, gradient accumulation of 24, maximum sequence length 512, and seed 0 by default. The effective batch size is `batch_size`×`accum_iter`×world size. Optimization uses AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.05, and base learning-rate parameter 5×10^{-3}; the actual learning rate is scaled by effective batch size divided by 256. The schedule uses one epoch of linear warm-up followed by half-cycle cosine decay to zero. The activation-loss weight is warmed linearly from 0 to 10.0 over the first epoch and remains at 10.0 thereafter. The base model is frozen; LoRA, router, and SCA parameters are trained, with LoRA rank 16, router hidden width 512, SCA bottleneck 256, and dynamic routing beginning after the first two Transformer layers. LoRA is applied to the query, key, and value projections W_q, W_k, W_v and the FFN up, down, and gate projections $W_{\text{up}}, W_{\text{down}}, W_{\text{gate}}$; it is not applied to the attention output projection W_o. It uses dropout 0.05 and no additional α/r multiplier. The Vanilla references use rank-16 LoRA with the same dropout and Q/K/V and FFN projection scope, without routers or SCA. For AVERT, the frozen base weights are loaded in bfloat16, while the trainable LoRA, router, and SCA parameters are maintained in float32 during training.

Budget sampling. Training uses request-level targets in $\Omega \in [0.4, 1.0]$. For 90% of training batches, request targets are sampled continuously and independently from the uniform distribution on this interval. For the remaining 10%, one endpoint, 0.4 or 1.0, is selected at random and assigned to the current distributed microbatch. In distributed training, the sampled global target vector is broadcast from rank 0 before each worker selects its local slice. The default training path keeps prompt/prefill positions at full compute and applies the activation-rate objective only where decode routing is active; the auxiliary PIQA variant enables routing on prefill positions as described in Section 4.1. In the default decode-routing path, the first two request-local decode inputs after prefill are reserved for full execution and excluded from the activation-rate objective. In the auxiliary PIQA path, the first two valid prefill positions are likewise forced to execute and excluded from its objective.

Hardware. The main LLaMA training, inference, quality evaluation, TPOT, and peak decode VRAM experiments use a single NVIDIA A800 80GB GPU. With batch size 1 and gradient accumulation of 24, training uses an effective batch size of 24; the base learning-rate scaling therefore yields an actual learning rate of 4.6875×10^{-4}. The released scripts use single-GPU execution with `NPROC_PER_NODE=1`. Core software dependencies are listed in the supplementary `requirements.txt`; the README records the optional vLLM requirement for Prometheus-2 scoring.

Evaluation configuration. The reported quality sweep uses 13 requested targets from 0.40 to 1.00 in increments of 0.05. Alpaca generation uses the full 1,000-example held-out split, batch size 16, maximum generation length 256, temperature 0, and top-$p = 1$. GSM8K uses all 1,319 test questions with 8-shot prompting, batch size 16, maximum generation length 256, greedy decoding, and flexible answer extraction. IFEval uses all 541 official prompts in the zero-shot setting, batch size 16, maximum generation length 512, temperature 0, and top-$p = 1$. The official prompt is wrapped in the Alpaca instruction template for generation; the bundled rule-based evaluator still receives the original official prompt and instruction keys. PIQA uses all 1,838 examples as an auxiliary prefill-skipping probe. Each example is formatted as `Question: <goal>` followed by `Answer:`; both candidate continuations are scored by log likelihood, and the reported accuracy selects the larger log likelihood after normalization by the candidate's UTF-8 byte length. The auxiliary checkpoint is trained only on the same 51,564-example Alpaca split; PIQA is used only for evaluation. Prometheus scoring uses `prometheus-eval/prometheus-7b-v2.0` in direct-assessment mode with bfloat16 inference, temperature 0, and a 512-token judge-output limit. The judge receives the instruction, generated response, reference answer, and a fixed rubric: scores 1–2 denote unusable or substantially incorrect responses, 3 denotes an acceptable response with errors or omissions, 4 denotes a mostly correct and complete response with minor issues, and 5 denotes an accurate, complete, fluent, and well-organized response. Malformed or unparseable judge outputs receive score 1; the released script records and reports their count. All evaluated models use bfloat16 inference for the reported generation, TPOT, and peak decode VRAM measurements. Unless otherwise stated in the paper, generation and K/V measurements use batch size 16; mixed-budget and batch-size experiments use the configurations specified in Sections 4.3–4.4 and Appendices D–E.

