Selective KV-Cache Sharing to Mitigate Timing Side-Channels in LLM Inference

Kexin Chu[†]
University of Connecticut, CT, USA

Zecheng Lin[†]
Independent Researcher

Dawei Xiang
University of Connecticut, CT, USA

Zixu Shen
University of Connecticut, CT, USA

Jianchang Su
University of Connecticut, CT, USA

Cheng Chu
Indiana University Bloomington, IN, USA

Yiwei Yang
UC Santa Cruz, CA, USA

Wenhui Zhang
Independent Researcher

Wenfei Wu
Peking University, Beijing, China

Wei Zhang[*]
University of Connecticut, CT, USA

***Abstract*—Global KV-cache sharing has emerged as a key optimization for accelerating large language model (LLM) inference. However, it exposes a new class of timing side-channel attacks, enabling adversaries to infer sensitive user inputs via shared cache entries. Existing defenses, such as per-user isolation, eliminate leakage but degrade performance by up to 38.9% in time-to-first-token (TTFT), making them impractical for high-throughput deployment.**

To address this gap, we introduce SafeKV (Secure and Flexible KV Cache Sharing), a privacy-aware KV-cache management framework that selectively shares non-sensitive entries while confining sensitive content to private caches. SafeKV comprises three components: (i) a hybrid, multi-tier detection pipeline that integrates rule-based pattern matching, a general-purpose privacy detector, and context-aware validation; (ii) a unified radix-tree index that manages public and private entries across heterogeneous memory tiers (HBM, DRAM, SSD); and (iii) entropy-based access monitoring to detect and mitigate residual information leakage.

Our evaluation shows that SafeKV mitigates 94%–97% of timing-based side-cahnnel attacks. Compare to per-user isolation method, SafeKV improves TTFT by up to 40.58% and throughput by up to 2.66× across diverse LLMs and workloads. SafeKV reduces cache-induced TTFT overhead from 50.41% to 11.74% on Qwen3-235B. By combining fine-grained privacy control with high cache reuse efficiency, SafeKV reclaims the performance advantages of global sharing while providing robust runtime privacy guarantees for LLM inference.

I. Introduction

Large language models (LLMs) now underpin applications from dialogue to complex reasoning. To meet time-sensitive inference demands, key–value (KV) caching stores intermediate attention states (“keys” and “values”) to eliminate redundant computation for sequential or similar prompts, thereby accelerating generation [70]. This efficiency gain is amplified through KV cache sharing across multiple requests. In particular, prompts with common prefixes, such as shared dialogue history or structured prompting patterns, enable substantial throughput improvements and latency reduction. Consequently, KV-cache sharing has become a critical mechanism for boosting throughput and reducing response latency in large-scale, multi-user LLM deployments. Empirical studies confirm that a substantial portion of real-world prompts exhibit prefix-level or structural overlap [42], [74], making shared KV reuse both practical and highly beneficial.

Despite these performance benefits, KV cache sharing raises serious privacy and security concerns in shared or multi-tenant deployments. Specifically, KV-cache sharing across mutually untrusted users can lead to unintended information leakage. Recent research has shown that adversaries can infer cache hits by issuing carefully crafted queries and measuring response latencies. These timing variations can leak whether a particular prefix has been previously cached, indirectly exposing the query patterns of others. Over time, such cache probing attacks can reveal partial or even complete user inputs [54], [57], [60], [73], posing serious threats to user privacy.

These side-channel attacks are particularly alarming for two key reasons. First, they require no special privileges: the attacker simply interacts with the LLM through its standard API, mimicking normal user behavior. Second, the leaked content often contains highly sensitive data, such as medical questions, financial details, or private instructions to LLM agents. Given the low barrier to attack and the high severity of potential data leakage, KV-cache-based side channels represent a pressing and under-addressed security threat in real-world LLM deployments.

The severity and practicality of these attacks have been demonstrated across several recent works. For example, PromptPeek [60] and InputSnatch [73] both exploit token-wise probing and cache-hit timing feedback to reconstruct private user prompts with high accuracy. Song et al. [54] further ex-

† These authors contributed equally to this work.
∗ Corresponding author (wei.13.zhang@uconn.edu).

pand the threat landscape by uncovering timing-based leakage in a broader range of caching schemes, including semantic and partial-match reuse, across open-source and commercial LLM systems. Collectively, these works expose a new class of LLM-specific vulnerabilities that arise from performance-oriented cache reuse mechanisms.

