Rethinking Latency Denial-of-Service: Attack the LLM Serving Framework, Not the Model

Tianyi Wang, Huawei Fan, Yuanchao Shu, Peng Cheng, Cong Wang*
Zhejiang University, Hangzhou, China
{wty1998, hw.phil.fan, ycshu, lunarheart, cwang85}@zju.edu.cn

Abstract

LLM inference is inherently expensive, even a modest slowdown can translate into substantial operating costs and severe availability risks. Recently, a growing body of research known as latency attacks focuses on crafting inputs to trigger worst-case output lengths. However, we report a contrary finding that these algorithmic-level latency attacks are largely ineffective against modern LLM serving systems. We reveal that system-level optimization such as continuous batching provides a logical isolation to mitigate contagious latency impact on co-located users. Thus, in this paper, we shift our focus from the algorithm to the system layer, and introduce a new Fill and Squeeze attack strategy targeting the state transition of the scheduler. "Fill" first exhausts the global KV cache to induce Head-of-Line blocking, while "Squeeze" forces the system into repetitive preemption. By manipulating output lengths using different attack prompts, and leveraging side-channel probing of memory status, we demonstrate that the attack can succeed in a practical black-box setting with much less cost. Extensive evaluations on vLLM indicate up to $75 - 742\times$ TTFT degradation relative to benign baselines and $1.5 - 4\times$ average slowdown on Time Per Output Token compared to existing attacks with 30-40% lower attack cost. Code: `https://github.com/Phil-Fan/FS-attack`

1 Introduction

LLM API spending has surged from $3.5B to $8.4B in the last six months [23], driven by billion-scale daily requests [34]. Meeting strict latency requirements of conversational agents [28, 3], code generation [39, 4], and document summarization [44, 54] while maximizing GPU utilization is the core challenge of modern LLM serving. As AI companies are competing to accommodate larger context windows (e.g., 1M/65K input/output tokens for Gemini 3.0 Pro [13]), a new class of Denial-of-Service threats is looming known as *latency attacks* [8, 19, 32, 11, 5, 20, 56, 59] that exploits the autoregressive nature of LLMs to suppress the end-of-sequence (`EOS`) token and trap models in indefinite babbling. Existing literature reports $7 - 10\times$ latency amplification [11] that could *theoretically* threaten service availability.

However, counter-intuitively, we first find that the existing model-centric latency attacks are surprisingly *ineffective* in production environments. Through extensive empirical and theoretical analysis, we find that simply prolonging generation length fails to inflict significant latency on co-located users, and often leaves Time-to-First-Token (TTFT) nearly unchanged even when malicious requests constitute a large fraction of the workload. To answer "*Why latency attacks fail to delay?*", we delve into the natural resilience of *Continuous Batching* (CB) [18, 52, 57, 21], a technique originally proposed to schedule requests at the iteration level so that short requests do not need to wait for the long requests to finish. This helps create a *logical isolation* between the normal traffic and the attacker's long-running request. Even if the attacker could occupy the scheduling slot, it does not prevent shorter, legitimate requests from entering and exiting the system. As a result, this relegates most latency attack as largely self-inflicted rather than contagious on co-located users [50].

*Corresponding Author

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

To make latency attacks effective again, we shift our focus to the system layer by introducing **Fill and Squeeze** (F&S), a new attack strategy targeting the deterministic state transitions of the scheduler. It can be integrated with any attack methods as pluggable payloads. The core intuition is to manipulate memory resources to trigger pathological behaviors rather than simply slow down individual generations. The proposed attack operates via two vectors: 1) *Fill*, which focuses on exhausting the VRAM rapidly. By injecting adversarial requests that generate long sequences and potentially leveraging ambient traffic peaks [41], the attacker could quickly drive the global KV cache usage to a boundary point that triggers the scheduler's admission control, inducing memory-based Head-of-Line (HOL) blocking; 2) *Squeeze*, which exploits the preemption logic. Once the system is brought to the saturation boundary, it forces the scheduler into an alternation between continuous preemption and recovery, that diverts precious GPU cycles into expensive memory swapping or recomputation operations [42, 49].

Yet, implementing this strategy in a production environment faces two challenges. The first one is the visibility gap. Different from white-box KV cache estimation [22], the attacker acts as a standard tenant with no privileged access to system internals (current KV cache occupancy). Hence, it is difficult to determine the exact timing to launch an injection and whether the injection has brought the system to the boundary proximity of GPU VRAM. Second is the economic viability. A brute-force strategy of continuously sending maximum-length requests incurs high monetary costs and low stealthiness [2, 26]. In the real world, where attackers must pay per token, an attack is only worthwhile if the damage outweighs the cost. Hence, it is necessary to come up with a more cost-effective strategy that can take advantage of the deterministic states of the serving scheduler.

To tackle these challenges, we first discover that the Inter-Token Latency (ITL) of standard requests exhibits a strong correlation with global GPU memory usage. We leverage this proxy to train a lightweight regressor [17] that allows the attacker to estimate KV usage without system privileges in black-box. Further, to minimize attack cost, we formulate the attack into a constrained optimization problem, decompose attack prompts into three different tiers measured by output lengths and design a dynamic injection strategy to switch between the Fill and Squeeze at different intensities. The results show $20 - 280\times$ average slowdown on Time to First Token (TTFT) and $1.5 - 4\times$ average slowdown on Time Per Output Token (TPOT) compared to the existing attacks.

2 Background and Related Works

2.1 LLM Serving

LLMs generate text autoregressively via prefill and decoding, in which decoding continues until an end-of-sequence [EOS] token is produced or a predefined maximum length is reached. Because each decode step must read the accumulated KV-cache, it is *memory-bound*. Production serving systems like vLLM [18], Orca [52], SGLang [57] maximize throughput via Continuous Batching (CB), which schedules requests at the iteration level [52, 31]. When any running request completes, a new waiting request is immediately admitted, preventing short requests from queuing behind long ones. When aggregated demand exceeds capacity, the scheduler either *preempts* active requests (via swap to CPU DRAM or KV recomputation) or blocks new admissions.

To handle real-world dynamic request lengths, Sarathi-Serve [1] proposes chunked prefill to split long prompts into smaller batches, piggybacking them onto decoding steps to reduce pipeline bubbles. DynaServe [27] introduces a unified execution plan that dynamically partitions resources between prefill and decode to handle unbalanced workloads. Similarly, Aqua decouples memory and compute allocation via network-accelerated memory offloading [41]. To address user fairness, VTC employs fair queuing to ensure equal resource distribution [31]. Importantly, there is a distinction between handling natural long-context traffic and malicious latency attacks. In production environments, the arrival of long requests follows a stochastic distribution (e.g., Poisson process) [31, 18, 27, 36]. Modern schedulers are robust to handle occasional congestion by swapping them to DRAM or delaying their admission until resources free up. In contrast, our work approaches LLM serving from an adversarial perspective by transforming the VRAM capacity constraint into a failure mode that standard scheduling policies (like FCFS or LIFO preemption) cannot easily resolve.

2.2 Latency Attacks

Latency attacks on AI systems evolve under the classic DoS principle to induce computational overload. The pioneering examples are the series works of Sponge Examples [32, 5, 30], which

targets NLP/vision models to deliberately inflate inference time and energy usage. Latency attacks can be broadly categorized into: optimization-based, whose objective is to postpone the generation of EOS token for as long as possible [8, 19, 10, 33]; prompt engineering-based, which exploits the model's instruction-following capability to maximize the output length [20, 56, 59]. It is achieved by constructing prompts that involve complex logical reasoning or force the model to explain multiple concepts. Although the existing attacks are theoretically sound, we demonstrate that most of these model-centric approaches struggle in production environments. In contrast, this work targets the scheduler by exploiting the resource management logic rather than optimizing the generation length.

3 Why Latency Attacks Fail under LLM Serving

3.1 Empirical Analysis

We first perform empirical analysis by establishing a multi-tenant environment on the vLLM framework (NVIDIA L40S, 48GB VRAM running Qwen3-8B [51], `gpu_memory_utilization`=0.90) with batch size 16 and varying proportions of adversarial requests.

Finding #1 (CB Naturally Defends Against Latency Attacks). As shown in Fig. 1, even with 50-75% adversarial requests using LoopLLM (a strong attack) [19], the TTFT of normal users increases by *only* $3-5.7\%$ and TTFT remains nearly flat.

This finding represents a sharp contrast to the existing works claiming over orders of magnitude of latency [11]. We attribute such natural resilience to CB, which was originally designed to improve throughput. Unlike static batching, where batch latency is dictated by the longest sequence (HOL blocking), CB dynamically schedules requests at the iteration level. An attacker's request that delays emitting an EOS token simply occupies scheduling slot for a longer duration, but it does not logically block the scheduling of shorter, legitimate requests, which can enter and exit the batch independently. Thus, model-centric latency attacks become self-inflicted rather than contagious.

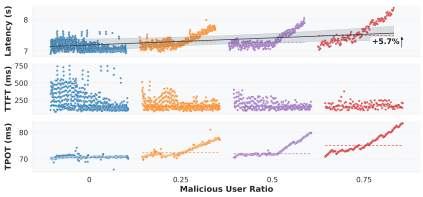


Figure 1: Impact of latency attacks on normal users.

Finding #2 (TPOT Leaks Physical Contention). While TTFT is protected, there is a consistent TPOT increase of 5ms (roughly 7%) under adversarial load in Fig. 1. CUDA profiling (Figure 2) reveals `flash_fwd_splitkv_kernel` execution time increases monotonically with aggregate KV-cache usage, while compute-bound kernels (`rms_norm`, `elementwise`) remain flat.

This phenomenon is in line with [41, 58] but from a latency-adversarial perspective. TPOT increase stems from the *memory-bound* nature of the autoregressive decoding phase. In implementations like vLLM with FlashAttention [7], the decoding kernel must stream the entire history of KV cache blocks from High Bandwidth Memory (HBM) to on-chip SRAM at every generation step. Essentially, this finding exposes an intriguing architectural insight: while CB is able to thwart latency attacks to certain extent, such logical isolation on the software-level is insufficient to enforce hardware isolation due to memory bandwidth contention. Although such minor contention is too weak to constitute a DoS attack by itself, it creates a deterministic connection between global VRAM usage and individual request latency (indicated by TPOT). This unintended information leak serves as the theoretical foundation, which plays an indispensable part in the proposed attack described next.

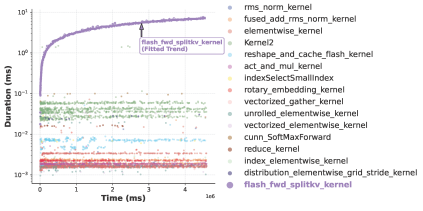


Figure 2: CUDA kernel execution under the latency attack.

3.2 Theoretical Analysis

To explain the empirical findings based on theoretical models, we perform analysis to characterize the attack boundaries. Consider a serving system with batch size B. Let r_{adv} denote an adversarial request generating L_{adv} output tokens (the attacker's goal is $L_{\text{adv}} \to \infty$), and let r_n be a co-located normal request with expected output length much shorter than L_{adv}. Detailed proofs are provided in Appendix B.2.

Proposition 1 (CB Logical Isolation). In normal cases with no memory pressure, system utilization $\rho < 1 - 1/B$ and batch size B, the expected TTFT overhead induced on r_n by r_{adv} satisfies,

$$\mathbb{E}\Big[\text{TTFT}_n^{\text{attack}}\Big] - \mathbb{E}\Big[\text{TTFT}_n^{\text{benign}}\Big] \leq \frac{T_{\text{prefill}}(r_{\text{adv}})}{B}, \tag{1}$$

where $T_{\text{prefill}}(r_{\text{adv}})$ is the prefill latency of r_{adv}. Consequently, the TTFT penalty is independent of L_{adv} when the system is stable and memory-unconstrained.

Corresponding to Finding #1, Proposition 1 shows that model-centric latency attacks, which target output length L_{adv}, cannot increase the average TTFT of co-located users beyond a small scheduling overhead, regardless of how many tokens the attacker generates: once r_{adv} enters the running queue it occupies one out of B slots, but CB continuously replenishes the remaining $B - 1$ slots with waiting requests. Hence r_{adv}'s prolonged decode phase does not logically block r_n. Corresponding to Finding #2, since co-located requests still compete for shared HBM bandwidth during KV-cache access, we derive the relation between Inter-Token Latency (ITL) and KV cache utilization.

Proposition 2 (ITL-KV Correlation). Let F_t denote the total active KV state referenced across all running requests at decode step t. In memory-bound regime, the per-step decode latency T_t is approximately affine in aggregate KV utilization,

$$T_t \;\approx\; a \cdot \frac{F_t}{BW_{\text{HBM}}} + b, \;\; \text{where} \;\; a = 2 \cdot n_{\text{layers}} \cdot n_{\text{kv_heads}} \cdot d_{\text{head}} \cdot \text{sizeof}(T). \tag{2}$$

a captures the per-token KV memory access cost (both keys and values, across all layers and KV heads, T is data type), and b captures compute-bound overhead independent of context length.

Proposition 2 explains Finding #2 and is further validated in Appendix E. The attention kernel (`flash_fwd_splitkv_kernel`) must stream the full KV-cache for all active contexts from HBM to on-chip SRAM at every generation step. As aggregate KV usage grows, this memory traffic saturates HBM bandwidth, making decode latency a proxy for global memory pressure. This is confirmed by CUDA profiling (Fig. 2): the attention kernel exhibits monotonic growth with KV usage while compute-bound kernels (`rms_norm`, `elementwise_ops`) remain flat. It lays the foundation for launching the attack in black-box settings discussed next.

Proposition 3 (FCFS-Induced HOL Blocking). Consider a typical FCFS admission queue $\mathcal{Q}_t = (q_1, q_2, \cdots, q_m)$ and $r(q_i)$ as the blocks required to admit request q_i and F_t as the current free block count. Assume $r(q_1) > F_t$, but there exists q_j, $j > 1$ with $r(q_j) \leq F_t$. If the scheduler does not perform skip-ahead admission, then no request q_j can be admitted until $F_t \geq r(q_1)$ or q_1 is removed. The minimum blocking duration for every later request is $W_j \geq \tau(q_1, t) - t$, where $\tau(q_1, t)$ is the first future time when free blocks reach $r(q_1)$. The attacker drives to meet this condition by accumulating,

$$\sum_{r \in \texttt{RUNNING}} \texttt{KV_blocks}(r) \geq \texttt{total_blocks} - M_{\text{req}}. \tag{3}$$

where $\texttt{KV_blocks}(r)$ is the number of physical blocks allocated to request r, `total_blocks` is the total GPU pool size and M_{req} is the blocks required for the HOL waiting request.