BUDDY reimplementation. No official BUDDY source code was publicly available when the comparison was prepared. Our explicit-budget reimplementation follows the paper-specified first and last always-executed Transformer blocks, Linear–SiLU–Linear Decision Module, budget-to-$k = \mathrm{round}(bL) - 2$ mapping over the middle blocks, hard Top-k selection, rerouting at every autoregressive decode step, first-layer Value states as the planner feature, and aligned zero-filled K/V positions for skipped states. The paper names a Token-Pooling operator and a Taylor prior without releasing their exact implementations. We reconstruct Token-Pooling as a causal mean over first-layer Value states and provide a first-order residual-removal proxy for the optional prior, but the reported BUDDY runs set the prior weight to zero and do not load or use a prior. For the comparison in this paper, BUDDY training covers $[0.4, 1.0]$, uses five epochs rather than the paper's two, and follows the common full-prefill evaluation protocol. The optional Group Relative Policy Optimization (GRPO) budget predictor is not used because every BUDDY evaluation supplies an explicit budget. We therefore label all results BUDDY (reimpl.).

Synchronized TPOT measurement. For all synchronized system benchmarks, including Qwen, prompts are left-truncated or attended left-padded to 128 positions unless the experiment specifies another physical prompt length. The full batch completes full-compute prefill before 128 single-token decode steps. The token produced by prefill bootstraps the first decode step and is excluded from both decode timing and the output count; EOS stopping is disabled. The prompt-length study instead uses physical prefill lengths from 128 to 2048 while keeping 128 decode-produced tokens. The LLaMA-3.1-8B batch-size sweep uses the same fixed-shape system workload, while its Alpaca quality metrics use the standard generation protocol.

For the standard BS=16 operating-point benchmark, one warm-up batch is followed by ten timed batches. TPOT is the synchronized decode wall-clock time over the ten timed batches divided by $10 \times 16 \times 128$. Timing includes model forward passes, routing, gather/scatter, SCA, compact-cache handling, greedy token selection, and the Python decode loop. Experiments with other batch sizes or repetition counts state those settings separately.

After each batch's full prefill and bootstrap, prefill logits are released and CUDA peak statistics are reset. Peak decode VRAM is then measured over decode, and the operating-point value is the maximum per-batch decode peak across the warm-up and timed batches. AVERT's per-request, per-layer K/V buffers round the initial required length to a 64-token block and leave one additional 64-token block of headroom; when a buffer fills, capacity grows to the rounded maximum of twice the current capacity and the required length. This budget-independent policy makes physical allocation stepwise rather than equal to the number of valid cached positions. D-LLM system results use our compact K/V path; its released implementation computes later-token attention over the full materialized history before applying the eviction mask.

M EFFICIENCY VERSUS REALIZED ACTIVATION

Requested targets are not used as the horizontal coordinate in this comparison. For AVERT, D-LLM, GateSkip-FFT, and GateSkip-LoRA, we record realized activation at each method's routing granularity together with decode TPOT and peak decode VRAM under the fixed 128-token prefill, 128-token decode, BS=16 system protocol. For AVERT, the activation coordinate excludes its two reserved full-execution decode positions. Each curve connects that method's own 13 measured operating points; equal horizontal values need not represent equal FLOPs across whole-block and sublayer methods. BUDDY is omitted from this diagnostic because its explicit-budget route executes an exact integer Top-k count at every decode step, up to budget discretization.