In this paper, we propose SafeKV , a secure and efficient KV-cache management system designed to mitigate prompt leakage in LLM inference. Unlike approaches that enforce strict user-level cache isolation [49], which would sacrifice performance, SafeKV adopts a selective reuse strategy. At the time of cache block creation, the system classifies each entry as either sensitive or safe for reuse, based on the privacy characteristics of the prompts. This classification is driven by a multi-tier, extensible privacy detection pipeline that analyzes both token content and contextual semantics. Only those KV entries deemed safe are eligible for cross-user reuse, enabling SafeKV to preserve the performance advantages of caching while enforcing strong privacy boundaries between users.

However, designing such a system raises three challenges:

- **Challenge 1: Accurate and Efficient Privacy Classification.** How can the system reliably differentiate sensitive from non-sensitive KV entries with low latency and computational overhead? Rule-based methods offer low cost but limited recall, while deep learning models improve accuracy but introduce inference delays.
- **Challenge 2: Risk Mitigation under Imperfect Detection.** Even rare misclassifications can lead to privacy violations if sensitive content is mistakenly shared. How can the system detect and respond to such leakage risks in real time?
- **Challenge 3: Scalable Cache Lifecycle Management.** How should the system organize private and shared caches to maximize reuse without sacrificing security? Effective cache management must support fast prefix matching, minimize redundancy, and avoid structural fragmentation.

To tackle these challenges, SafeKV rethinks KV-cache design from a privacy-first yet performance-aware perspective. Rather than treating privacy enforcement as an add-on, it integrates security into the cache lifecycle itself. Through selective isolation and asynchronous detection, SafeKV ensures that sensitive content remains confined without penalizing overall system throughput. The result is a scalable and efficient LLM serving infrastructure that offers strong privacy protection alongside high-performance inference.

- **Privacy-Aware KV-Cache Classification.** We design SafeKV to enable fine-grained KV-cache sharing by classifying cache entries as either safe or sensitive at creation time. To achieve accurate and efficient classification, we introduce a multi-tier hybrid detection framework combining rule-based matching, general privacy detector, and context-aware validation. The detection pipeline is asynchronous, extensible, and optimized for high-throughput LLM serving.

TABLE I: Cache Mechanisms Comparison of Different LLM API Vendors

Vendor/Frameworks	Stream	Caching Mechanisms	Cache Lifetime
OpenAI [48]	Y	Prefix Caching	5–10 minutes
DeepSeek [31]	Y	Prefix Caching	Hours to days
Anthropic Claude [24]	Y	Prefix Caching	5 minutes
Google Gemini [30]	Y	Prefix Caching	Default 1 hour
MoonShot Kimi [50]	Y	Prefix Caching	Customization
vLLM [40]	Y	Prefix Caching	Customization
SGLang [72]	Y	Prefix Caching	Customization

- **Resilient Defense Against Misclassification.** To mitigate the risks of imperfect privacy detection, SafeKV incorporates runtime safeguards including cache-level isolation by default and fallback handling for suspicious access. It ensures that sensitive data remains confined even under partial detection errors, preserving privacy without interrupting inference execution.
- **Scalable and Efficient Cache Management.** We implement a privacy-aware caching subsystem using a unified radix-tree structure that supports both private and shared entries. It features memory-efficient indexing, adaptive reuse policies, and multi-tier memory coordination, enabling high reuse rates, fast lookups, and minimal performance overhead across large-scale deployments.

II. Background

A. LLM Inference

Large Language Models (LLMs), built on the Transformer architecture [59], compute contextual relationships using scaled dot-product attention over Query (Q), Key (K), and Value (V) vectors [38], [43]. Inference proceeds in two stages: the *Prefill* phase processes the entire prompt in a single forward pass to compute K/V embeddings and generate the first output token. The subsequent *Decoding* phase generates one tokens at a time, reusing cached K/V embeddings to apply attention over the growing sequence. This KV-cache mechanism reduces decoding complexity from quadratic to linear in sequence length, significantly improving efficiency for long prompts and multi-turn dialogue [35].

B. KV-Cache Sharing

To improve inference efficiency, modern LLM serving systems widely employ *KV-cache sharing*, allowing reuse of previously computed key/value embeddings across requests with overlapping prompt prefixes. As summarized in Table I, most commercial and open-source systems support prefix-based caching, where shared prefixes enable subsequent requests to skip redundant computation. This leads to substantial latency reductions.

Cache retention policies vary across deployments. OpenAI and Anthropic retain cache entries for only a few minutes, while DeepSeek extends cache lifetime to hours or days. Others, such as Google Gemini and MoonShot Kimi, adopt flexible or tunable policies. Open-source frameworks like vLLM and SGLang expose fine-grained controls over cache eviction and reuse. Despite these differences, most systems follow the same underlying paradigm: prefix-based caching and reuse to accelerate inference.