Corollary 3 (Preemption Cost). When the scheduler preempts a victim request r_v of current sequence length L_v, the recovery cost satisfies,

$$T_{\text{recover}}(r_v) = \begin{cases} T_{\text{prefill}}(L_v), & \text{recomputation recovery,} \\ S_{\text{KV}}(L_v) \;/\; BW_{\text{PCIe}}, & \text{swap recovery,} \end{cases} \tag{4}$$

where $T_{\text{prefill}} \approx \Theta(L_v^2)$ represents standard transformer where attention computation dominates, and the swapped KV state has size $S_{\text{KV}}(L_v) = 2L_v \cdot n_{\text{layers}} \cdot n_{\text{kv_heads}} \cdot d_{\text{head}} \cdot \text{sizeof}(T)$. The attacker's marginal cost to trigger preemption by adding δ tokens near the saturation boundary is only $O(\delta \cdot \text{price_per_token})$, yielding an asymmetry ratio $T_{\text{recover}}/C_{\text{trigger}} \in \Omega(L_v/\delta)$.

Proposition/Corollary 3 together provide a foundation for the proposed F&S Attack from the cost perspective. The attacker could toggle the system across saturation boundary with minimal tokens, and force the system to repeatedly pay the full recovery cost T_{recover} for each preemption event, while their own injection cost scales only with δ.

4 Attack LLM Serving Framework with Fill and Squeeze

We propose Fill and Squeeze (F&S) attack that targets the LLM scheduler's state machine rather than the model's generation process. The key observation is that the LLM serving scheduler only

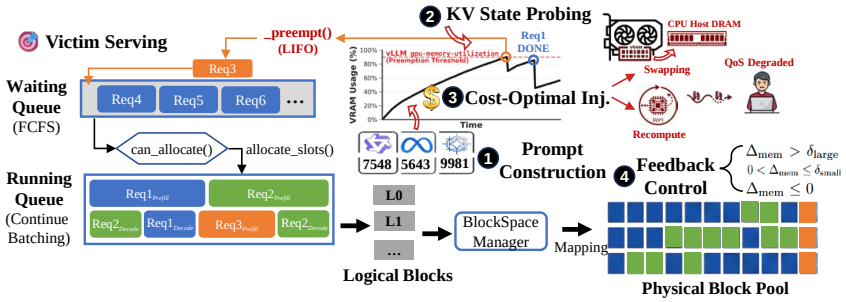


Figure 3: Overview of F&S attack steps and scheduling states, in which a key is the transition between the FCFS-based `Waiting Queue` and the continuously-batched `Running Queue`.

has a few stable states: an IDLE state with sufficient free KV-cache capacity, a **SATURATED** state where memory-based HOL blocking becomes active, and a **THRASHING** state where the system oscillates near the saturation boundary due to repeated preemption and recovery. F&S drives the system from IDLE into THRASHING through: **Fill**, which rapidly depletes available KV-cache blocks, and **Squeeze**, which adds just enough load near the boundary to repeatedly trigger costly preemption. E.g., in vLLM [18], the transition is between the FCFS-based `WAITING` queue and the continuously batched `RUNNING` queue. A more detailed interpretation is available in Appendix B.3.

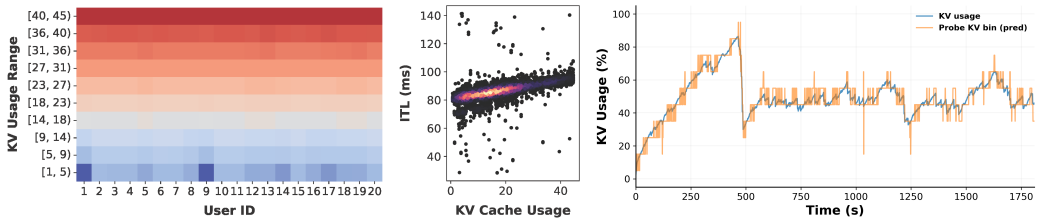
The attack operates in four phases: ❶ payload-agnostic prompt construction, ❷ black-box KV state probing, ❸ cost-optimal injection, and ❹ adaptive feedback control. These phases jointly enable the attacker to estimate the current KV-cache state, select the cheapest prompt combinations that can cross the saturation boundary, and avoid self-preemption while sustaining disruption.

❶ Payload-Agnostic Prompt Construction. A primary objective of the attacker is to utilize tiered prompts to save attack cost and avoid detection, i.e., $\mathcal{P} = \{\mathcal{P}_{\text{High}}, \mathcal{P}_{\text{Mid}}, \mathcal{P}_{\text{Low}}\}$ and maximized **Expansion Ratio** $L_{\text{out}}/L_{\text{in}}$. $\mathcal{P}_{\text{High}}$ fills the KV cache for rapid depletion `free_block_queue` and $\mathcal{P}_{\text{Low}}$ tips an already saturated scheduler into preemption, which is also used for probing. $\mathcal{P}_{\text{Mid}}$ sustains memory pressure during transitional states. Our method supports any pluggable payload (different latency attacks) that can induce long outputs. In this work, we instantiate it with plain-text prompts for black-box transferability [25, 9] and ExtendAttack [59] to trigger extended reasoning.

❷ Black-Box KV-Cache Estimation via ITL Probing. A practical challenge of launching F&S attack is to know the system KV-cache utilization in real-time. Prior work such as Helix [22] assumes privileged access to request output lengths and historical statistics. However, in black-box adversarial settings, the attacker is also an external tenant and lacks direct visibility into the global scheduler state. Let U_{sys} denote the fraction of used KV-cache blocks over total blocks. Based on Finding #2 and Proposition 2, we could infer U_{sys} through the ITL-based side channel induced. Operationally, the attacker periodically dispatches probe requests $p_{\text{probe}} \in \mathcal{P}_{\text{Low}}$ and records their ITL sequence $\mathbf{t} = (t_1, t_2, \ldots, t_k)$ for estimation.

It turns out that the attacker can use a lightweight classifier, $\mathcal{M}_\theta : \mathbb{R}^k \to [0, 1]$ to map the ITL sequence to discrete KV-usage bins with high accuracy in real-time. For instance, we can instantiate $\mathcal{M}_\theta$ with LightGBM [17]. Compared with sequence models, LightGBM achieves higher accuracy and substantially lower inference latency, which allows the attacker to synchronize injection timing with the scheduler state at sub-second granularity. As shown in Fig. 4c, the predicted KV usage consistently tracks the ground truth across both gradual accumulation and abrupt memory release. The low variance in Fig. 4b further indicates that the probe is robust to stochastic scheduling noise.

❸ Cost-Optimal Injection Strategy. With the set of tiered prompt $\mathcal{P}$ and the estimator $\mathcal{M}_\theta$, the attacker selects an injection policy that minimizes time-averaged cost while keeping the system at or above the saturation boundary C_{sat} identified in Sec. B.3. Let $p_t \in \mathcal{P}$ denote the prompt selected at time t, and let $L_{\text{in}}(p_t)$ and $L_{\text{out}}(p_t)$ denote its input and generated output lengths. We adopt industrial cost model, Cost $= \alpha \cdot L_{\text{in}} + \beta \cdot L_{\text{out}}$ [24]. Given unit input and output token prices α and β (e.g., $\alpha = \$0.50$, $\beta = \$1.50$ per 10^6 input/output tokens), F&S can be formulated as a cost



(a) Per-User Trends (b) ITL Heatmap (c) Tracing the KV-Cache Usage Prediction

Figure 4: Black-box estimation of KV usage on Qwen3-8B. (a) Similar latency imposed on all users due to shared memory contention; (b) strong linear correlation between ITL and global KV-cache usage (U_{sys}); (c) trace of KV-cache estimation accuracy using LightGBM.

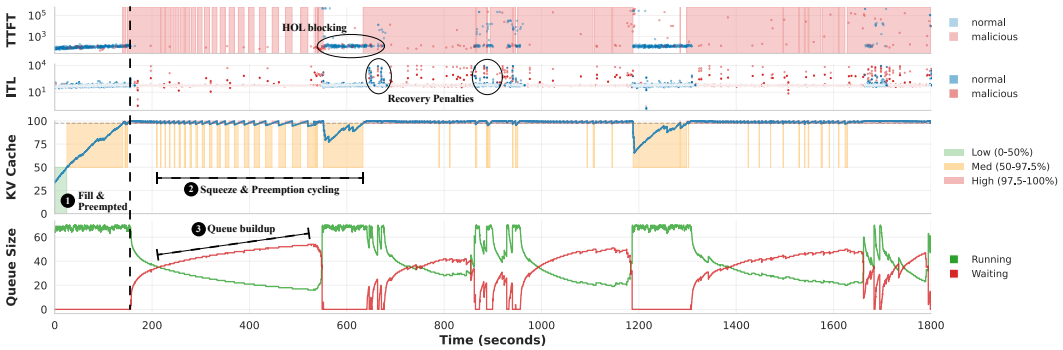


Figure 5: Visualization of TTFT, ITL, Global KV-Cache usage and queue size of F&S attack over time. The Waiting Queue Size (bottom) accumulates as opposed to the Running Queue, which directly correlates with the massive outliers observed in the TTFT Scatter Plot (top right), where user requests are blocked for tens of seconds before processing begins, confirming that the TTFT surge stems from HOL blocking.

minimization problem,

$$\mathbf{P1}: \quad \min_{p_1,\ldots,p_T \in \mathcal{P}} \quad \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \left(\alpha L_{\text{in}}(p_t) + \beta L_{\text{out}}(p_t)\right) \\ \text{s.t.} \quad \mathcal{M}_\theta(\mathbf{t}_t) + \mathcal{E}(p_t) \geq C_{\text{sat}} + \delta, \quad \forall t \in [1, T], \tag{5}$$

where $\mathcal{M}_\theta(\mathbf{t}_t)$ is the estimated current KV-cache usage from the latest ITL vector, $\mathcal{E}(p_t)$ is the expected memory expansion of prompt p_t, and $\delta > 0$ is a safety margin ensuring that the system enters the saturation region. To see why our tiered strategy is reasonable, let $\gamma = C_{\text{High}}/C_{\text{Low}} - 1$ denotes the cost ratio between high- and low-intensity prompts. There are two cases: 1) If the system is idle, i.e., $U_{\text{sys}} < C_{\text{sat}} - \delta_{\text{large}}$, any feasible policy must fill a large memory gap $\Delta_{\text{mem}} = C_{\text{sat}} - U_{\text{sys}}$. In this regime, $\mathcal{P}_{\text{High}}$ is the cheapest way to add sufficient memory pressure in one shot due to its large expansion ratio; 2) If the system is near saturation, i.e., $U_{\text{sys}} \in [C_{\text{sat}} - \delta_{\text{small}}, C_{\text{sat}})$, the constraint in Eq. (5) can be satisfied using $p \in \mathcal{P}_{\text{Low}}$, since its expected expansion is sufficient to cross the threshold. As shown in our real-world experiments, under ambient traffic, the attack becomes cheaper during peak-load periods as he only needs to bridge the remaining gap to saturation with low-cost prompts.

❹ **Adaptive Feedback Control**. To implement the optimization strategy, F&S adopts a closed-loop feedback controller and the attack process is visualized in Fig.5. The attacker repeatedly probes the system state, estimates the distance to saturation, and chooses between Fill, Squeeze, and Back-off:

Fill. When $\Delta_{\text{mem}} > \delta_{\text{large}}$, the system is relatively idle. The attacker dispatches high-intensity prompts from $\mathcal{P}_{\text{High}}$ to rapidly consume the `free_block_queue`. This pushes the scheduler toward the HOL-blocking condition. This corresponds to the Fill ramp in Fig.5, where the global KV-Cache usage rises rapidly and remains pinned to the High (97.5–100%) band.

Squeeze. When $0 < \Delta_{\text{mem}} \leq \delta_{\text{small}}$, the system is parked near saturation. The attacker switches to low-intensity prompts from $\mathcal{P}_{\text{Low}}$, which impose just enough memory pressure to cross the saturation threshold. This triggers preemption while minimizing the attacker's own token cost and the system

pays a recovery cost through swap or recomputation. In Fig.5, it is confirmed by the dense red dots of preempted events in the top subplots and continuous sawtooth patterns on the KV-Cache usage.

Adaptive Back-off. When $\Delta_{\text{mem}} \leq 0$, the system is already beyond the saturation boundary and may preempt the most recently admitted running request. If the attacker's own requests are among the most recent, they risk being selected as victims (*self-preemption*). The back-off mechanism therefore pauses injection, allowing the system to spend cycles recovering a legitimate request before the attacker resumes from a slightly below-saturation state. The overall pseudocode is detailed in Appendix 1.

5 Experiments

5.1 Threat Model

We consider an LLM inference service using default vLLM v0.11.2 [18]: a shared global KV pool, FCFS admission (no skip-ahead), LIFO preemption, no per-tenant KV quota, and client-visible streaming timestamps, with no input filtering applied [16, 55, 2]. The attacker is a black-box tenant with only public API access and no knowledge of model parameters or runtime internals; in our usage "black-box" means no victim-side access, not that a single frozen estimator transfers across hardware, and the ITL probe of Sec. 4 is trained on a matching surrogate. Using standard concurrency, the attacker synchronizes long-context queries; when rate limits are enforced per account, multiple identities keep aggregate pressure up, and this still does not bypass entity-, payment-, or global-level enforcement. Unlike prior self-inflicted latency attacks [8, 19, 59, 56], our goal is to degrade service for co-located *benign* users through hardware contention that continuous batching cannot isolate. F&S targets the setting above and is not a claim of universal bypass: strict entity-level quotas, skip-ahead admission, fair queuing, output buffering, and hard length caps each weaken it in a different way (Sec. 6).

5.2 Experimental Setup

Serving Environment. We conduct experiments on a server equipped with $8\times$ NVIDIA L40S GPUs (48GB VRAM) powered by vLLM v0.11.2 [18]. We use mainstream open-source model families including Qwen3-8B [51], Gemma3-12B-it [38] and DeepSeek-R1-Distill-Llama-8B [14], and set the `gpu_memory_utilization` to 0.90 to allocate 90% of the VRAM for KV-Cache and model weights. This setup represents a standard production node and the analysis can be easily extended to large-scale GPU clusters. More setup details are available in Appendix C.

Workload and Datasets. Following [31, 18, 27, 36], we first emulate normal traffic with Poisson distribution for the main results (Sec. 5.3) and adopt Alpaca [37] as the primary dataset, since its consistent instruction-following format provides a stable baseline for evaluating core performance metrics. Additionally, we also utilize ShareGPT dataset [6] for ablation studies.

Baselines. We benchmark our attack against Engorgio [8], LoopLLM [19] and ExtendAttack [59]. These baselines primarily function as algorithmic complexity attacks that are oblivious of the scheduler's state. For F&S strategy, we utilize plain-text prompts and ExtendAttack as the underlying methods for cross-model transferability and maximum attack strength, respectively. For fair comparison, we deploy the baselines using a fixed-interval injection. In each slot, the adversarial prompt is randomly sampled from the adversarial prompt collections. This represents the baseline attackers that are not aware of the system state and relevant cost. Note that our study is based on the existing fixed-schedule attacks, rather than adaptive wrappers on the existing schemes. The component ablation in Appendix D.1 further demonstrates that merely sending adaptively is still insufficient.