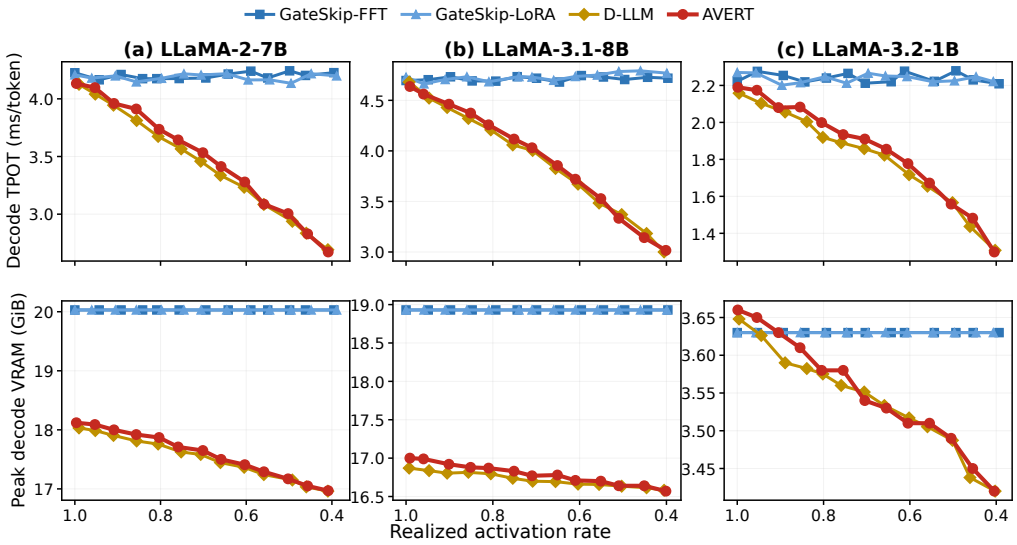

Figure 21: **Decode efficiency versus measured realized activation.** Top: decode TPOT. Bottom: peak decode VRAM. Each method is plotted at its own 13 measured activation coordinates; no interpolation is used to force common x values.

N EXISTING QUALITY–EFFICIENCY OPERATING POINTS

We join the measured Alpaca Prometheus scores with TPOT and peak decode VRAM measurements by backbone, method, and requested target. These are configuration-level comparisons: quality uses the generation evaluation protocol, while system measurements use the fixed 128-token prefill and 128-token output protocol. For each of the 195 baseline operating points (five baselines, three backbones, 13 targets), we choose the AVERT point with the nearest Prometheus score; an exact quality-distance tie is resolved by the higher requested target. A pair is retained if the absolute score difference is at most 0.05. Neither latency nor memory participates in selecting the pair. The tolerance is descriptive and is not a statistical equivalence margin; points are not interpolated, and unmatched

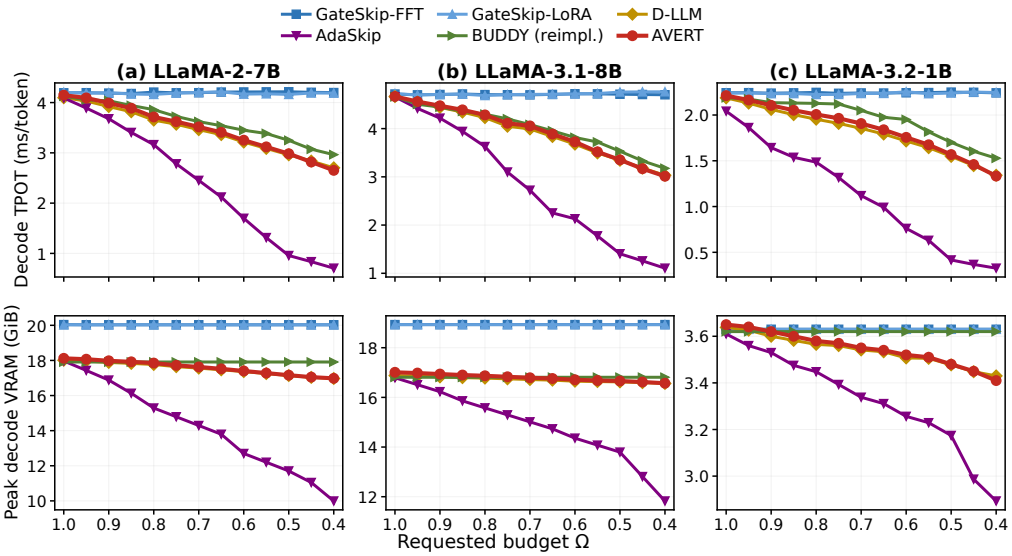

Figure 22: **Measured TPOT and peak decode VRAM across requested targets.** GateSkip-FFT and merged GateSkip-LoRA are measured separately. Equal requested targets need not imply equal realized activation or equal quality.

anchors are excluded from the conditional comparison. The supplementary CSV records all matched and unmatched anchors.