C. Timing Side-Channel Attack

While shared KV caching significantly improves inference performance, it introduces a subtle but serious vulnerability: a software-level timing side channel. By measuring the response latency of queries, particularly the time to first token (TTFT), an attacker can infer whether certain prefix tokens were previously cached, revealing information about other users' inputs. Prior works [29], [53], [54], [60], [65], [73] demonstrate that such attacks can be mounted even under black-box API access:

- **Probe.** The attacker submits a crafted prompt and records the TTFT.
- **Detect.** A low TTFT suggests the prefix was already cached (cache *hit*); a higher latency indicates cache *miss*.
- **Reconstruct.** By iteratively extending the prefix and probing candidates token-by-token, the attacker identifies the correct next token based on TTFT differences, gradually recovering sensitive prompt content.

Notably, this vulnerability is not limited to exact prefix-matching caches. Systems like GPTCache [25], which reuse responses for semantically similar inputs, also exhibit latency-based leakage: similar queries produce faster responses if related prompts were previously cached. These attacks require no privileged access, only the ability to issue queries and measure response time, yet have proven effective against both commercial and open-source LLM services, exposing a fundamental tension between inference efficiency and user privacy in shared environments.

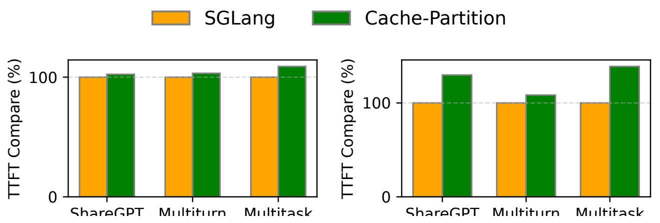
D. Motivation: Privacy Risk and Isolation Cost

a) ***The Risk of KV-Cache Leakage.****:* To access the privacy risks posed by structured privacy-sensitive data in large language model (LLM) training corpora, we analyze two widely adopted datasets: C4 [51] and Pile [37]. As summarized in Table II, these datasets contain substantial volumes of personally identifiable information (PII), including usernames, phone numbers, credit card details, and US Social Security Numbers. For example, the C4 dataset alone contains more than 1.4 billion occurrences of usernames, and the Pile dataset includes approximately 69 million instances of US bank account numbers. These findings underscore the widespread presence of sensitive data in real-world corpora, rendering inference-time leakage through shared components like the KV cache a realistic and pressing concern.

These statistics shows a tangible threat: if KV-cache entries containing such structured PII are shared across users, attackers exploiting pattern-guided probing can efficiently detect the existence of sensitive content within shared caches, potentially reconstructing sensitive content. Therefore, protecting privacy-sensitive tokens from unintended reuse becomes a core requirement in multi-tenant LLM deployments.

TABLE II: Personal Information Counts in C4 and Pile.

Personal Information Type	C4	Pile
User Name	1,444,683,066	3,273,163,949
Phone Number	19,592,273	23,191,595
Email Number	9,056,833	13,336,793
US Bank Number	7,139,838	69,763,678
Credit Card Number	61,405	741,815
US SSN	2,352,339	12,541,022
IP Address	1,890,090	14,975,663
Total	1,484,780,621	3,407,722,116



(a) Llama-2-13B (b) Llama-2-70B

Fig. 1: Normalized performance of TTFT between *SGLang (global-sharing)* and *Cache-Partition (isolated-per-user).*

b) ***Performance Impact of Full Isolation.****:* A straightforward mitigation strategy is to enforce per-user cache isolation, thereby preventing any possibility of cross-user KV reuse for sensitive content [49]. However, this comes at a steep cost: isolating the KV cache for each user forfeits the computational and memory benefits afforded by cache sharing [44], [63]. Specifically, it results in redundant storage of identical prefixes and diminishes memory efficiency across the multi-tiered memory hierarchy (HBM, DRAM, and SSD). Given that real-world LLM workloads frequently exhibit significant cross-user query reuse(as shown in Table III), per-user isolation introduces severe performance overheads. Our motivation experiments, as shown in Figure 1, reveal that the Cache-Partition(per-user isolation) increases Time-to-First-Token (TTFT) by 2.3% to 8.9% for LLaMA-2-13B [7] and by 8.3% to 38.9% for LLaMA-2-70B [8] across three different datasets.

TABLE III: Intra-session and Inter-session KV-Cache reuse rates across different datasets.