Evaluation Metrics. Attack effectiveness is evaluated via the following metrics. For user-perceived latency, we first examine TTFT. This metric represents the initial wait time and an extended TTFT implies failure to acknowledge user inputs due to queue congestion. Second, we monitor TPOT to assess generation quality. Its degradation represents non-smooth interaction that causes the model to get stuck or hang on generating subsequent tokens. In addition, we also measure the 99th percentile (P99) latency for both TTFT and TPOT. It captures the tail-end user experience, representing the worst-case delay felt by the vast majority of users. To measure the system burden, we track the number of *preemption* (cumulative `_preempt()` invocations), which quantifies the frequency of scheduler thrashing and indicates additional overhead of recomputation or memory swap. All main

Table 1: Comparison of attack effectiveness. ↑ and ↓ denote the higher/lower the better, respectively. TTFT/TTFT P99 measured in seconds (**s**), TPOT/TPOT P99 measured in milliseconds (**ms**). Each cell is the mean over five independent runs.

Target Model	Attack Method	TTFT (↑)	TTFT P99 (↑)	TPOT (↑)	TPOT P99 (↑)	Preempt#	Attack Request# (↓)	Cost ($) (↓)
Qwen3-8B	Benign (No Attack)	0.15	0.27	34.78	49.66	0	0	—
	Engorgio	0.12	0.24	48.17	59.51	0	4613	1.18
	LoopLLM	1.80	58.05	114.12	1249.20	1104	295	0.80
	ExtendAttack	0.85	11.90	80.03	474.01	538	529	0.92
	F&S+plain-text	11.05	400.13	194.45	2159.74	618	180	0.75
	F&S+ExtendAttack	35.95	408.86	181.93	991.19	524	73	0.42
Gemma3-12B-it	Benign (No Attack)	0.13	0.21	68.00	76.28	0	0	—
	Engorgio	0.13	0.20	68.26	77.22	0	2221	0.83
	LoopLLM	3.46	36.83	126.04	193.95	3414	899	0.71
	ExtendAttack	10.55	52.71	163.50	680.00	3179	684	0.68
	F&S+plain-text	97.83	167.73	244.29	3544.80	1367	302	0.67
	F&S+ExtendAttack	104.50	178.36	158.57	1453.84	1380	555	0.46
DeepSeek-R1-Distill-Llama-8B	Benign (No Attack)	0.08	0.14	37.51	41.15	0	0	—
	Engorgio	0.10	0.14	48.86	55.80	0	2027	1.17
	LoopLLM	1.07	0.56	57.37	115.30	89	992	1.06
	ExtendAttack	13.29	220.58	188.86	1021.82	773	168	0.85
	F&S+plain-text	5.31	110.27	89.52	478.90	1098	321	0.95
	F&S+ExtendAttack	16.08	284.41	157.80	1215.48	506	183	0.86

and ablation results are averaged over five independent runs: TTFT/TPOT means and P99 values are first computed over benign requests within each run, preemptions, attack requests and cost are per-run totals, and each reported value is the mean of these run-level statistics. Note that due to heavy-tail latency, certain P99 values (e.g., DeepSeek-R1 with LoopLLM in Table 1) may appear lower than the mean, which is possible when the tail extends beyond the 99th percentile.

5.3 Main Results

First, we analyze the attack effectiveness on a single GPU and scale up to multiple GPUs later. Table 1 demonstrates that our strategy achieves a superior balance between attack efficacy and economic cost compared to the baselines. We can see that the existing scheduler-oblivious attacks rarely drive the system into HOL blocking. In contrast, our state-aware strategy consistently pushes the system to the saturation boundary by maximizing HOL events, yielding catastrophic TTFT inflation. E.g., when paired with plain-text prompts (F&S+plain-text), we achieve a significant $75\times$ TTFT slowdown on Qwen3-8B (0.15 s $\rightarrow$ 11.05 s), and $742\times$ (0.13 s $\rightarrow$ 97.83 s) on Gemma3-12B-it. Meanwhile, F&S+ExtendAttack magnifies degradation even further. E.g., on Qwen3-8B, TTFT further increases to 35.95 s. Both methods carry much lower cost compared to the baselines.

5.4 Ablation Study

Attack Intensity. To quantify how attack efficacy and economic cost evolve with attack intensity, we vary the percentage of malicious users while keeping the total request rate constant. Fig. 6 reveals a clear separation: across model architectures, baseline methods exhibit a gradual TTFT increase as the malicious ratio grows, whereas our method causes a faster performance degradation. Interestingly, the jump of TTFT does not necessitate an equivalent increase of attack cost. Cost evaluations on Fig. 6 (right) reveal that baseline methods incur costs that scale linearly with attack intensity, because *F&S* employs low-intensity prompts during "Squeeze", thereby continuously strengthening the attack at a much lower monetary cost. Empirical results on Gemma3-12B-it show that *F&S* achieves a $733.9\times$ relative QoS degradation at 73.3% of the baseline cost.

Table 2: Workload sensitivity of F&S attack strategy. Δ indicates degradation ratio. TTFT/TPOT in s/ms.

Model	Workload	TTFT	ΔT	TPOT	ΔP
Qwen3-8B	Alpaca	11.05	75.6×	194	5.6×
	ShareGPT	2.57	41.0×	254	7.3×
Gemma3-12B-it	Alpaca	97.83	742.8×	244	3.6×
	ShareGPT	65.89	720.6×	359	5.3×
DeepSeek-R1-D-Llama-8B	Alpaca	5.31	68.2×	90	2.4×
	ShareGPT	5.28	67.8×	80	2.1×

Impact of Workload Distributions. We further evaluate the impact of workload characteristics on F&S attack. Table 2 demonstrates the results across two contrastive profiles: Alpaca (short-context instruction following) [37] and ShareGPT (long-context dialogue) [6]. It reveals an intriguing phenomenon that challenges the prevailing assumption that long-context traffic is more vulnerable. Notably, the most acute *TTFT* degradation frequently occurs under short-context workloads. E.g., on

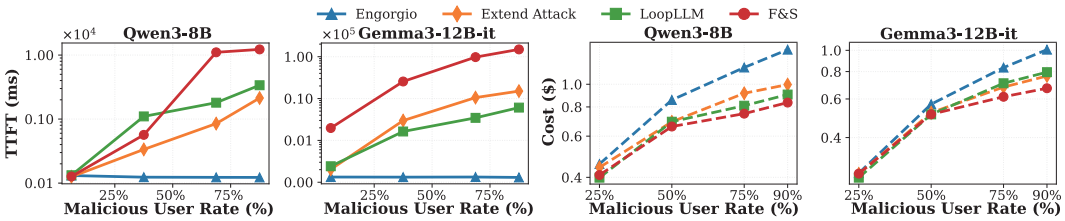


Figure 6: Impact of malicious user rate on attack efficacy and cost across different model architectures.

Qwen3-8B, Alpaca exhibits a 75.6× TTFT degradation (11.05 s), far exceeding the 41.0× observed on ShareGPT (2.57 s), and Gemma3-12B-it follows a similar trajectory. In contrast, DeepSeek-R1 remains relatively unchanged across both distributions (68.2× vs. 67.8×), suggesting an architectural stability to workload-induced variance.

Further, TTFT and TPOT exhibit divergent sensitivities to these workload shifts. While Alpaca triggers higher TTFT inflation on Qwen3-8B and Gemma3-12B-it, ShareGPT consistently incurs more severe TPOT degradation (e.g., 7.3× vs. 5.6× on Qwen3-8B). This result aligns with the two-stage design of F&S: *Fill* acts as an admission bottleneck that dominates TTFT by inducing queuing delays, whereas the sustained decoding phase of long-context dialogues amplifies the interference once the system is saturated (occupying TPOT). These findings highlight a critical risk in production environments that high-throughput, lightweight workloads can suffer catastrophic responsiveness stalls or even timeouts, while long-context sessions experience persistent throughput erosion. Hence, designers of the scheduling logic should consider diverse request distributions to be prepared for adversarial, extreme situations.

5.5 Realistic Production Scenarios

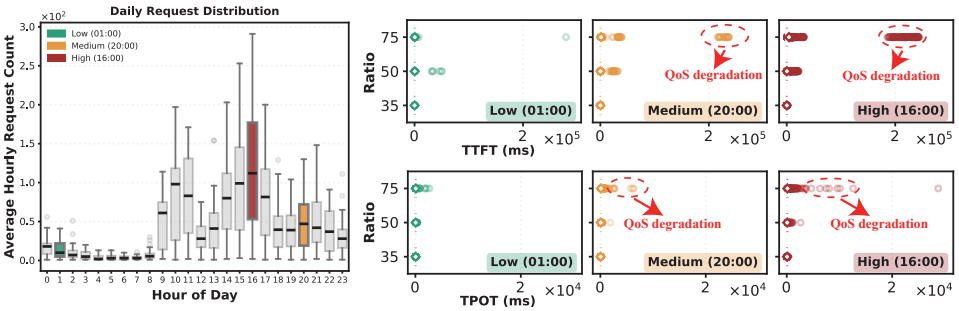
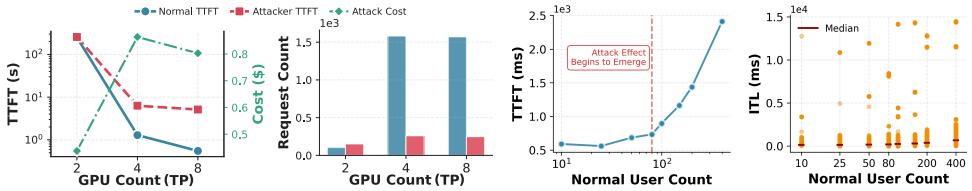


Figure 7: Attack impact on normal users under real-world traffic patterns. Left: Hourly request distribution from BurstGPT traces with three representative traffic intensities. Right: TTFT and TPOT distributions for normal users under varying malicious ratios.



(a) Multi-GPU impact (b) User load impact

Figure 8: Multi-GPU scalability analysis under varying hardware resources and load configurations.

Real-World Workload. To evaluate the F&S attack under realistic deployment conditions, we emulate real-world traffic patterns based on the BurstGPT dataset [45] (10.31M real traces). We categorize the workload into three representative time windows as highlighted by different colors on Fig. 7 (Left). The results indicate that the attacker could take advantage of ambient traffic patterns to reduce cost substantially, and ambient traffic acts as a *force multiplier*. By naturally consuming the KV-Cache headroom, the normal, benign requests lower the saturation barrier that allows the attacker to succeed more easily with 40% less cost. Visualized in Fig. 7 (Right), this results in volatility for both TTFT and TPOT during high traffic.

Multi-GPU Tensor Parallelism. Intuitively, a natural defense against resource exhaustion is over-provisioning. By scaling up resources, service providers aim to raise the bar for the attackers. Thus, to evaluate how F&S scales against horizontal scaling, we fix the attack strengths and examine its performance across varying degrees of Tensor Parallelism (TP) on the $8\times$ NVIDIA L40S server. As expected, expanding from TP2 to TP4 significantly mitigates user-perceived latency (TTFT) by raising the memory saturation barrier shown in Fig. 8(a). The expanded KV-Cache capacity prevents the attacker from successfully triggering the *Squeeze* phase (repetitive preemption), forcing the adversary to remain stuck in the capital-intensive "Fill" phase, continuously dispatching expensive High-Intensity prompts (P_{High}) without inducing thrashing. TP8 provides ample resources that allow benign users to aggressively occupy scheduling slots, effectively starving the attacker of admission and neutralizing the threat at a lower total cost. Other vLLM parameters primarily shift the attack cost profile rather than its feasibility. E.g., `block_size` controls allocation granularity near saturation; preemption mode (recomputation vs. swap) and `swap_space` determine the per-preemption recovery cost, and the scheduling policy dictates whether FCFS-HOL exists. A detailed parameter-to-mechanism mapping is provided in Appendix G.3.

Generalization to Reasoning. Table 3 reports the attack efficacy on Qwen3-8B and DeepSeek-R1-Distill-Llama-8B with reasoning tasks. While traditional baselines like *Engorgio* incur high costs ($1.23) with negligible impact (TTFT $\approx$ 0.10s), our results highlight the composability of the F&S strategy. Specifically, combining F&S with *ExtendAttack* consistently amplifies performance degradation across both architectures. On Qwen3-8B, this hybrid approach pushes TTFT to 27.65 s and TPOT to 196.99 ms, surpassing *ExtendAttack* alone (20.11 s / 191.0 ms) while reducing the financial cost to $0.68. This enhancement is equally evident on DeepSeek-R1, where F&S boosts the TTFT of *ExtendAttack* from 10.99 s to 13.24 s.

Table 3: Attack effectiveness and cost analysis under reasoning.

Model	Method	TTFT	TPOT	Cost($)
Qwen3-8B	Engorgio	0.10	45.9	1.23
	ExtendAttack	20.11	191.0	0.75
	LoopLLM	2.11	64.7	0.95
	F&S+plain-text	21.32	135.2	0.78
	F&S+ExtendAttack	27.65	196.99	0.68
DeepSeek*	Engorgio	0.10	50.19	1.13
	ExtendAttack	10.99	190.96	0.70
	LoopLLM	0.16	58.02	1.00
	F&S+plain-text	1.10	67.47	0.93
	F&S+ExtendAttack	13.24	142.34	0.74

6 Conclusion and Future Work

In this paper, we show that existing model-centric latency attacks only impose minor performance impact on modern serving systems, because continuous batching isolates co-located requests at the iteration level. We therefore target the scheduler instead. By exploiting the ITL side channel and tiered prompts, F&S pushes the shared KV cache into saturation and repeated preemption, and inflicts much higher latency on co-located users at lower attack cost.

On the other hand, simple defense such as per-account rate or concurrency limits only slow one Fill instance when requests are spread across identities; a KV quota enforced per entity below the saturation threshold keeps that entity from inducing Fill, and hard output caps take away the retained KV that Fill needs. Skip-ahead admission, fair queuing [31], and priority scheduling [15] remove most FCFS head-of-line blocking, though aggressive fairness can starve legitimate long-context users, and KV held by running requests still sustains the TPOT slowdown from Squeeze. If the provider buffers streaming output, the ITL probe loses its signal and F&S falls back to a less efficient open-loop attack. Perplexity filters [2, 26] catch optimization-based prompts, but the plain-text prompts used by F&S read as ordinary text and pass through. Thus, we leave a much larger scale evaluation of F&S-type of resource consumption attack and defense to the future work.

Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers and the area chair for their detailed feedback, in particular the questions on threat-model scope, provider-side safeguards, and statistical reporting, which led to the added component ablation, production prober evaluation, and countermeasure discussion. This work is supported by National Science Foundation of China under grants 62576310, Zhejiang Provincial National Science Foundation of China under Grant No. LZ25F020007, ICT2026A20, Fong Shu Fook Tong Research Innovation Program, the Fundamental Research Funds for the Central Universities 226-2026-00070, and the ZJUCSE-Hikvision Joint Laboratory.

References

[1] 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)*, pages 117–134, 2024.

[2] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv preprint arXiv:2308.14132*, 2023.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[4] et. al. Chen, Mark. Evaluating large language models trained on code. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[5] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374, 2022.