Table 6: Existing-point comparisons within 0.05 Prometheus score. Counts apply only to matched anchors and may reuse the same AVERT point; they are not independent statistical trials.

Baseline	Matched / 39	Lower TPOT	Lower peak decode VRAM
GateSkip-FFT	10	10	10
GateSkip-LoRA	15	15	15
D-LLM	25	20	13
AdaSkip	11	7	3
BUDDY (reimpl.)	15	15	15

The TPOT comparison is more consistent than the memory comparison: AVERT has lower peak decode VRAM in 13/25 matched D-LLM pairs and 3/11 matched AdaSkip pairs. For example, on LLaMA-3.1-8B, AVERT at requested $\Omega = 0.55$ scores 2.89, compared with AdaSkip's 2.85 at $\Omega = 0.85$. Their measured TPOTs are 3.519 and 3.942 ms/token, respectively, while peak decode VRAM is 16.68 and 15.86 GiB. These measured configurations give AVERT a 10.7% TPOT reduction and 5.2% higher peak decode VRAM at nearby Prometheus scores, without searching for an exact activation match. This does not establish equivalence on every quality metric.

O TOKEN-DEPTH ALLOCATION AND LAYER SETS ACROSS BUDGETS

Uncertainty and executed depth. For each requested target, we generate an AVERT continuation and replay its prefixes with the same checkpoint forced to execute all layers to obtain dense-reference next-token entropy. The depth used to process a generated token is paired with the entropy of its next-token prediction; the first two full-execution decode positions and a final position without a successor target are excluded. Entropy is an offline difficulty proxy, not a router input. Figure 23 groups positions by entropy quartile and reports activation-rate differences from Q1 in percentage points. Higher-uncertainty groups use greater mean depth in the displayed settings. This is a descriptive association; it does not require every higher-entropy token to use more layers, and continuations

can differ across requested targets. Under BUDDY's fixed explicit target, the integer Top-k count is constant across tokens even when the selected layer identities change.

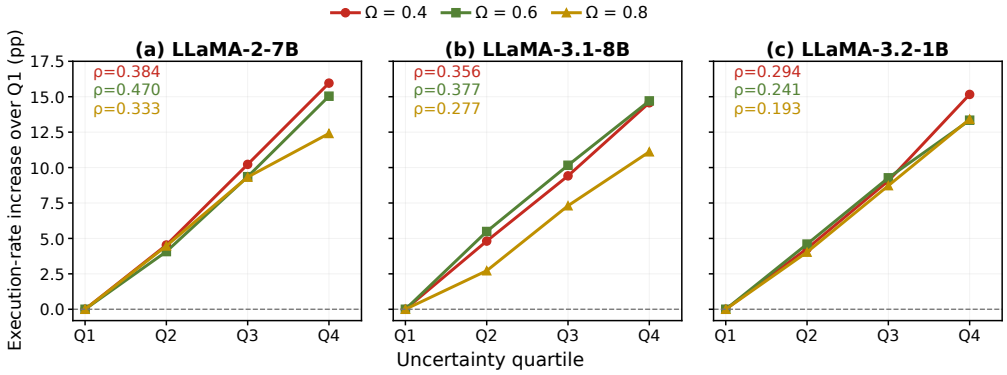

Figure 23: **Token uncertainty and execution depth.** Within each backbone and requested target, activation is shown relative to the lowest uncertainty quartile Q1. The vertical axis is a percentage-point difference, not the number of layers. ρ denotes the token-level Spearman correlation between dense-reference next-token entropy and executed depth; quartile means are shown only for visualization.

Non-nested layer selection. We teacher-force the same Alpaca reference continuations across all requested targets and record the executed sets $S_{\Omega,i}$ at each routed position i, excluding both the first two full-execution decode positions and the first two always-executed layers. For $\Omega_{\text{low}} < \Omega_{\text{high}}$, the nesting-violation rate is

$$\text{NVR} = \frac{\sum_i |S_{\Omega_{\text{low}},i} \setminus S_{\Omega_{\text{high}},i}|}{\sum_i |S_{\Omega_{\text{low}},i}|}. \tag{5}$$

Counts are pooled across positions before division. Figure 24 displays the 78 budget pairs from the 13-point grid. Positive values establish that a lower-budget route need not be contained in a higher-budget route. Text is fixed, while hidden states and layer-local cache histories may change with routing. In BUDDY's explicit-budget formulation, holding the planner features and prior fixed leaves the scores unchanged; changing only k yields nested sets with a consistent tie rule. Positive AVERT NVR shows that its layer set can change across budgets; it does not measure reversal of a common global importance ranking or explain the quality gains causally.