Dataset	**Intra-User Reuse**	**Inter-User Reuse**
ShareGPT V3 [1]	7.06%	25.49%
Multiturn Chat [2]	31.47%	9.45%
Prompt Multitasks [34]	0.0%	63.10%

To reconcile the trade-off between low-latency cache reuse and strong privacy isolation, we introduce **SafeKV** , a selective KV-cache sharing framework that distinguishes between private and non-private KV cache entries. Sensitive content is confined to user-specific private caches, while non-sensitive data is allowed to be shared safely across requests. This selective strategy achieves robust privacy protection without sacrificing the performance benefits of shared caching, striking a balance between security and efficiency.

III. Threat Model

A. System Model

Multi-tenant LLM serving spans local to cloud deployments. A core constraint across settings is limited GPU memory versus large per-request KV footprints, which hinders batching. Prior work mitigates this by *sharing* KV across requests with identical prefixes [41], [62], [71], improving concurrency and latency [41], [62], [71]. KV for token t_k is reusable across requests iff all preceding tokens match exactly. Hence, prefix mismatches (especially at the first token) preclude sharing. For example, **vLLM** [16], [41] materializes KV *blocks* tagged by hashes, timestamps, and refcounts; retains while memory allows, evicting oldest on pressure. **SGLang** [15], [71] stores KV in a radix tree on GPU; uses LRU eviction and *Longest Prefix Match* (LPM) scheduling to prioritize requests with longer shareable prefixes.

The scheduler impacts the KV cache sharing. Users submit requests to an inference server. A scheduler batches queued requests and dispatches them to a GPU worker; results are streamed back to users. N users issue requests r at frequency f. Each r has i input tokens $\{t_1, \ldots, t_i\}$; the server returns j output tokens $\{t_{i+1}, \ldots, t_{i+j}\}$, with j configurable (e.g., `max_tokens` in vLLM) [15], [16]. Orders requests by a scheduling policy P_S and forms a batch $b = \{r_1, \ldots, r_m\}$ under batching policy P_B. KV handling follows P_{KV}; responses follow P_O. A single LLM runs on one GPU. Memory M is partitioned into model M_{model}, KV M_{KV}, and other activations M_{others} [41]. With per-token KV size m_t (model-dependent), the concurrent token capacity is

$$T_{\max} = \left\lfloor \frac{M_{\text{KV}}}{m_t} \right\rfloor.$$

When KV sharing is enabled, entries persist while space permits; upon reaching $T_{\max}$, eviction proceeds per eviction policy P_E [60]. Scheduling policy P_S, batching policy P_B, eviction policy P_E, KV-cache policy P_{KV}, and output policy P_O are vendor-specific, evolve rapidly across serving stacks, and are often undocumented. Unlike prior work [60] that assumes these policies are public and fixed, we treat them as closed and time-varying; our analysis does not rely on their internals and remains robust to implementation-defined changes.

B. Threat Model

*1) **Attackers' Goal**:* An adaptive adversary seeks to recover prompt-borne secrets (e.g., names, emails) by exploiting timing side channels in shared LLM serving systems [54], [73]. As illustrated in Figure 2, the adversary uses standard APIs against the same backend as benign users and infers sensitive content from latency fluctuations—notably TTFT, which is induced by KV-cache reuse.

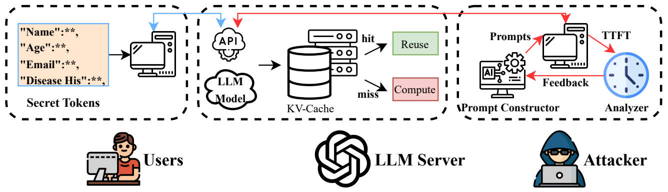


Fig. 2: Attack Overview.

*2) **Attackers' Knowledge**:* We assume a black-box adversary with no access to model internals or victim request contents, but who possesses *structural priors* over common secret formats (e.g., email addresses, phone numbers, SSNs), *tokenizer knowledge* to map candidate strings to the deployed model's token sequences using public tokenizers, and an understanding of *shared KV-cache semantics and reuse conditions*. The adversary is limited to API-visible outputs and timing signals (e.g., TTFT) and cannot observe model parameters, logits, or KV-cache states.