[6] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, March 2023.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[8] Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. An engorgio prompt makes large language model babble on. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Erin Feiglin, Nir Hutnik, and Raz Lapid. Benchoverflow: Measuring overflow in large language models via plain-text prompts. *arXiv preprint arXiv:2601.08490*, 2026.

[10] Jiyuan Fu, Kaixun Jiang, Lingyi Hong, Jinglun Li, Haijing Guo, Dingkang Yang, Zhaoyu Chen, and Wenqiang Zhang. Lingoloop attack: Trapping mllms via linguistic context and state entrapment into endless loops. *arXiv preprint arXiv:2506.14493*, 2025.

[11] Kuofeng Gao, Yang Bai, Jindong Gu, Shu-Tao Xia, Philip Torr, Zhifeng Li, and Wei Liu. Inducing high energy-latency of large vision-language models with verbose images. In *The Twelfth International Conference on Learning Representations*, 2024.

[12] Google. Rate limits — Gemini API documentation. `https://ai.google.dev/gemini-api/docs/rate-limits`. Accessed: 2026-02-03.

[13] Google Cloud. Gemini 3.0 pro. `https://ai.google.dev/gemini-api/docs/gemini-3`. Accessed: 2026-04-28.

[14] Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[15] Azam Ikram, Xiang Li, Sameh Elnikety, and Saurabh Bagchi. Ascendra: Dynamic request prioritization for efficient llm serving. *arXiv preprint arXiv:2504.20828*, 2025.

[16] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*, 2023.

[17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30, 2017.

[18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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*, pages 611–626, 2023.

[19] Xingyu Li, Xiaolei Liu, Cheng Liu, Yixiao Xu, Kangyi Ding, Bangzhou Xin, and Jia-Li Yin. LoopLLM: Transferable energy-latency attacks in LLMs via repetitive generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31770–31777, 2026.

[20] Xinyu Li, Tianjin Huang, Ronghui Mu, Xiaowei Huang, and Gaojie Jin. Pot: Inducing overthinking in llms via black-box iterative optimization. *arXiv preprint arXiv:2508.19277*, 2025.

[21] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, 2023.

[22] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 586–602, 2025.

[23] Menlo Ventures. 2025 mid-year llm market update. `https://menlovc.com/perspective/2025-mid-year-llm-market-update/`, June 2025. Accessed: 2026-04-29.

[24] OpenAI. Openai api pricing. `https://openai.com/api/pricing/`, 2024. Accessed: 2026-01-20.

[25] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[26] Mansi Phute, Alec Helbling, Matthew Hull, ShengYun Peng, Sebastian Szyller, Cory Cornelius, and Duen Horng Chau. Llm self defense: By self examination, llms know they are being tricked. *arXiv preprint arXiv:2308.07308*, 2023.

[27] Chaoyi Ruan, Yinhe Chen, Dongqi Tian, Yandong Shi, Yongji Wu, Jialin Li, and Cheng Li. Libra: Flexible request partitioning and scheduling for serving unbalanced and dynamic LLM workloads. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 1243–1258, Renton, WA, May 2026. USENIX Association.

[28] John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, Jacob Menick, Jiayi Weng, Juan Felipe Ceron Uribe, Liam Fedus, Luke Metz, Michael Pokorny, et al. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2(4), 2022.

[29] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

[30] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4571–4580, 2023.

[31] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 965–988, 2024.

[32] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, pages 212–231. IEEE, 2021.

[33] Wai Man Si, Mingjie Li, Michael Backes, and Yang Zhang. Excessive reasoning attack on reasoning llms. *arXiv preprint arXiv:2506.14374*, 2025.

[34] Amanda Silberling. Chatgpt users send 2.5 billion prompts a day, July 2025. TechCrunch.

[35] Linke Song, Zixuan Pang, Wenhao Wang, Zihao Wang, XiaoFeng Wang, Hongbo Chen, Wei Song, Yier Jin, Dan Meng, and Rui Hou. The early bird catches the leak: Unveiling timing side channels in llm serving systems. *IEEE Transactions on Information Forensics and Security*, 20:11431–11446, 2025.

[36] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 173–191, 2024.

[37] 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. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[38] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[39] Weixi Tong and Tianyi Zhang. Codejudge: Evaluating code generation with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20032–20051, 2024.

[40] U.S. Energy Information Administration. Electric power monthly: Average price of electricity to ultimate customers by end-use sector, 2024. Data for Virginia (Industrial Sector). Accessed: 2024-05-20.

[41] Abhishek Vijaya Kumar, Gianni Antichi, and Rachee Singh. Aqua: Network-accelerated memory offloading for llms in scale-up gpu domains. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 48–62, 2025.

[42] vLLM Documentation. Optimization configuration. `https://docs.vllm.ai/en/stable/configuration/optimization/`, 2026. Accessed: 2026-01-27.

[43] Tianyi Wang, Zichen Wang, Cong Wang, Yuanchao Shu, Ruilong Deng, Peng Cheng, and Jiming Chen. Can't slow me down: Learning robust and hardware-adaptive object detectors against latency attacks for edge devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19230–19240, June 2025.

[44] Yiming Wang, Zhuosheng Zhang, and Rui Wang. Element-aware summarization with large language models: Expert-aligned evaluation and chain-of-thought method. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8640–8665, 2023.

[45] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5831–5841, 2025.

[46] Roy Weiss, Daniel Ayzenshteyn, Guy Amit, and Yisroel Mirsky. What was your prompt? a remote keylogging attack on AI assistants. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3367–3384, Philadelphia, PA, aug 2024. USENIX Association.

[47] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[48] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. FastServe: Iteration-Level preemptive scheduling for large language model inference. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pages 57–74, Renton, WA, may 2026. USENIX Association.

[49] Yi Xu, Ziming Mao, Xiangxi Mo, Shu Liu, and Ion Stoica. Pie: Pooling cpu memory for llm inference. *arXiv preprint arXiv:2411.09317*, 2024.

[50] Xiaobei Yan, Yiming Li, Hao Wang, Han Qiu, and Tianwei Zhang. Bithydra: Towards bit-flip inference cost attack against large language models. *arXiv preprint arXiv:2505.16670*, 2025.

[51] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[52] 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)*, pages 521–538, 2022.

[53] Tianchen Zhang, Gururaj Saileshwar, and David Lie. Time will tell: Timing side channels via output token count in large language models. *arXiv preprint arXiv:2412.15431*, 2024.

[54] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.

[55] Yuanhe Zhang, Xinyue Wang, Haoran Gao, Zhenhong Zhou, Fanyu Meng, Yuyao Zhang, and Sen Su. PD^3F: A pluggable and dynamic DoS-defense framework against resource consumption attacks targeting large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 3641–3671, Suzhou, China, nov 2025. Association for Computational Linguistics.

[56] Yuanhe Zhang, Zhenhong Zhou, Wei Zhang, Xinyue Wang, Xiaojun Jia, Yang Liu, and Sen Su. Crabs: Consuming resource via auto-generation for LLM-DoS attack under black-box settings. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11128–11150, Vienna, Austria, jul 2025. Association for Computational Linguistics.

[57] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

[58] 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)*, pages 193–210, 2024.

[59] Zhenhao Zhu, Yue Liu, Zhiwei Xu, Yingwei Ma, Hongcheng Gao, Nuo Chen, Yanpei Guo, Wenjie Qu, Huiying Xu, Zifeng Kang, Xinzhong Zhu, and Jiaheng Zhang. ExtendAttack: Attacking servers of LRMs via extending reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 35257–35265, 2026.

A Broader Impacts and Responsible Disclosure

Responsible Disclosure. This work strictly adheres to established ethical standards in AI & Cyber research. All experiments are conducted in controlled and isolated environments, and no attacks are launched against public LLM services to avoid potential disruption to real-world users. Our evaluation relies exclusively on publicly datasets without collecting or processing any private user data, and all adversarial inputs are reviewed to exclude harmful/offensive content. As the proposed method has been timely archived by `https://www.promptfoo.dev/`, code artifacts consist of benchmarking scripts and evaluation harnesses are available via the anonymous link. We would double check for ethical issues before making them public, but not as attack tools or prompt arsenals.

Societal Impact. This work exposes a previously understudied class of system-level attack surface in LLM serving frameworks. Understanding these attack surfaces enables service providers to design more robust schedulers, admission controls, and per-tenant isolation mechanisms. Defense mechanisms discussed in Appendix G provide an actionable guidance for practitioners. While this paper proposes a new system-level attack, our ultimate goal is defensive. We hope it serves as a wake-up call to service providers as current systems designers often treat latency degradation as a *performance issue* rather than a *security threat*. It is essential to understand that DoS attack in LLMs carries equivalent damage potential to conventional network DDoS attacks with asymmetric monetary cost from the attacker and service provider, especially the operational cost and carbon emissions from datacenters are becoming pressing issues. The second note is that this work provides a quantitative framework for service providers to "red team" their own infrastructure as a stress test, such that they could align attacker's cost vs. system degradation. This would allow providers to identify essential boundaries from a wide array of system parameters, and help providers design better admission control policies that remain robust in a competitive and fluid environment.

B Technical Appendices and Supplementary Material

B.1 Threat Model

The full threat model, including the five deployment assumptions (shared global KV-cache pool; FCFS admission; preemption enabled; no per-tenant KV isolation; visible streaming per-token timestamps), the attacker's capabilities and limits, and the explicit scope-and-boundary statement of our claims, is stated in the main text (Section 5.1). We do not repeat it here to avoid duplication; the discussion in Appendix G on countermeasures uses the same terminology and boundary conditions.

B.2 Theoretical Analysis

Our objective here is to identify *sufficient conditions* under which latency amplification can emerge in standard continuous-batching LLM serving with default settings, but not to prove or exhaust that all serving frameworks with various customized scheduling policies.

Proposition 1 (CB Logical Isolation). In normal cases with no memory pressure, system utilization $\rho < 1 - 1/B$ and batch size B, the expected TTFT overhead induced on r_n by r_{adv} satisfies, where $T_{\text{prefill}}(r_{\text{adv}})$ is the prefill latency of r_{adv}. Consequently, the TTFT penalty is independent of L_{adv} when the system is stable and memory-unconstrained.

Proof. Under CB, the scheduler assigns requests to iteration slots. Once r_{adv} enters the running queue, it occupies one out of B slots, leaving the remaining $B - 1$ slots available for normal requests. Thus, r_{adv} can delay normal requests only if it occupies the slot that would otherwise admit normal requests. Since CB continuously admits new requests whenever running requests complete, the additional waiting time of normal requests is bounded by the fraction of scheduling capacity consumed by r_{adv}, namely approximately $1/B$ of the prefill cost. If Δ_{slot} denotes the duration of one scheduling iteration, then the expected waiting time satisfies,

$$\mathbb{E}[W_n] \approx \mathbb{E}\left[\frac{\texttt{slots_before}(r_n)}{B}\,\Delta_{\text{slot}}\right], \tag{6}$$

which yields Eq. (1). For typical production settings, e.g., $B \geq 16$ and $T_{\text{prefill}} \approx 100$ ms, the overhead is below 6.25 ms per adversarial request. The key insight is that the decoding phase of r_{adv}, even if extremely long, does not by itself block normal requests, because CB continuously replenishes freed slots with waiting requests. □

Proposition 2 (ITL-KV Correlation). Let F_t denote the total active KV state referenced across all running requests at decode step t. In memory-bound regime, the per-step decode latency T_t is approximately affine in aggregate KV utilization, α captures the per-token KV memory access cost (both keys and values, across all layers and KV heads, T is data type), and β captures compute-bound overhead independent of context length.

Proof. During decoding, the attention kernel streams KV-cache blocks from HBM to on-chip SRAM. For a single request, the attention time scales linearly with context length:

$$T_{\text{attn}}(r_i) \approx \frac{N_{\text{ctx}}(r_i)\, d_{\text{model}}\, \texttt{sizeof(T)}}{\text{BW}_{\text{HBM}}}. \tag{7}$$

With N concurrent requests, their attention kernels contend for the same HBM bandwidth. Under the *Roofline Model* [47], the effective service time grows with aggregate memory demand. Therefore, the attention component can be rewritten as a function of the total active context length, yielding Eq. (2). Compute-bound operations, such as normalization and elementwise kernels, contribute the approximately constant term β, while the attention component grows with KV-cache utilization. □

In addition, CUDA profiling shows that attention decode kernels increase with KV-cache usage, whereas compute-bound kernels remain comparatively stable. The measured R^2 values further confirm a strong ITL–KV correlation under production-relevant concurrency levels.

Proposition 3 (FCFS-Induced HOL Blocking). Consider a typical FCFS admission queue $\mathcal{Q}_t = (q_1, q_2, \cdots, q_m)$ and $r(q_i)$ as the blocks required to admit request q_i and F_t as the current free block count. Assume $r(q_1) > F_t$, but there exists q_j, $j > 1$ with $r(q_j) \leq F_t$. If the scheduler does not perform skip-ahead admission, then no request q_j can be admitted until $F_t \geq r(q_1)$ or q_1 is removed. The minimum blocking duration for every later request is $W_j \geq \tau(q_1, t) - t$, where $\tau(q_1, t)$ is the first future time when free blocks reach $r(q_1)$. The attacker drives to meet this condition by accumulating, where $\texttt{KV_blocks}(r)$ is the number of physical blocks allocated to request r, `total_blocks` is the total GPU pool size and M_{req} is the blocks required for the HOL waiting request.

Proof. The scheduler iterates through the `WAITING` queue in FCFS order and invokes `can_allocate()` for each request. This function returns `False` when `free_block_queue.num_free_blocks` $< M_{\text{req}}$. Upon the first allocation failure, the scheduler terminates the admission loop rather than skipping smaller requests, thereby preserving FCFS fairness. Hence, once the HOL requests cannot be allocated, all subsequent waiting requests are also prevented from entering the running queue, which establishes the HOL-blocking condition. The stall persists until the running queue frees at least M_{req} KV blocks. Since running requests release KV blocks only upon completion, recomputation-based recovery, or explicit preemption, the minimum block-freeing rate is bounded by decode throughput, yielding Eq. (3). □

Corollary 1 (Preemption Cost). When the scheduler preempts a victim request r_v of current sequence length L_v, the recovery cost satisfies,

$$T_{\text{recover}}(r_v) = \begin{cases} T_{\text{prefill}}(L_v), & \text{recomputation recovery,} \\ S_{\text{KV}}(L_v) \;/\; BW_{\text{PCIe}}, & \text{swap recovery,} \end{cases} \tag{8}$$

where $T_{\text{prefill}} \approx \Theta(L_v^2)$ represents standard transformer where attention computation dominates, and the swapped KV state has size $S_{\text{KV}}(L_v) = 2L_v \cdot n_{\text{layers}} \cdot n_{\text{kv_heads}} \cdot d_{\text{head}} \cdot \text{sizeof}(T)$. The attacker's marginal cost to trigger preemption by adding δ tokens near the saturation boundary is only $O(\delta \cdot \text{price_per_token})$, yielding an asymmetry ratio $T_{\text{recover}}/C_{\text{trigger}} \in \Omega(L_v/\delta)$.