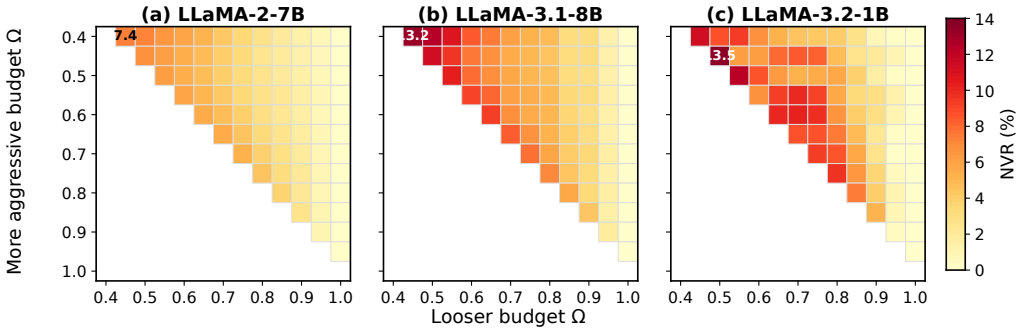

Figure 24: **Non-nested selections across requested targets.** Each cell compares the same reference text under two budgets. Nonzero NVR means that some dynamically executed low-budget layer–token pairs are absent at the higher budget.

P REDUCED QWEN2.5-3B EVALUATION

We evaluate Qwen2.5-3B *base* (36 Transformer layers) at requested targets $\Omega \in \{1.0, 0.7, 0.4\}$, comparing AVERT, our BUDDY reimplementation, and a separately finetuned Vanilla reference. This reduced sweep supplements the main LLaMA experiments and is excluded from their aggregate win counts. Generation-quality and system measurements use full-compute prefill. PIQA remains an auxiliary prefill-skipping probe; its AVERT results use the separately trained prefill-skipping checkpoint. Qwen2.5-3B is converted to the shared execution backend while retaining its native tokenizer; the supplementary code includes the converter and a parity check.

Table 7: **Qwen2.5-3B quality.** Prometheus is on the 1–5 scale; other metrics are fractions. SP/SI denote strict prompt/instruction accuracy. PIQA is the auxiliary prefill probe.

Method	Ω	Prometheus	ROUGE-L	BLEU-4	GSM8K-Flex	IFEval-SP	IFEval-SI	PIQA
Vanilla	1.00	3.460	0.2943	0.1263	0.5747	0.2680	0.3993	0.7552
AVERT	1.00	3.445	0.2948	0.1265	0.5777	0.2847	0.4065	0.7579
AVERT	0.70	3.444	0.2927	0.1253	0.4890	0.2828	0.3957	0.7383
AVERT	0.40	3.133	0.2871	0.1136	0.2115	0.2717	0.3825	0.7116
BUDDY (reimpl.)	1.00	3.289	0.2984	0.1269	0.5489	0.2699	0.3753	0.7563
BUDDY (reimpl.)	0.70	2.653	0.2589	0.0950	0.0735	0.0961	0.1918	0.6834
BUDDY (reimpl.)	0.40	1.328	0.1554	0.0263	0.0190	0.0665	0.1523	0.5506

Table 8: **Qwen2.5-3B decode efficiency.** TPOT and peak decode VRAM are measured separately from quality, with 128-token prefill and 128 decode-produced tokens. The prefill bootstrap token is excluded from the decode count.

Method	Ω	TPOT (ms/token)	Peak decode VRAM (GiB)
Vanilla	1.00	3.9012	7.77
AVERT	1.00	4.0402	7.99
AVERT	0.70	3.4792	7.78
AVERT	0.40	2.5997	7.58
BUDDY (reimpl.)	1.00	3.9311	7.80
BUDDY (reimpl.)	0.70	3.4787	7.80
BUDDY (reimpl.)	0.40	2.6533	7.80

At requested $\Omega = 0.4$, AVERT has higher scores than BUDDY on all seven reported quality metrics, with TPOT of 2.5997 versus 2.6533 ms/token and peak decode VRAM of 7.58 versus 7.80 GiB. At $\Omega = 0.7$, their TPOTs are nearly identical (3.4792 and 3.4787 ms/token). These are measured operating points, without a claim of statistical significance for small timing differences. BUDDY's budget-independent 7.80 GiB peak is consistent with retaining aligned K/V positions; these retained positions also leave later attention histories unshortened. AVERT's peak decreases from 7.99 to 7.58 GiB across the sweep.