*3) **Attack Strategy**:* The attack proceeds in an iterative, adaptive probing loop designed to reconstruct user secrets using side-channel leakage. The methodology is illustrated in Figure 2 and follows these stages:

- **Prompt Construction:** The attacker prepares a set of candidate prompts with shared prefixes likely to overlap with the vitim's orivacy query. Candidates may be derived from sensitive data templates or synthetically generated via local LLMs.
- **Timing Probing:** Each candidate is submitted to the inference backend. The attacker records TTFT to determine whether prefix tokens were already cached, exploiting reduced latency as a leakage signal.
- **Adaptive Refinement:** Using TTFT feedback, the attacker prunes unpromising candidates and recursively extends the prefix until the complete sensitive content is inferred.
- **Distributed Evasion:** To bypass rate limiting or entropy-based defenses, the attacker can distribute probes across multiple user identities and schedule them non-uniformly.

C. SafeKV Approach

*1) **Defense Goals**:* SafeKV seeks to prevent timing-based side-channel leakage stemming from shared KV-cache reuse in multi-tenant LLM inference. Our defense is designed to achieve the following objectives:

- **G1: Privacy Preservation.** Ensure that KV-cache blocks containing privacy-sensitive content are never reused across user boundaries.
- **G2: Performance Retention.** Selectively share safe KV blocks to preserve the throughput and latency benefits of cache reuse.
- **G3: Runtime Resilience.** Detect and mitigate residual misclassifications via access pattern monitoring and adaptive isolation.

2) ***Defense Knowledge:*** SafeKV assumes a *black-box* threat model and operates with limited observability. Specifically:

- **No privileged access:** The system does not rely on internal model parameters, gradients, or logits.
- **Token-level visibility:** Observes tokenized prompts and per-request cache metadata (e.g., prefix length, reuse count).
- **Detection priors:** Leverages pattern-based matching, lightweight privacy detectors, and context-aware LLM validators to identify sensitive inputs.
- **Behavioral signals:** Tracks KV reuse entropy and user distribution drift to detect anomalous access patterns at runtime.

3) ***Proof of Defense:*** Let $r_v = t_1, t_2, \ldots, t_k$ denote a victim prompts prefix and $C(t_1, t_2, \ldots, t_k)$ indicate whether the corresponding KV-Cache block is publicly visible. An attacker issues prob request r_a with matching prefix and observe the TTFT latency:

$$\text{TTFT}(r_a) = \begin{cases} T_{hit} & \text{if } C(t_1, t_2, \ldots, t_k) = \texttt{True} \\ T_{miss} & \text{otherwise} \end{cases} \quad (1)$$

SafeKV ensures that $C(t_1, t_2, \ldots, t_k) = \texttt{True}$ only if the block has passed all detection stages. Let α_i denote the false negative rate of detection $Tier - i$ $(i \in \{1, 2, 3\})$. Then the probability of a sensitive block being publicly exposed is:

$$P_{\text{leak}} \leq \prod_{i=1}^{3} \alpha_i \quad (2)$$

Empirically, we observe $\alpha_1 < 0.63$, $\alpha_2 < 0.04$, and $\alpha_3 < 0.29$, even at worst cas, yielding $P_{\text{leak}} < 0.03$ in practice (detailed in Section VII).

In the presence of such leakage, SafeKV monitors entropy $\mathcal{H}_b$ of block b:

$$\mathcal{H}_b = \frac{u_{\text{cnt}}}{\text{hit}_{\text{cnt}}} \quad (3)$$

where u_{cnt} is the number of unique user accesses and hit_{cnt} the total access count. A rising entropy combined with historical low u_{cnt} triggers a downgrade of b from public to private, terminating reuse and bounding leakage exposure over time.

Collectively, these mechanisms ensure that: leftmargin=*

- Only KV blocks passing strict multi-tier detection are reusable across users.
- Any residual misclassification is detected within bounded usage epochs.
- Private blocks never induce observable latency variation across users.

IV. Overview of SafeKV

SafeKV is a privacy-aware KV-cache management framework that enables safe cache sharing by isolating sensitive content while maximizing reuse of non-sensitive entries. Its design addresses three key challenges outlined in Section I: (1) Accurate and Efficient Privacy Classification, (2) Risk Mitigation under Imperfect Detection, and (3) Scalable Cache Lifecycle Management.

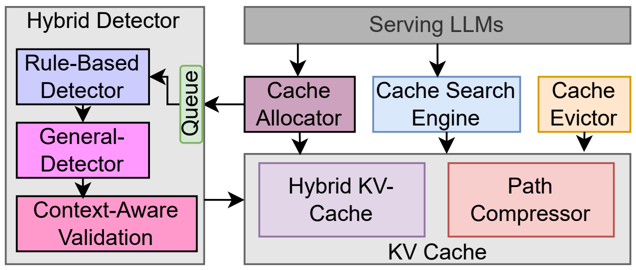


Fig. 3: The Architecture Overview of SafeKV .