Proof. For recomputation-based recovery, the scheduler discards the victim's KV cache and replays the forward pass over the full accumulated sequence, including both the prompt and previously generated tokens. Transformer forward complexity over L_v tokens is $O(L_v d^2 + L_v^2 d)$, where the quadratic attention term dominates for long contexts, yielding $\Theta(L_v^2)$ cost. For swap-based recovery, the KV cache stores keys and values for each layer and each token; transferring this state over PCIe gives latency linear in L_v, namely $\Omega(L_v/\text{BW}_{\text{PCIe}})$. The attacker, however, only needs to push aggregate KV usage beyond the preemption threshold. Near the saturation boundary, this may require only one additional KV block, so the triggering cost scales with δ, whereas the victim's recovery cost scales with L_v. □

Note that in practice, the wall-clock recovery time can be reduced via FlashAttention, chunked prefill and prefix caching. The analysis above provides an upper bound of the recovery cost.

B.3 vLLM Scheduler Basics

B.3.1 KV-Cache Saturation

The physical limit of GPU VRAM is a hard constraint in LLM serving. In vLLM, memory management is orchestrated by the `BlockSpaceManager`[2], which maps logical KV blocks to physical GPU memory.

To manage concurrency, the scheduler maintains a `WAITING` queue for pending requests and a `RUNNING` queue for those currently executing on the GPU. During the autoregressive decoding phase, the memory footprint of every active request grows monotonically. For each inference iteration, the `Scheduler` assigns new physical blocks from the global `free_block_queue`. Unlike computational slots that are ephemeral and reset after every step, the KV-Cache blocks allocated to running sequences are persistent. As requests in the `RUNNING` queue progress through the decoding phase, they continuously consume the remaining capacity in the `free_block_queue`. An adversary can take advantage of this by exploiting long sequences to quickly deplete the memory resources and drive the system toward a saturation point.

When this physical limit is reached, the scheduler's admission logic creates a bottleneck. Following a strict First-Come-First-Served (FCFS) policy, the scheduler iterates through the `WAITING` queue and invokes `can_allocate()` for the request at the head of the line. If this check fails due to memory exhaustion, scheduling terminates immediately to preserve request ordering and fairness, which we term as *Memory-Based HOL Blocking*. This blockage creates a DoS situation even if the computational capacity is still available, i.e., global `num_batched_tokens` budget used by `chunked_prefill` is sufficient to accommodate subsequent, smaller requests. But due to physical block unavailability, legitimate short requests are stalled indefinitely.

B.3.2 Preemption and Recovery Penalties

The vulnerability emerges when the system attempts to recover from this saturation. When the cumulative memory demand of `RUNNING` requests surpasses the physical capacity, the scheduler must reclaim resources to prevent a deadlock. It triggers a deterministic preemption mechanism by invoking `_preempt()` to free up space. To minimize the impact on long-running tasks, vLLM typically enforces a Last-In-First-Out (LIFO) policy and selects the most recently scheduled request (r_{victim}) to evict. For fairness, the preempted victim is not discarded but prioritized by being prepended to the front of the `WAITING` queue. Depending on different implementations, vLLM operates under one of the following principles for recovery:

- ✦ **Swapping (I/O Bottleneck):** Common in legacy architectures (e.g., vLLMv0), memory swap offloads the victim's KV blocks from high-speed GPU VRAM to CPU host memory. Recovery requires migrating these blocks back over the bandwidth-constrained PCIe bus.
- ✦ **Recomputation (Compute Bottleneck):** In unified scheduler architectures (e.g., vLLMv1), the system prioritizes implementation simplicity by discarding the victim's KV-Cache entirely. To resume execution, the engine must re-process the prompt and all previously generated tokens from scratch. This shifts the workload from the linear-time decoding back to the quadratic-time prefilling.

While preemption is intended to ensure stability, it exposes an asymmetry facing adversaries. The attacker can trigger a preemption with a marginal increase in memory usage, yet the system must pay a disproportionately high cost to recover the victim request later. By toggling across the saturation boundary, the attacker can induce the system into back-and-forth preemption thrashing loop that continuously imposes pressure on the memory hierarchy.

B.4 Attack Prompt Construction

Here, we provide more detail and analysis of the attack cost and how we finally reach the compositional prompt sets.

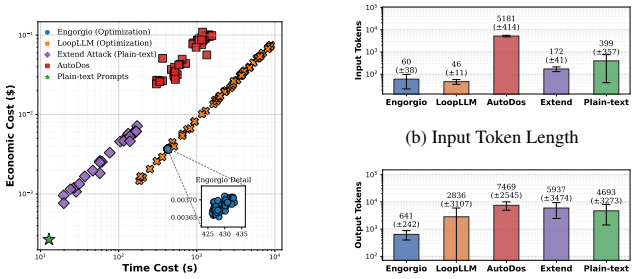
Attack Cost Analysis. While existing literature has proposed a wide range of adversarial techniques [8, 19, 59, 56], their economic viability remains underexplored. In production environments,

[2]Source code references are based on the official vLLM repository: `https://github.com/vllm-project/vllm`.

the usage is metered by pay-per-use billing so monetary expenditure plays an essential role for launching an attack. To evaluate different attack strategies, we formalize the cost structures into three paradigms: ❶ white-box attacks that craft adversarial prompts on a surrogate model and transfer prompts to the target model [8, 19]; ❷ black-box attacks that repetitively query an API to craft adversarial prompts [59, 56]; ❸ plain-text prompts that utilize deterministic logic and instruction-follow capability of LLM that are natural and non-adversarial [9]. Their monetary cost can be quantified as,

$$C = \begin{cases} T_{\text{opt}} \cdot P_{\text{avg}} \cdot p_e, & \text{(White-Box)} \\ \sum_{j=1}^{N} \left(H_{i,j} \cdot p_i + H_{o,j} \cdot p_o \right), & \text{(Black-Box)} \\ H_i \cdot p_i + H_o \cdot p_o. & \text{(Plain-text)} \end{cases} \tag{9}$$

Since *white-box* attacks only consume electricity energy on the attacker's side, it takes T_{opt} optimization time on a P_{avg} average power GPU with unit electricity price p_e. However, this computational investment often yields diminishing returns, as gradient-based optimization suffers from extensive time of convergence and poor cross-model transferability, making the adversarial prompts fragile under different configurations. *Black-box* attacks take N query iterations to accumulate over the sum of input and output tokens $H_{i,j}$ and $H_{o,j}$ for all j iteration, priced by the rates p_i and p_o accordingly. Finally, *plain-text prompts* incur minimal one-time cost for a single query and response.



(a) Economic vs. Time Cost

(b) Input Token Length

(c) Output Token Length

Figure 9: Attack cost and effectiveness. (a) Monetary cost (USD) vs. computational time (seconds). *Direct Generation* (green star) incurs minimal overhead, whereas optimization-based methods show log-linear cost scaling. (b) Distribution of output token lengths. Optimization methods (e.g., *AutoDoS*) produce significantly longer sequences than the concise outputs of direct heuristics.

We calibrate expenses using GPT-4o mini pricing and industrial electricity benchmarks [24, 40] as shown in Fig. 9a. The results demonstrate that optimization-based methods like AutoDoS and Engorgio exhibit a log-linear increase in costs, while plain-text prompts reduce orders of magnitude monetary and timing cost ($100 - 500\times$ less). Further, as shown in Figs. 9b and 9c, plain-text prompts produce output sequences comparable to optimization while using significantly shorter inputs. This minimal footprint makes them less likely to be detected via length-complexity filters compared to the verbose prompts such as AutoDoS [56]. Thus, we adopt plain-text prompts as one of the main methods in attack payload construction.

Plain-text prompts use structured natural-language instructions that exploit instruction-following behavior to force legitimate but long generations [25, 9]. For example, a prompt may ask the model to write a story where every sentence is immediately followed by a 50-word analysis of its grammar and syntax. Such prompts are semantically coherent, transferable across instruction-tuned models, and less exposed to perplexity-based filters [2, 26]. Second, the framework can incorporate stronger payloads such as ExtendAttack [59], which further improves disruption by triggering extended reasoning behavior. We represent each payload as a parameterized template $T(s)$, where s is a variable seed such as a topic or number.

Expansion Ratio Bound. For plain-text templates $T(s)$ on an instruction-tuned model, the expansion ratio satisfies $R = L_{\text{out}}/L_{\text{in}} \geq 10$ with probability greater than 0.85, as demonstrated empirically across 1000 template instantiations on Qwen3-8B, with mean $R = 11.8$ and standard deviation $\sigma = 3.2$.

Tiered Design. Prompts are grouped into three tiers according to their target KV-cache footprint:

- ❶ **High-Intensity** ($\mathcal{P}_{\text{High}}$): near-limit generations with $L_{\text{out}} \to L_{\max}$, typically 4000–8000 tokens, used during Fill to rapidly deplete `free_block_queue`.
- ❷ **Mid-Intensity** ($\mathcal{P}_{\text{Mid}}$): moderate generations with $L_{\text{out}} \in [1000, 2000]$ tokens, used to sustain memory pressure during transitional states.

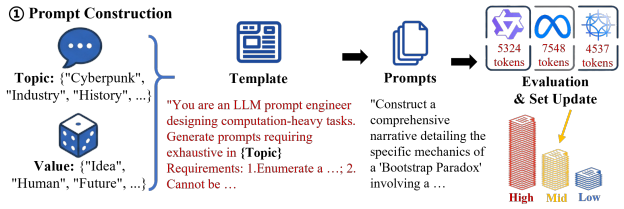


Figure 10: Hierarchical design of adversarial prompt construction by categorizing prompts into a tiered set ($\mathcal{P}_{High}, \mathcal{P}_{Mid}, \mathcal{P}_{Low}$).

❸ **Low-Intensity** ($\mathcal{P}_{\text{Low}}$): short but non-trivial generations with $L_{\text{out}} \approx 500$ tokens, used during Squeeze to tip an already saturated scheduler into preemption and also used for probing.

C Experiment Settings

C.1 Illustration of Performance Metrics

To characterize both system-level efficiency and user-perceived responsiveness, we mainly consider the following key performance metrics:

`TTFT (Time to First Token)` captures the initial latency perceived by the end-user. Formally, it is defined as the interval between request submission (T_0) and the generation of the first output token: $\text{TTFT} = T_{\text{schedule}} + T_{\text{prefill}}$, where T_{schedule} denotes the queuing delay incurred by the scheduler, and T_{prefill} corresponds to the compute-bound prefilling stage [48, 1]. Low TTFT indicates high system responsiveness and user retention.

`TPOT (Time Per Output Token)` measures the average latency between consecutive output tokens during the decoding phase. TPOT is an indicator of sustained generation throughput and primarily constrained by memory bandwidth.

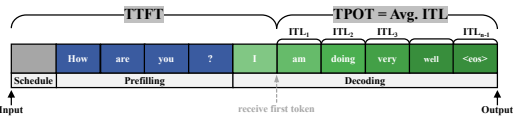


Figure 11: Illustration of performance metrics (TTFT, TPOT, ITL).

`ITL (Inter-Token Latency)` records the fine-grained time intervals between the generation of consecutive tokens. Unlike TPOT which provides an aggregated average, the ITL sequence captures instantaneous fluctuations caused by system contention or interference at each generation step. In this work, we leverage the ITL sequence as an important side-channel signal for inferring the real-time memory usage.

`End-to-End Latency` represents the total wall-clock time elapsed from request submission to the completion of the full response, which captures the cumulative time of queuing, prefilling and decoding.

C.2 Configuration

Hardware Specifications. Our experiments are conducted on a high-performance multi-GPU server equipped with 8 × NVIDIA L40S GPUs (48 GB VRAM each) interconnected via PCIe Gen4. The host system is powered by dual AMD EPYC 7763 64-Core processors (128 physical cores total) operating at a boost frequency of 3.53 GHz, supported by 1 TB of system RAM and 512 MiB of L3 cache to facilitate large-scale model operations.

We select a diverse suite of mainstream open-source model families, including Qwen3-8B, Gemma3-12B-it and DeepSeek-R1-Distill-Llama-8B. Our selection criteria are primarily driven by community adoption; these models consistently rank among the highest download counts on Hugging Face in their series, thereby representing the de facto standard for real-world deployments.

Software Stack. The experimental environment runs on Linux, utilizing NVIDIA Driver 580.95.05 and CUDA 13.0 for hardware acceleration. The serving infrastructure is built upon vLLM, leveraging PyTorch as the underlying tensor computation backend. Attack orchestration and side-channel analysis pipelines are implemented in `Python 3.11`, employing LightGBM [17] for lightweight

Input: Saturation threshold C_{sat}; margins $\delta_{\text{large}}, \delta_{\text{small}}$; prompt arsenal $\mathcal{P} = \{\mathcal{P}_{\text{High}}, \mathcal{P}_{\text{Mid}}, \mathcal{P}_{\text{Low}}\}$; estimator $\mathcal{M}_\theta$
Output: Adaptive injection decisions for Fill, Squeeze, and Back-off
$t \leftarrow 0$;
while *Attack Active* **do**
  Dispatch $p_{\text{probe}} \in \mathcal{P}_{\text{Low}}$ and record ITL sequence $\mathbf{t}_t$;
  $\widehat{U}_{\text{sys}} \leftarrow \mathcal{M}_\theta(\mathbf{t}_t)$;
  $\Delta_{\text{mem}} \leftarrow C_{\text{sat}} - \widehat{U}_{\text{sys}}$;
  if $\Delta_{\text{mem}} > \delta_{\text{large}}$ **then**
    Dispatch $p \in \mathcal{P}_{\text{High}}$; `// Fill: rapidly deplete free KV blocks`
  else if $0 < \Delta_{\text{mem}} \leq \delta_{\text{small}}$ **then**
    Dispatch $p \in \mathcal{P}_{\text{Low}}$; `// Squeeze: cross saturation with minimal cost`
  else
    Sleep for t_{wait}; `// Back-off: avoid self-preemption`
  end
  $t \leftarrow t + 1$;
end

Algorithm 1: Fill and Squeeze Attack

regression modeling and Scikit-learn for statistical evaluation. To ensure the reproducibility of our results, we detail the specific versions of key software dependencies in Table 4.

Table 4: Key Software Dependencies

Package	Version	Package	Version
Python	3.11.0	Transformers	4.57.6
PyTorch	2.9.0	vLLM	0.11.2
TorchVision	0.24.0	xFormers	0.0.33.post1
Torchaudio	2.9.0	LightGBM	4.6.0
CUDA	13.0	Scikit-learn	1.7.2
Accelerate	1.12.0	NumPy	1.26.4
SciPy	1.17.0	Pandas	2.3.3

D More Details of Fill and Squeeze Attacks

The operational logic of the *Fill and Squeeze* attack is formalized in Algorithm 1. In principle, it establishes a closed-loop feedback control system that synchronizes adversarial injection with the victim's estimated memory state. For completeness, we illustrate the major steps in the following.