Q SENSITIVITY TO PROMPT LENGTH

We evaluate LLaMA-3.1-8B at BS=16 with full-compute prefill lengths of 128–2048 tokens and 128 decode-produced tokens. Peak decode VRAM is measured during decoding, including resident weights and prompt K/V but excluding prefill-only temporaries. Prompts are left-truncated or attended left-padded to each target length; padded positions participate in attention, computation, and cache storage. This controlled experiment therefore measures sensitivity to physical context length rather than natural long-document quality.

Table 9: Prompt-length sensitivity of AVERT on LLaMA-3.1-8B. TPOT is in ms/token; peak decode VRAM is in GiB.

	Decode TPOT			Peak decode VRAM		
Prompt length	$\Omega = 1.0$	0.7	0.4	$\Omega = 1.0$	0.7	0.4
128	4.6728	4.0781	3.0392	17.05	16.88	16.62
256	5.2226	4.6315	3.6883	17.56	17.38	17.11
512	6.0288	5.1887	4.0898	18.57	18.38	18.11
1024	7.1113	6.0759	5.0831	20.58	20.38	20.09
2048	9.0585	7.8505	6.8457	24.57	24.35	24.05

Decode TPOT increases because each generated token attends to a longer full-compute prompt history. Peak decode VRAM increases because the longer prompt's K/V remains resident throughout decoding. Lowering Ω reduces both quantities at every tested length. In absolute terms, the TPOT reduction relative to $\Omega = 1.0$ generally becomes larger for longer prompts: it increases from 0.59 to 1.21 ms/token between 128 and 2048 prompt tokens at $\Omega = 0.7$, and from 1.63 to 2.21 ms/token at $\Omega = 0.4$. Peak-memory savings grow much less, from 0.17 to 0.22 GiB at $\Omega = 0.7$ and from 0.43 to 0.52 GiB at $\Omega = 0.4$. Because prefill always executes all layers, longer prompts add nearly the same prompt K/V memory at every target. Routing and compact K/V omission apply only to the fixed 128-token decode.

R VARIATION ACROSS TRAINING SEEDS

We independently train five additional AVERT models on LLaMA-3.1-8B with seeds $\{0, 1, 2, 3, 4\}$, starting each run from the same unmodified Meta base checkpoint; the main-results reference checkpoint is not included in these five runs. All other training and evaluation settings are held fixed. Table 10 reports the mean and sample standard deviation across the five runs at three requested budgets.

Table 10: Five independent AVERT training runs on LLaMA-3.1-8B. Entries are mean $\pm$ sample SD, not standard error. Prometheus uses the 1–5 scale; other metrics are fractions.

Metric	$\Omega = 1.0$	$\Omega = 0.7$	$\Omega = 0.4$
Prometheus-2	3.644 ± 0.034	3.394 ± 0.021	2.036 ± 0.030
ROUGE-L	0.2807 ± 0.0016	0.2644 ± 0.0018	0.1995 ± 0.0019
BLEU-4	0.1237 ± 0.0019	0.1118 ± 0.0016	0.0607 ± 0.0014
GSM8K-Flex	0.4238 ± 0.0046	0.2177 ± 0.0035	0.0206 ± 0.0035
IFEval strict-prompt	0.3619 ± 0.0058	0.2776 ± 0.0060	0.0943 ± 0.0037
IFEval strict-instruction	0.4712 ± 0.0046	0.3871 ± 0.0049	0.1892 ± 0.0055

Across the tested budgets, the sample SD is at most 0.034 for Prometheus, 0.0046 for GSM8K-Flex, and 0.0060 across the IFEval metrics. The differences between budget settings are larger than the observed variation across training seeds. The low GSM8K-Flex accuracy at $\Omega = 0.4$ is consistent across seeds, so the degradation at this aggressive budget is not specific to one training run.