A. Design Objectives

To address these challenges, SafeKV is guided by the following system goals:

- **G1: Privacy-Aware Reuse.** Maximize KV-cache reuse opportunities across users without compromising privacy.
- **G2: Minimal Latency Impact.** Ensure that detection and access control introduce negligible delay to LLM inference.
- **G3: Scalability.** Maintain efficiency and correctness across large-scale deployments with diverse user traffic patterns and cache workloads.

B. System Architecture

To enforce fine-grained privacy control without undermining inference efficiency, SafeKV adopts a modular architecture centered around two core components, each tailored to address specific system-level challenges introduced in Section I.

1) **Hybrid Detection Pipeline. (addresses Challenge 1 & 2)** A multi-tier privacy classification system that evaluates each KV-cache block at allocation time. It includes: (i) Tier-1 rule-based pattern matching, (ii) Tier-2 lightweight transformer-based detection, and (iii) Tier-3 context-aware validation using large models. The detection pipeline is asynchronous and incorporates runtime fallback mechanisms to ensure coverage without stalling inference (Section V).
2) **SafeKV Memory Manager. (addresses Challenge 3)** A radix-tree–based KV-cache index that supports unified management of public and private entries. It incorporates path compression for efficient lookups and coordinates eviction based on sensitivity and reuse (Section VI).

These two components work in tandem: the detection pipeline classifies cache entries based on privacy semantics, and the memory manager enforces fine-grained sharing policies accordingly. Together, they enable SafeKV to support secure, scalable, and low-latency KV-cache sharing in LLM serving systems.

V. SafeKV-Detect: Hybrid Privacy Detection

In this section, we present the design of SafeKV 's privacy detection subsystem. This component is responsible for classifying each KV-cache block as private or shareable at allocation time and enforcing runtime safeguards against potential misclassifications.

A. Design Requirements

SafeKV 's detection system is designed to satify the following requirements:

- **R1: Lightweight and Extensible.** Detection must incur minimal overhead to support real-time inference and remain adaptable to new privacy policies and domains.
- **R2: Accurate and Context-Aware.** The system must detect both explicit and subtle privacy-sensitive content, including implicit identifiers embedded in conversational context.
- **R3: Resilient Against Misclassification.** False negatives must be detected and contained through runtime behavior monitoring and anomaly-aware fallback strategies.
- **R4: Asynchronous and Non-Blocking.** Detection must be decoupled from the main serving path to avoid degrading latency or throughput.

To meet these goals, SafeKV employs a three-tier hybrid detection pipeline (Section V-B) for challenge 1, a runtime fallback mechanism (Section V-C) for challenge 2, and an asynchronous scheduling strategy (Section V-D) for efficiency.

B. Three-Tier Hybrid Detection Strategy

Each newly generated KV-cache block undergoes a three-stage classification process at creation time. The pipeline is structured as follows:

- **Tier 1: Rule-Based Pattern Matching.** This stage efficiently captures explicit sensitive content using configurable pattern matching techniques.
- **Tier 2: General Privacy Detector.** This stage employs a compact transformer model to assess the privacy sensitivity of text blocks not flagged by regex or heuristics. This scoring stage balances detection accuracy, latency and model size to support efficient detection.
- **Tier 3: Context-Aware Validation.** This stage combines the context (including the history of multi-turn conversations) to evaluate the sensitivity. Avoid leaking sensitive information when combined with its context, which is hard to be detected by tier 1/2. This ensures the robustness of semantically related phrases or multi-turn conversations.

This tiered architecture allows SafeKV to balance detection accuracy, extensibility, and performance across diverse input types and usage scenarios.

1) *Rule-Based Pattern Matching*:

We first deploy a rule-based detection stage to efficiently identify explicit sensitive content using configurable pattern-matching methods.

This initial defense combines regular expressions with customizable blacklists. Regular expressions capture structured variable-length data such as emails, phone numbers, and IDs, while blacklists manage fixed-format terms like internal project codes or organization-specific tokens. This dual approach ensures rapid and high-coverage detection of known privacy-sensitive patterns.