To bridge the visibility gap inherent in black-box serving, the execution cycle begins with a monitoring phase (Lines 3–6) where low-intensity requests ($p_{probe} \in \mathcal{P}_{Low}$) exploit the ITL side-channel to infer the global KV-cache utilization ($\hat{U}_{sys}$) via the estimator $\mathcal{M}_\theta$. This estimation allows the calculation of a critical *memory gap* ($\Delta_{mem} = C_{sat} - \hat{U}_{sys}$), representing the remaining capacity before the scheduler forces victim eviction. Driven by this metric, the algorithm dynamically toggles its injection strategy to manipulate the scheduler's state machine: when the system is underutilized ($\Delta_{mem} > \delta_{large}$), the attacker enters the *Fill* regime, dispatching high-complexity prompts ($p \in \mathcal{P}_{High}$) to rapidly deplete the `free_block_queue` and accelerate the system toward saturation; conversely, as the memory state approaches the fragile preemption boundary ($0 < \Delta_{mem} \leq \delta_{small}$), the strategy shifts to the *Squeeze* regime, where minimal token injections ($p \in \mathcal{P}_{Low}$) are sufficient to tip the aggregate load over the physical threshold C_{sat}, triggering the deterministic and computationally expensive preemption mechanism. Finally, to ensure attack sustainability, a *Back-off* mechanism (Lines 14–16) creates a temporal window (t_{wait}) when the system is overloaded, preventing the adversary's own requests from becoming eviction victims while trapping benign users in a resource-draining livelock of continuous allocation and recomputation.

D.1 Component Ablation

To isolate the contribution of each F&S component, we compare the full attack against four ablated variants on Qwen3-8B under the same five-run aggregation protocol used for the main results

(Section 5.3): (i) **Fill-only**, disabling the tier switch and injecting only $\mathcal{P}_{\text{High}}$ prompts open-loop; (ii) **Squeeze-only**, disabling the Fill ramp and issuing only $\mathcal{P}_{\text{Low}}$ probes from an idle state; (iii) **no adaptive back-off**, keeping the tier logic but never sleeping when $\Delta_{\text{mem}} \leq 0$; and (iv) **non-tiered adaptive injection**, keeping closed-loop dispatch but using a single fixed prompt template regardless of $\hat{U}_{\text{sys}}$.

Table 5: Component ablation of F&S on Qwen3-8B, five-run averages. Setting definitions in text.

Setting	TTFT (s)	TTFT P99 (s)	TPOT (ms)	TPOT P99 (ms)	Preempt#	Req#	Cost ($)
Full F&S	**12.124**	**403.282**	**205.90**	**2892.10**	268	**205**	**0.74**
Fill-only	3.572	92.553	111.39	1122.38	854	508	0.81
Squeeze-only	0.095	0.139	48.94	58.93	0	3462	1.19
No adaptive back-off	0.803	24.608	78.75	322.67	583	448	0.94
Non-tiered adaptive	0.272	0.921	73.39	246.61	384	510	0.99

Findings. Full F&S produces the largest mean and tail degradation with the fewest requests and the lowest cost. *Fill-only* creates high aggregate pressure and many preemptions (854, the highest of any row) but cannot repeatedly control the recovery boundary, so mean TTFT only reaches 3.57 s (vs. 12.12 s for full F&S). *Squeeze-only* cannot first establish the near-saturation state, so no preemption ever fires and the "attack" collapses to benign-level latency at $3\times$ the cost. *Removing back-off* lets self-preemption evict the attacker's own long requests, degrading both TTFT (0.80 s) and cost ($\$0.94$). *Non-tiered adaptive* keeps closed-loop feedback but loses the ability to switch between capacity-consuming and boundary-tipping prompts, so it neither reaches nor maintains saturation.

Why we do not report a stand-alone "no ITL estimator" row. The estimator $\mathcal{M}_\theta$ drives *both* adaptive back-off and tier selection: removing it disables both decisions simultaneously and is therefore not an independent one-factor ablation. The Fill-only row and the published fixed-schedule baselines (Engorgio, LoopLLM, ExtendAttack in Table 1) already play the role of the corresponding open-loop comparisons.

D.2 Effects of Chunked Prefill

Previously, we state that Continuous Batching (CB) acts as the primary logical isolation mechanism against latency attacks on the system level. In addition to CB, modern frameworks employ a complementary optimization known as chunked prefill to mitigate interference [1].

In traditional iteration-level scheduling, the prefill phase of a new request is atomic. If an adversary submits a prompt with extremely long length (e.g., the prompts of Autodos exhibit this behavior [56]), the GPU remains occupied by the prefill kernel for a significant duration, stalling the decoding iterations of all co-located users. This creates a compute-bound HOL blocking, which is different from the memory-based HOL blocking during the decoding phase mentioned earlier. Chunked Prefill alleviates this by decomposing the prefill operation into smaller chunks. Instead of processing the entire prompt in one pass, the scheduler processes a fixed budget of tokens (e.g., 512 tokens) per iteration, interleaving these prefill chunks with the decoding steps of other active requests.

We posit that chunked prefill plays a *secondary factor* in mitigating latency attacks because it addresses compute latency, ensuring that Inter-Token Latency (ITL) remains stable during the arrival of large requests. However, it does not mitigate the memory consumption of those requests. As the chunks are processed, the global KV-cache usage still rises monotonically. Once the memory is exhausted, the scheduler effectively reverts to the blocking behaviors exploited by our Fill vector, regardless of how the prefill was computed. The Squeeze stage targets the preemption logic during the decoding phase. Since Chunked Prefill is active primarily during the initialization of a request, it barely offers protection against the decoding phase induced by an adversary toggling the system across the saturation boundary.

D.3 More Discussions on Model-Specific Characteristics

Our experiments reveal that the efficacy of a latency attack payload can be highly model-dependent. An interesting case is the strong affinity of ExtendAttack for the DeepSeek-R1 model, as shown in Tables 1 and 3. We attribute this to DeepSeek-R1's inherent bias towards meticulous, step-by-step reasoning and explicit self-explanation when faced with complex or unusual inputs. The core mechanism of ExtendAttack is to force a model into a series of computationally intensive, yet

semantically trivial, decoding sub-tasks [59]. The poly-base ASCII obfuscation acts as a powerful trigger for this behavior in DeepSeek-R1. The model dedicates a substantial volume of tokens to explicitly document its reasoning process by performing base conversions, mapping values to characters, and verifying each step, before addressing the primary query. Overall, this behavior effectively turns the model's own reasoning alignment into a Denial-of-Service amplifier. This pronounced effect is observed to be unique to the DeepSeek-R1 model in our tests, suggesting a specific architectural or fine-tuning artifact. This observation reinforces a key aspect of our work: F&S strategy is payload-agnostic. While some payloads like ExtendAttack may have a natural advantage on certain models, F&S provides a universal, system-level amplification mechanism.

On the cost side, on Qwen3-8B, the F&S+ExtendAttack variant not only causes the most damage but also accomplishes with the lowest monetary cost ($0.42). This slightly counter-intuitive result occurs because the high efficiency of the ExtendAttack payload in generating long sequences allows the F&S to reach the "Fill" and "Squeeze" states with fewer requests, thus optimizing the attacker's budget. Thus, the optimal pairing of a payload with a target model family is an interesting avenue for future research.

D.4 Generalization to SGLang

To validate the generalization of F&S to SGLang [57], we first perform an analysis on the source code (commit `99c0b62`, files `schedule_policy.py` and `scheduler.py`).

Fill: token-budget admission creates the same HOL blocking. SGLang's admission gate lives in `PrefillAdder.add_one_req()` (`schedule_policy.py`). Before admitting a waiting request, the scheduler computes the remaining token budget:

$$\texttt{rem_total_tokens} = \underbrace{\texttt{available_size()} + \texttt{evictable_size()}}_{\text{free + evictable RadixCache tokens}} - \underbrace{\texttt{rem_total_token_offset}}_{\text{reserved for running reqs}}, \quad (10)$$

where `rem_total_token_offset` accumulates, for every already-admitted request, the estimate `extend_input_len` + `max_new_tokens` (capped at `CLIP_MAX_NEW_TOKENS` $= 4096$) + `page_size`. A new request is rejected with `AddReqResult.NO_TOKEN` as soon as `total_tokens` $\geq$ `rem_total_tokens`. Once this condition fires, `budget_state()` halts the entire `get_new_batch_prefill` loop; no subsequent waiting request is inspected regardless of its size. This is structurally identical to the HOL-blocking mechanism in vLLM's `can_allocate()` loop (Proposition 3).

The F&S attack exploits this budget in two compounding ways. First, long-running adversarial requests grow `rem_total_token_offset` rapidly, shrinking the effective budget even before physical memory is exhausted. Second, the conservative `CLIP_MAX_NEW_TOKENS` cap means the system reserves 4096 future tokens *per admitted request*, making saturation of the budget arithmetically easier than in vLLM's block-counting model. Concretely, with a token pool of K tokens and n running requests each clipped at 4096, the budget becomes negative once $n \cdot 4096 \geq K$, even if physical memory still has free blocks.

The LPM policy does not neutralize the Fill phase. SGLang's default scheduling policy is Longest Prefix Match (`CacheAwarePolicy.LPM`), which reorders the waiting queue to maximize RadixCache hits. However, `_determine_active_policy()` automatically reverts to plain FCFS when the queue length exceeds 128 requests—precisely the high-load condition induced by the Fill phase. Even under LPM, once `add_one_req()` returns `NO_TOKEN` for the head-of-line request, the outer admission loop terminates and no further requests are considered. The cache-aware ordering only affects *which* request sits at the head of the queue as it cannot bypass the hard budget gate.

Squeeze: `retract_decode` mirrors vLLM's preemption asymmetry. When memory pressure prevents new admissions, SGLang recovers headroom via `retract_decode` (`scheduler.py`), which moves a subset of actively decoding requests back to the waiting queue. The retracted requests lose all decode progress and must be re-prefilled from scratch (recompute mode only—SGLang has no CPU-swap fallback at this stage), incurring an $\Theta(L_v^2)$ recovery cost identical to Corollary 1. The attacker exploits the same asymmetry: toggling the token budget across the saturation boundary at marginal cost $O(\delta \cdot \text{price_per_token})$ forces repeated `retract_decode` cycles, each of which makes a legitimate long-running request pay the full quadratic recomputation penalty.

Mapping between vLLM and SGlang. Table 6 maps each vLLM mechanism exploited by F&S to its SGLang counterpart. The Fill phase is enabled by the same HOL-blocking admission loop and the Squeeze phase is enabled by identical recomputation-based recovery asymmetry. The main architectural differences are that (i) SGLang's forward-looking token-budget makes the Fill phase easier to trigger than vLLM's block-count model, and (ii) the RadixCache evictable pool provides an additional depletion target absent in vLLM.

Table 6: Mapping of F&S mechanisms between vLLM and SGLang. Source: vLLM v0.11.2 `BlockSpaceManager`; SGLang commit `99c0b62` `schedule_policy.py` / `scheduler.py`.

F&S mechanism	**vLLM (v0.11.2)**	**SGLang (commit `99c0b62`)**
Admission gate	`can_allocate()`	`add_one_req()` $\rightarrow$ `budget_state()`
HOL-blocking trigger	Free blocks $< M_{\text{req}}$	`rem_total_tokens` ≤ 0
Budget type	Block count (reactive)	Token count (forward-looking)
Budget reservation	None	$+$`CLIP_MAX_NEW_TOKENS` $= 4096$ per req
Default queue order	FCFS	LPM (reverts to FCFS at queue > 128)
Long-output attack amplification	KV growth over time	KV growth *plus* immediate budget inflation
Eviction / recovery	`_preempt()` LIFO	`retract_decode`
Recovery mode	Recompute or CPU swap	Recompute only
Recovery cost	$\Theta(L_v^2)$ or $\Omega(L_v / BW_{\text{PCIe}})$	$\Theta(L_v^2)$
Additional depletion target	—	RadixCache `evictable_size()`

D.5 Experimental Validation on SGLang

To supplement the experiment validation across different serving frameworks, we validate the effectiveness of F&S attack on SGLang across three representative target models in Table 7 and visualize the corresponding scheduler-level dynamics in Fig. 12. Overall, F&S remains highly efficient under SGLang's serving architecture, inducing severe user-facing latency degradation with low attack volume and low financial cost.

Why SGLang Could Be More Vulnerable to Latency Attacks. Corresponding to our prior code-level analysis, from Table 7 baselines, we observe that SGLang is more vulnerable to the latency attacks baselines. On Gemma3-12B-it, LoopLLM achieves 267.30 s vs. F&S's 212.44 s; on DeepSeek-R1, ExtendAttack achieves 57.53 s vs. F&S's 55.07 s. We posit that it is due to SGLang's forward-looking budget model: any attack that submits requests with large *expected* output lengths can rapidly saturate `rem_total_tokens` because each request immediately consumes up to `CLIP_MAX_NEW_TOKENS` $= 4096$ of the reservation budget. LoopLLM and ExtendAttack, which target very long outputs, benefit disproportionately from this mechanism on SGLang compared to vLLM. In particular, ExtendAttack's prefill-phase amplification (forcing extended chain-of-thought reasoning) inflates the per-request budget reservation as well as the physical KV footprint. On DeepSeek-R1, whose reasoning alignment generates especially long outputs under ExtendAttack's trigger, the baseline can match F&S's raw TTFT.

Cost Benefits of F&S. Nevertheless, these do not diminish the contribution of F&S as the major distinction is on the *cost and request efficiency*. Even when baselines achieve comparable peak TTFT, they do so with substantially more attack traffic and higher cost. As shown in Table 7, on DeepSeek-R1, ExtendAttack requires 180 requests at \$3.08, while F&S+plain-text achieves a comparable 55.07 s TTFT with 181 requests at \$2.96. On Gemma3-12B-it, LoopLLM requires 262 requests at \$0.99 to reach its peak 267 s; F&S achieves 212 s at \$0.86. We can see that while brute-force methods can sometimes match or exceed F&S in raw TTFT by aggressively saturating SGLang's token budget, they generally require more attack traffic or higher cost. In contrast, the Squeeze phase forces repeated `retract_decode` cycles, as reflected by the oscillatory queue behavior and repeated saturation–recovery transitions. Once retracted, actively decoding victim requests lose decode progress and must be re-prefilled from scratch; because SGLang uses recomputation rather than CPU-swap fallback in this path, each recovery imposes a $\Theta(L_v^2)$ penalty on long-running victim requests. The attacker, however, only pays a marginal token cost to keep the system near the admission boundary. Thus, F&S generalizes beyond vLLM that despite implementation differences, modern

LLM serving frameworks that combine HOL-blocking admission control with recomputation-based recovery expose a common low-cost latency amplification channel.

SGLang-Specific Findings. Fig. 12 provides the visualization of F&S on SGLang. The event timeline, queue statistics, and KV-cache trace show that F&S pushes SGLang into a sustained high-pressure regime: KV usage remains close to saturation and repeatedly exceeds the 97.5% threshold, while the running and waiting queues exhibit sharp oscillations. Brute-force attacks that heavily flood the queue can sometimes saturate SGLang's budget even without adaptive feedback, because the conservative `CLIP_MAX_NEW_TOKENS` reservation lowers the Fill threshold compared to vLLM. This means SGLang's serving architecture is, in some respects, more vulnerable to ordinary flooding attacks than vLLM —a finding that highlights the security cost of forward-looking budget reservation. F&S amplifies this by also exploiting the Squeeze phase: repeated `retract_decode` cycles force victim requests through full $\Theta(L_v^2)$ recomputation at minimal attacker cost, maintaining high latency pressure even when the attacker reduces injection volume.