TABLE IV: Taxonomy of User-Related Data Categories in Privacy Detection

Category	Type	Examples
Privacy	General Information	Nickname, avatar, signature
(Personal)	Basic Information	Third-party account information
	Identity Information	ID card, passport, driver's license, SSN
	Location Information	Country, region
	Biometric Identification	Fingerprints, face, voiceprint, iris, gene info
	System/Network Identification	UserID, IP, Cookie, RFID, password, certs
Device	Software Device Information	Android ID, IDFA, IDFV, OS, region
	Hardware Device Information	MAC, IMEI, GUID, serial number
Profile	Cultural & Social Info	Job, education, qualification certificates
	Financial Info	Bank account, property, loan records
	Social Info	Likes/follows, contacts, collections
	Service Content Info	Browsing/purchase/download records
Behavior	Service Log Info	Login, behavior, purchase logs

TABLE V: Comparison of lightweight fine-tuned models for PII detection

Model	Base Arch.	Size	Accuracy	Token vs Seq	Langs / Types
DistilBERT-PII [5]	DistilBERT-base	66M	95.22%	Token-level	1 lang / 5 types
Piiranha-v1 [6]	DeBERTa-v2-base	125M	99.44%	Token-level	6 langs / 17 types
PII-BERT-base [3]	BERT-base-cased	110M	99.11%	Token-level	English / gen data
dbert-pii-det. [14]	DistilBERT-uncased	66M	94.33%	Token-level	mixed syn.
DePrompt [55]	ChatGLM2-6B	6B	95.95%	Sequence-level	Chinese
GPT-4o-mini [52]	GPT-4o-mini	1B	98.95%	Sequence-level	English / Edu

Designed for extensibility and ease of operation, the detection module supports a pluggable, configuration-driven framework. Developers can dynamically register new privacy rules via regular expressions or update blacklists through a standardized interface. The system features hot-reloading capabilities, allowing new rules to take immediate effect without service downtime, providing agility for evolving privacy needs.

Table IV summarizes the current categories of user-related data types already supported in the rule-based matching stage. These include structured privacy identifiers (e.g., names, IDs), device metadata, behavioral logs, and profile-related content. Each entry in the table corresponds to a category that is currently covered by our default pattern library, either through regular expressions or Trie-based detection logic.

2) *General Privacy Detector*:

While SafeKV 's Tier-1 employs configurable rule-based detection for identifying common PII, such approaches inherently suffer from limited coverage. First, they rely on predefined patterns and regular expressions, which cannot generalize to diverse or ambiguous forms of sensitive information. Second, they fail to detect context-dependent privacy risks—such as personal addresses, medical history, or relational descriptors, that do not conform to rigid syntactic structures. To address these limitations and improve coverage of latent privacy leakage, SafeKV introduces a Tier-2 detector based on a lightweight language model capable of generalized privacy classification.

To determine an effective model for this purpose, we firstly conducted a comprehensive evaluation of fine-tuned transformer-based detectors in terms of resource efficiency (model size and memory footprint), multilingual and multi-type PII coverage, and detection accuracy. As summarized in Table V, models such as *DistilBERT-PII*, *PII-BERT-base*, and *Piiranha-v1* demonstrate strong overall performance, achieving over 93% accuracy across multiple languages and PII categories. Among them, *Piiranha-v1* strikes the best balance, offering multilingual detection (6 languages, 17 types) with 99.44% accuracy and only 125M parameters, making it an attractive candidate for lightweight inference.

However, traditional fine-tuned classifiers still exhibit several limitations: (1) their token-level classification often leads to false positives and fragmented predictions, (2) they generalize poorly to real-world inputs beyond their training domains, their detection accuracy degrades significantly when encountering unseen or rare PII categories. For instance, on a curated subset of the `pii-masking` [4] containing 16 PII types unsupported by *Piiranha-v1* (e.g., SSN, IPV4, PHONEIMEI), Piiranha-v1 achieves only a 33.43% detection rate.

To overcome these issues, we explored compact, general-purpose LLMs such as the Qwen3 [18], [22], [23] and LLaMA-3 [9]–[11] series (details in Appendix A) and benchmarked them for robustness, accuracy, and inference cost. Based on these evaluations, we selected **LLaMA-3.2-1B** as the final detector in Tier-2, which achieves an accuracy of near 100% on the same test set. This model offers improved adaptability to diverse prompt contexts, stronger resistance to out-of-distribution inputs, and maintains low inference latency suitable for real-time deployment.

3) ***Context-Aware Validation****:*

While rule-based and lightweight model detectors (Tier-1 and Tier-2) effectively capture explicit or localized privacy indicators, they often fail in scenarios involving implicit cues, cross-segment dependencies, or multi-turn conversational flows. To address these limitations, SafeKV introduces a context-aware validation mechanism that leverages the in-service LLM already deployed for user inference. For KV cache blocks whose privacy status remains uncertain after initial filtering, SafeKV constructs enriched prompts by embedding relevant conversational history. These prompts are then evaluated by the in-service LLM to assess the overall privacy sensitivity of the contextual input.