Table 7: Comparison of attack effectiveness on SGLang under attack settings. ↑ and ↓ denote the higher/lower the better, respectively. TTFT is measured in seconds (**s**), and TPOT is measured in milliseconds (**ms**).

Target Model	Attack Method	TTFT (↑)	TPOT (↑)	Attack Request# (↓)	Cost ($) (↓)
Qwen3-8B	Engorgio	0.09	48.40	4487	3.88
	LoopLLM	12.65	55.40	607	3.01
	ExtendAttack	6.74	57.55	422	3.31
	F&S+plain-text	42.40	56.44	202	2.71
Gemma3-12B-it	Engorgio	201.03	65.88	317	0.85
	LoopLLM	267.30	70.26	262	0.99
	ExtendAttack	174.79	62.26	362	1.02
	F&S+plain-text	212.44	58.97	340	0.86
DeepSeek-R1-Distill-Llama-8B	Engorgio	0.10	46.85	2041	4.11
	LoopLLM	1.97	50.42	1091	3.87
	ExtendAttack	57.53	58.76	180	3.08
	F&S+plain-text	55.07	60.18	181	2.96

E More Analysis of Side-Channel Probing

During inference-time attacks, the usage of KV-cache often serves as a critical latent variable that influences both model behavior and system-level scheduling decisions. However, in black-box or restricted-observability attack settings, an adversary typically has no direct access to the real-time KV-cache usage.

This raises a fundamental question: *Can KV-cache usage be inferred indirectly through observable system-side signals?*

Inspired by prior work on system-level side-channel analysis[35, 46, 53], we explore whether ITL can be leveraged to probe the usage state of KV-cache. In particular, we study the correlation between ITL and KV-cache usage and investigate the feasibility of constructing a KV-cache usage prober based solely on time-series observations of ITL.

E.1 Correlation under Different System Loads

In practical deployment environments, system load is inherently dynamic. Therefore, a viable side-channel probe should exhibit stable behavior across a wide range of load conditions. To investigate this, we evaluate the relationship between KV-cache usage and ITL under multiple concurrency levels ($C = 1, 2, 4, 8, 16$).

Experimental Setup. We conduct experiments by issuing concurrent requests to the serving system and recording both per-token ITL and real-time KV-cache utilization. Each concurrency level is tested over multiple runs to ensure statistical stability. The resulting data comprises over 900K aligned (ITL, KV-Usage) pairs across all configurations.

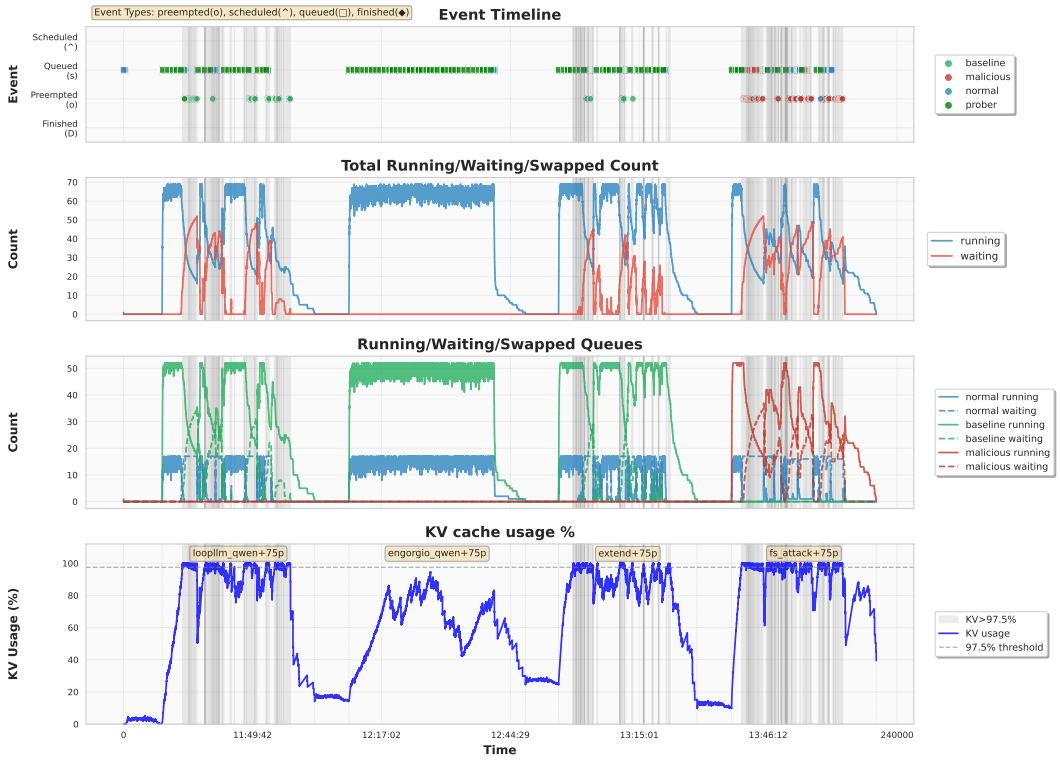


Figure 12: Scheduler dynamics of SGLang under different attacks. From top to bottom, the panels show request events, queue sizes, per-traffic queue composition, and KV-cache usage over time. Gray regions indicate KV usage above the 97.5% threshold. F&S induces sustained KV pressure, oscillating queues, and repeated saturation–recovery behavior, illustrating Fill-induced HOL blocking and Squeeze-induced recovery amplification.

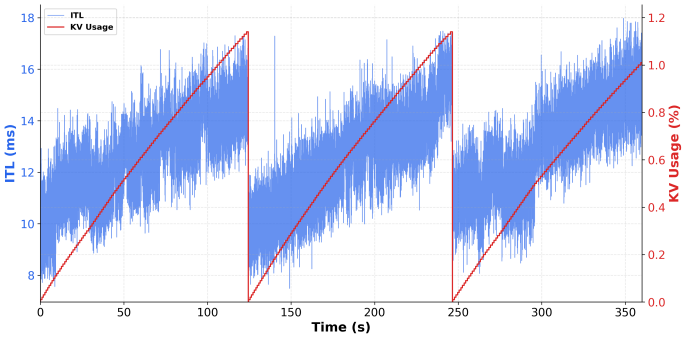
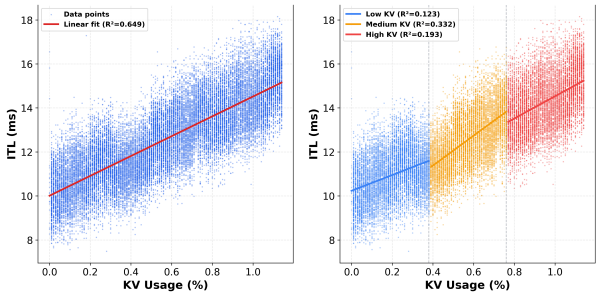


Figure 13: Temporal dynamics of ITL and KV-cache usage under single-user load (Concurrency=1). The blue trace represents per-token ITL, while the red trace indicates KV-cache utilization. Both metrics exhibit synchronized periodic patterns, suggesting a strong temporal correlation.



(a) Linear regression fit ($R^2 = 0.649$) (b) Region-wise analysis (Low/Medium/High)

Figure 14: KV-ITL correlation analysis. (a) Scatter plot with linear regression showing positive correlation between KV-cache usage and ITL. (b) Region-wise breakdown reveals that global correlation is driven primarily by cross-region variations rather than intra-region dynamics.

Quantitative Results. Table 8 summarizes the linear correlation strength (R^2) between KV-cache usage and ITL for each concurrency level. We observe a clear positive correlation across all configurations, with the correlation strength increasing significantly as concurrency grows.

Table 8: Linear correlation (R^2) between KV-cache usage and ITL under different concurrency levels.

Concurrency	R^2	**ITL (ms)**	**KV (%)**
C = 1	0.649	7 – 18	0 – 1.14
C = 2	0.848	7 – 22	0 – 2.29
C = 4	0.958	5 – 45	0 – 4.57
C = 8	0.978	8 – 80	0 – 9.13
C = 16	0.980	8 – 65	0 – 9.13

Fig. 15 visualizes the KV-ITL scatter plots for concurrency levels $C = 2, 4, 8, 16$ with linear regression fits. At low concurrency (C = 1), the correlation is moderate ($R^2 = 0.649$) due to limited variation in KV-cache usage. As concurrency increases, both the range of KV-cache utilization and the correlation strength grow substantially. For concurrency levels ≥ 4, the R^2 exceeds 0.95, indicating a highly predictable linear relationship. The 0.9 threshold is exceeded for all concurrency levels ≥ 4, demonstrating that the KV-ITL relationship is sufficiently stable for practical side-channel probing in realistic multi-user scenarios.

Analysis. The observed trend can be explained by two factors: (1) Higher concurrency leads to greater variation in KV-cache usage, providing a wider dynamic range for correlation analysis; (2) Under higher load, the system's memory pressure becomes more pronounced, causing ITL to respond more sensitively to KV-cache changes.

When KV-cache usage approaches saturation (e.g., near 100% utilization), system-level preemption, eviction, and scheduling mechanisms are triggered more frequently. As a result, ITL may exhibit increased non-linear behavior in extreme cases. However, in typical operating regimes (KV usage below 90%), the linear approximation remains highly accurate.

This observation motivates the design of a temporal KV-cache usage prober that tracks ITL dynamics over time rather than relying on instantaneous measurements, and suggests that the probe is most effective under moderate-to-high system load conditions. In addition, these trends are consistently observed on both Qwen3-0.6B and Qwen3-8B, suggesting that the phenomenon is not specific to a particular model scale.

E.2 Comparison of Time-Series Modeling Methods

Given the dynamic evolution of ITL during inference, we formulate the estimation of KV-cache usage as a time-series classification task. To determine the optimal probing engine, we conduct a comparative analysis between Gradient-Boosting Decision Trees (GBDT), specifically `LightGBM`, and representative deep learning architectures (LSTM, GRU, and Transformers) implemented via the `tsai` framework.

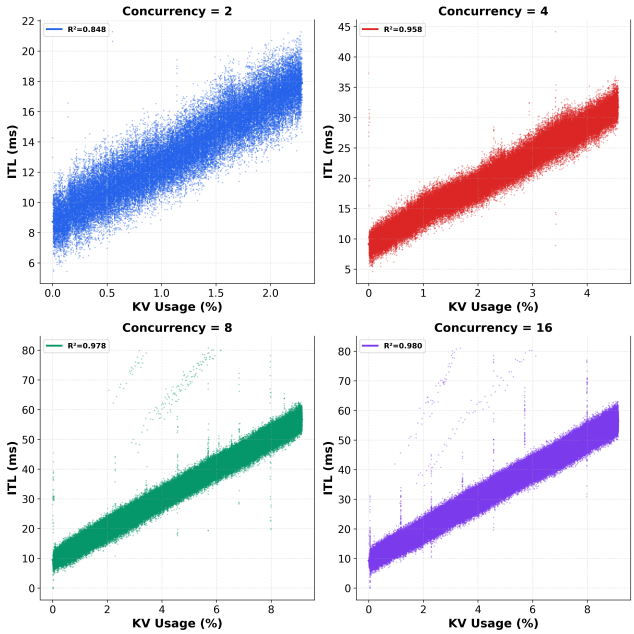


Figure 15: KV-ITL correlation under different system load configurations (Concurrency $= 2, 4, 8, 16$). Each subplot shows the scatter plot with linear regression fit. The correlation strengthens significantly as concurrency increases, with R^2 exceeding 0.95 for $C \geq 4$.

Table 9: Qwen3-8B classification performance comparison on 10-class dataset

Model	Acc.	F1	Train (s)	Infer (s)
LightGBM	0.87	0.88	6.94	0.01
tsai-LSTM	0.80	0.79	83.25	0.55
tsai-GRU	0.80	0.79	83.25	0.55
tsai-Transformer	0.75	0.75	158.91	0.56

As summarized in Table 9, our empirical results on the Qwen3-8B dataset reveal that `LightGBM` consistently outperforms its neural counterparts, achieving the highest accuracy (0.87) and F1-macro score (0.88). This suggests that GBDT-based methods effectively capture the decision boundaries within the high-dimensional features of side-channel traces. More critically, `LightGBM` demonstrates an overwhelming advantage in inference efficiency, maintaining a near-zero computational footprint (< 0.01s) compared to the significant overhead of neural models (e.g., ≈ 0.56s for `tsai-Transformer`). Since real-time side-channel attacks demand probing latencies negligible enough to avoid synchronization errors with the target LLM's token generation, we adopt `LightGBM` as the core engine for our KV-cache usage estimator.

E.3 Estimator Training

The attacker trains $\mathcal{M}_\theta$ on a surrogate system of the same model and hardware type by varying concurrent load to sweep through different U_{sys} levels. In a black-box API setting, the attacker can construct such a surrogate by renting the same GPU type and serving the same model class, then sweeping load from 1 to B concurrent requests. Thus, the ground-truth KV usage is obtained from the surrogate's internal metrics. Our previous results indicate high estimation accuracy.

E.4 Production Multi-Replica Prober Evaluation

To characterize how the ITL-based prober behaves outside the controlled L40S bench, we conducted, in coordination with an industry partner, a deliberately small-scale prober evaluation in a production scale-out topology. We limited the experiment to two replicas and *passive observation only*: no active F&S attack was launched against production traffic, and no cross-node model sharding was

probed. This scope reflects our responsible-evaluation and ethical-safety requirements; a larger active experiment could disrupt production services.

Setup. A production deployment of a Qwen3-235B-A22B–based model, vLLM 0.11.2, FCFS scheduling, streaming timestamps, and 8×H20 with TP8 per replica. The reference uses one replica behind the production gateway in sticky mode; the multi-node setting uses two independent replicas on separate nodes behind the same gateway in round-robin mode. The single-replica production collection contains 2.12M aligned (ITL, KV-usage) rows; the two-replica collection contains 336,467 aligned windows across three load stages.

Single-replica baseline (controlled vs. production). Table 10 compares the prober trained and evaluated on the controlled L40S bench against the same prober trained and evaluated on the single-replica production H20 deployment. Accuracy and macro-F1 drop from 0.870/0.880 to 0.771/0.765, i.e., the prober retains roughly 88% of the controlled-bench accuracy despite the change in GPU (L40S→H20), model class (Qwen3-8B→Qwen3-235B-A22B), and end-to-end network path.

Table 10: Single-replica prober: controlled L40S bench vs. single-replica production H20 deployment. Each setting trains and evaluates on its own aligned (ITL, KV-usage) data.

Setting	Accuracy	Macro-F1
Controlled L40S, Qwen3-8B	0.870	0.880
Production H20, single replica	0.771	0.765

Two-replica round-robin collection. For the multi-node experiment, we passively collect windows under three load stages driven by ambient production traffic. Table 11 summarizes request counts and observed KV-usage ranges across the stages. The 336,467 windows in this collection serve as the held-out test set for the zero-shot transfer in Table 12.

Table 11: Production round-robin collection: request counts, observed KV usage, and aligned window counts per load stage.

Stage	Requests (completed / dispatched)	Observed KV usage	Windows
Low	320 / 320	0.19–1.35%	4,031
Mid	637 / 640	2.43–34.73%	153,328
High	950 / 960	2.96–62.58%	179,108
Total	1,907 / 1,920	0.19–62.58%	336,467

Zero-shot transfer to two-replica traffic. We now transfer the single-replica prober *as-is*—without any additional calibration—to the two-replica round-robin test set. As shown in Table 12, accuracy and macro-F1 degrade to 0.681/0.646, retaining approximately 84% of the single-replica macro-F1 and 73% of the controlled-bench macro-F1. This reflects graceful degradation rather than signal collapse under production load balancing.

Table 12: Zero-shot transfer from the single-replica prober to the two-replica round-robin test set of Table 11. No target-specific recalibration.

Test set	Windows	Accuracy	Macro-F1
Single-replica in-domain reference	15,355	0.771	0.765
Two-replica round-robin	336,467	0.681	0.646

Client-observed stalls as an independent signal. Beyond the window-level classification metrics, we also directly inspect per-token ITL streams observed at the client. Table 13 counts, at each load stage, how many streaming chunks exceed 200 ms, 500 ms, and 1 s. At low load, no chunk exceeds 200 ms; at high load, 828 chunks exceed 200 ms, 222 exceed 500 ms, and 56 exceed 1 s (maximum observed ITL 1,332.9 ms). This load-dependent tail is independent of any model prediction and confirms that the ITL side channel remains observable at the client end even under two-replica routing.

Summary. Together, Tables 10–13 establish that ITL-based KV observation degrades gracefully but remains feasible across production replicas on multiple nodes, without any target-specific recalibration. As stated in Section 5.1, "black-box" refers to no victim-side access; it does not mean that a single frozen predictor transfers zero-shot to every deployment, and matching-surrogate calibration remains the recommended workflow.

Table 13: Client-observed stalls by load stage. Threshold columns count streaming chunks whose ITL exceeds the header value (in milliseconds).

Stage	Chunks scanned	Max ITL	>200 ms	>500 ms	>1000 ms
Low	120,119	65.5 ms	0	0	0
Mid	3,505,123	410.9 ms	2,954	0	0
High	3,955,238	1,332.9 ms	828	222	56

E.5 Additional Observations and Practical Considerations

During our experiments, we observe that the coverage of KV-cache usage states in the training data plays a critical role in KV estimation performance. If high-usage or saturated KV-cache states are underrepresented during training, the resulting model often struggles to correctly identify such regimes at test time. Therefore, to achieve reliable KV-cache usage inference, the training data collected offline should span the full range of KV-cache utilization levels, ensuring sufficient representation of extreme operating conditions.

F System Profile

To empirically validate the memory bandwidth contention in Finding #2 and subsequent propositions, we perform a granular micro-architectural characterization of the GPU execution environment using the NVIDIA Nsight Systems profiling suite. First, we instrument the vLLM inference loop with NVTX markers to delineate request boundaries and capture high-fidelity CUDA kernel traces using the command: `nsys profile -output profile -trace=cuda,nvtx exp.py`. Subsequently, to facilitate scalable and structured analysis, the raw traces are transcoded into a relational SQLite schema via `nsys export -t sqlite -output profile.sqlite profile.nsys-rep`. This approach allows for the systematic extraction of kernel-level performance metrics from the `CUPTI_ACTIVITY_KIND_KERNEL` table.

Our profiling results reveal a clear divergence in scaling behavior; while compute-bound operators (e.g., `rms_norm`, element-wise kernels) maintain near-constant execution durations across varying loads, the attention mechanism (specifically the `flash_fwd_splitkv_kernel`) exhibits heavy-tailed latency growth under memory pressure. These findings provide micro-architectural evidence for the relationship, confirming that the attention operator's sensitivity to memory bandwidth is the primary driver of performance degradation under high context loads.

G Discussion on Countermeasures

In this section, we provide more analysis of potential countermeasures, ranging from inherent model constraints to active defense mechanisms. We specifically examine the behavior of Llama-3.1 under attack and discuss the feasibility of adversarial training versus system-level guardrails.

G.1 Impact of Strict Length Constraints

Here, we provide additional results of the *Fill and Squeeze* attack against **Llama-3.1-8B-Instruct** in Table 14. Surprisingly, the attack performance on Llama-3.1 is significantly weakened compared to Qwen3 or Gemma3-12B-it. The TTFT under our F&S attack (126.68 ms) as well as others are surprisingly lower than the benign baseline (144.16 ms), and the TPOT degradation is minimal. This resilience stems from the Llama's intrinsic configuration because Llama-3.1 enforces a maximum context length of **8,192 tokens** and tends to aggressively terminate generation earlier than its peers.

While imposing rigid constraints effectively neutralizes latency attacks by preventing the massive KV-cache accumulation required for the *Fill* phase, such measures are fundamentally at odds with the trajectory of LLM development. Specifically, as the industry aggressively pivots towards near-infinite context windows (e.g., 1M+ tokens) to power applications like RAG and repository-level code analysis, artificially capping generation length severely undermines service utility. Therefore, we posit that relying on hard limits is not a sustainable long-term defense strategy.

Table 14: Attack effectiveness on Llama-3.1-8B-Instruct. Unlike other models, Llama-3.1 exhibits resilience due to strict generation length constraints.

Method	TTFT (ms)	TPOT (ms)	Preempt (#)	Req.# (#)	Cost ($)
Benign (No Attack)	144.16	32.28	0	0	—
Engorgio	122.28	39.22	0	5576	1.25
LoopLLM	137.16	57.53	86	2489	4.16
ExtendAttack	131.05	45.10	0	1850	2.10
F&S+plain-text	**126.68**	**54.35**	**27**	**1006**	**3.73**

G.2 Defensive Guardrails and Adversarial Training

Beyond inherent model constraints, we analyze active defense mechanisms. A common proposal is the **Perplexity Filter** [2, 26], which detects adversarial inputs by analyzing the perplexity of the prompt. While effective against optimization-based attacks (e.g., Engorgio) that often produce gibberish, it fails against our plain-text prompts which are semantically coherent and natural.

Another theoretical solution is through **Adversarial Training**, where the model is fine-tuned on adversarial examples to learn robustness against resource-exhaustion triggers [29] and a recent study demonstrates success in defending against sponge examples with adversarial training [43]. However, applying this to LLM is practically infeasible due to the prohibitive computational cost of minimax training 70B+ parameter models and unknown risk of degrading general capabilities.

Instead of retraining, we advocate for a lightweight system-level defense inspired by efficiency robustness studies [5]. The most viable path is implementing **Behavioral Analysis** that monitor generation frequency. By tracking the ratio of output-to-input tokens per user, the system can identify tenants who consistently trigger worst-case generation lengths. If a user's request history exhibits a pattern of inducing maximum output lengths at a high frequency, they can be temporarily isolated or rate-limited [12]. This approach balances service availability with protection, addressing the vulnerability without the need for expensive model modifications.

Other defense methods involves re-designing the scheduler but requires a more thorough validation. For example, we could allow the scheduler to skip the HOL request if it cannot allocate blocks and admit smaller subsequent request more flexibly. This strategy directly breaks the memory-based HOL blocking but may potentially violate FCFS processing order, and adaptive attackers could also exploit $\mathcal{P}_{\text{low}}$ to occupy these smaller subsequent requests. On the other hand, the LIFO victim selection policy could be updated with a *largest-memory-consumer* policy, which is akin to behavioral analysis that make the attacker's own high-memory requests more likely to be preempted. However, it may increase the recomputation overhead for benign long-context requests. We can see that there is indeed an unavoidable trade-off between performance and robustness. We defer a holistic design and a thorough quantitative evaluation to the future work.

G.3 vLLM Configuration and Attack-Mechanism Mapping

Rather than sweep every combination of vLLM parameters, we summarize how each knob moves a specific part of the F&S vulnerability surface. This complements the empirical TP2/TP4/TP8 sweep in Fig. 8(a), which already validates the predicted capacity trend.

The parameters that determine *admission capacity* are `max_num_seqs`, `gpu_memory_utilization`, and the tensor-parallel degree. `max_num_seqs` sets the effective batch size, so it directly controls concurrent admission and aggregate KV demand; larger values raise the steady-state KV pressure and shorten the time-to-saturation for the Fill phase. `gpu_memory_utilization` together with tensor parallelism determines the total KV-cache capacity, and thus the Fill saturation threshold C_{sat}: higher capacity forces the attacker to pay more before reaching C_{sat}, and at TP8 the fixed-strength attack we evaluate is neutralized.

The remaining knobs primarily move the *cost profile* of a Squeeze cycle rather than its feasibility. `block_size` controls allocation granularity near saturation; smaller blocks reduce overshoot around C_{sat} and make the Squeeze-triggering margin δ finer-grained. The preemption mode (`recomputation` vs. `swap`) and the associated `swap_space` determine the per-preemption recovery cost analyzed in Corollary 1: recomputation makes each Squeeze event pay a $\Theta(L_v^2)$ prefill penalty,

while swap turns it into a PCIe-bandwidth-bound $\Omega(L_v/\mathrm{BW}_{\mathrm{PCIe}})$ transfer. Finally, the scheduling policy governs whether FCFS-HOL exists at all; skip-ahead admission or fair queuing removes the no-skip precondition of Proposition 3 and therefore the primary Fill-induced TTFT amplification, though Squeeze-related TPOT effects can persist because they arise from bandwidth contention rather than from admission order.

Taken together, these trends explain why over-provisioning (more capacity) and admission redesign (skip-ahead, fair queuing) are the two most impactful defense levers, whereas block-size and preemption-mode adjustments primarily change the cost profile of an attack rather than its feasibility. We do not report a full sensitivity sweep across every parameter combination and encourage operators to red-team their specific deployments using this mapping.

H Limitations

We list here the concrete scope boundaries of F&S and of the evaluation, complementing the "Scope and Boundary of Claims" paragraph in the main-text threat model (Section 5.1).

Evaluation scope. The main attack experiments—Table 1, all ablations, the real-world trace study, and the TP2/TP4/TP8 defense sweep—run on a single 8×L40S node. Multi-node evidence is limited to the passive prober observation on a two-replica production deployment reported in Appendix E.4; we did not launch active F&S attacks against production traffic. Pipeline parallelism, speculative decoding, and tensor parallelism split across nodes have not been validated, as those deployments can change token cadence, KV placement, and scheduler state in ways that our results do not attempt to characterize. Our baseline comparison is likewise with respect to the published fixed-schedule attacks (Engorgio, LoopLLM, ExtendAttack): a generic latency-feedback wrapper around each baseline is a different research question, and the component ablation in Appendix D.1 only shows that adaptive dispatch alone is insufficient to reproduce F&S's effect, not that every possible adaptive redesign has been surveyed.

Estimator and side channel. The LightGBM estimator is trained on a matching surrogate of the same model class and GPU type. Throughout the paper, "black-box" refers to the absence of victim-side access; it does not mean that a single frozen predictor transfers zero-shot across arbitrary hardware, and Table 12 quantifies the graceful degradation observed when the GPU, model, and network path change simultaneously. The closed loop also depends on visible streaming per-token timestamps: if a provider hides token-level timing through server-side buffering, or injects sufficiently large emission jitter, F&S degrades to a less efficient open-loop injection. We characterize this qualitatively in Section 6 but do not sweep a jitter-versus-degradation curve.

Framework and theoretical scope. Results are reported on vLLM 0.11.2 and SGLang commit `99c0b62`. Future releases may patch the specific `can_allocate()` and `add_one_req()` paths we analyse, and although the underlying Fill/Squeeze taxonomy should remain applicable, the numerical thresholds will shift. Finally, Propositions 1–3 and Corollary 1 identify sufficient conditions under which CB logical isolation, ITL–KV correlation, FCFS-HOL blocking, and preemption asymmetry emerge; they are explanatory models rather than universal queuing-theoretic guarantees, and their applicability depends on the deployment assumptions in Section 5.1.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state the main scope and contributions of the paper, including the system-level latency attack setting, the Fill-and-Squeeze strategy, and the evaluation on modern LLM serving systems.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper discusses limitations regarding the evaluated serving framework, hardware configuration, workload assumptions, black-box state estimation, and the fact that the empirical results are obtained in controlled local environments.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Propositions 1–3 and Corollary 1 explicitly state operating assumptions, and complete proofs are provided in the Appendix.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: The paper provides an anonymous link to the code and artifacts. Additionally, it provides the experimental setup, target models, serving framework version, workload construction, attack baselines, evaluation metrics, and additional implementation details in Section 5.3 and Appendix C.

Guidelines:

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

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

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

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

   Answer: [Yes]

   Justification: The paper provides detailed experimental descriptions and implementation details, but the full-scale attack code and scripts are not released at submission time due to responsible-release considerations for dual-use attack code.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: The paper specifies the serving framework, target models, GPU configuration, memory utilization setting, workload generation, datasets, baselines, and evaluation metrics in Section 5.3 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: The paper reports distributional latency metrics and repeated-run statistics for the main experiments; the reported variability reflects independent executions under the same workload and serving configuration.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The paper reports the GPU type, number of GPUs, GPU memory, CPU configuration, system memory, CUDA/driver versions, vLLM version, and key software dependencies in the experimental setup and Appendix C.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work is conducted in controlled local environments, does not attack public APIs/services, does not collect private user data, and includes an ethical considerations section discussing dual-use risks and defensive motivation.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The paper discusses both positive impacts, such as improving the robustness of LLM serving systems, and negative impacts, such as the potential misuse of system-level latency attacks against deployed services.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [Yes]

    Justification: The paper evaluates the attack only in isolated local environments and frames the release as a reproducibility and defensive-testing artifact. The authors avoid attacking public services and discuss responsible-use considerations for dual-use attack techniques.

    Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: The paper cites the existing models, datasets, serving frameworks, and software packages used in the experiments. Their versions and sources are described in the experimental setup and appendix, and the released artifacts follow the corresponding licenses and terms of use.

Guidelines:

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

13. **New assets**

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

Answer: [Yes]

Justification: The paper introduces experimental artifacts and code for reproducing the evaluation, documented through the anonymized artifact repository and the implementation details in the Appendix.

Guidelines:

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

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

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

Answer: [N/A]

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

Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve human-subject experiments, crowdsourcing, or collection of personally identifiable information, so IRB approval is not applicable.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: The paper uses open-source LLMs as experimental target models in the LLM serving-system evaluation. Generative AI tools were also used for limited manuscript preparation assistance, including language editing and formatting checks; all scientific claims, experiments, and final text were verified by the authors.

    Guidelines:

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