This design is motivated by the observed shortcomings of earlier detection stages in handling complex and semantically subtle inputs. As demonstrated in Section VII-B0b, small-scale models often miss context-dependent privacy disclosures, whereas larger LLMs exhibit greater semantic understanding and contextual reasoning capabilities. However, deploying a separate LLM solely for validation is impractical due to the substantial infrastructure cost, especially since only a small fraction of requests reach this stage.

To balance detection accuracy and system efficiency, SafeKV reuses the existing inference LLM as the backend validator. By exploiting the framework's native scheduling and KV-cache reuse mechanisms, this design enables seamless integration of context-aware validation into the primary inference pipeline with minimal latency overhead. As a result, SafeKV achieves high-precision detection of nuanced privacy risks without incurring additional deployment cost or compromising overall system throughput.

C. Fallback Protection and Attack Mitigation

Despite employing a hybrid, multi-stage detection pipeline, SafeKV acknowledges that the privacy classification may occasionally fail, either due to nuanced semantic ambiguity, incomplete pattern coverage, or model's limitation. To safeguard against these residual risks, SafeKV incorporates a runtime anomaly-aware fallback mechanism that continuously monitors KV-cache access behaviors to detect and respond to suspicious activity post-deployment.

At the core of this runtime defense is a lightweight statistical monitor that continuously observes access behaviors to each KV-cache block. Specifically, SafeKV maintains a rolling window of metadata for each entry, recording the current hit count (*hit_cur*), the number of unique user identifiers (*u_cnt*), the previous hit count (*hit_pre*), and the previous number of unique users (*u_pre*). These values are used to compute the distribution entropy $entropy = u\_cnt/hit\_cur$ that reflects the dispersion of access: low entropy indicates access concentration (e.g., frequent hits by a single user), while high entropy suggests broad usage across accounts.

Upon detecting significant shifts in access patterns, SafeKV apply differentiated mitigation strategies based on the historical reuse pattern of the KV-cache block and the user type involved. If the block previously exhibited broad reuse (*u_pre*), the access pattern change is considered reasonable. However, if historical reuse is minimal ($u\_pre \approx 1$), the change is treated as suspicious:

- *Customer-owned blocks* are immediately reclassified as private to prevent potential leakage.
- *Business-owned blocks* undergo moderated traffic control and user authentication, including Zero Trust Identity (ZTI), certificate validation, and attestation. Alerts are also issued to enterprise clients to enable collaborative decisions regarding cache status adjustment.

This entropy-based runtime defense complements static detection by offering adaptive, context-sensitive protection, ensuring robust privacy safeguards while preserving system flexibility.

D. Asynchronous Detection and Streaming-Aware Scheduling

To prevent classification from degrading inference latency, SafeKV fully decouples the privacy detection pipeline from the critical execution path of LLM serving. All KV-cache blocks are initially assigned a private label by default, allowing the system to proceed immediately with inference. Privacy classification is executed asynchronously in a separate thread or coroutine, operating in batched mode to improve efficiency and exploit parallel hardware resources.

An adaptive thresholding mechanism is employed to interpret transformer scores in Stage 2. Instead of relying on fixed cutoffs, SafeKV adjusts thresholds dynamically based on system load, historical detection outcomes, and sensitivity score distributions. For instance, under high workload or suspected attack patterns, the system lowers the threshold for classifying blocks as private to favor conservatism. Conversely,

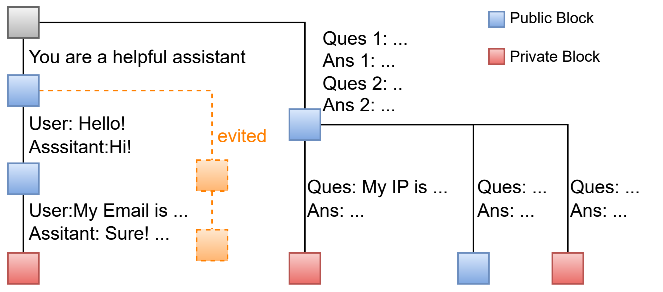


Fig. 4: The Unified Privacy/Public Cache Index Tree.

during normal operation, the thresholds may be raised slightly to improve cache reuse. This flexibility helps SafeKV strike a balance between false negatives and excessive over-isolation.

Once detection completes, blocks with confirmed public labels are reclassified and made available for cross-user reuse. Private blocks remain isolated. Furthermore, the privacy classification results are recursively propagated to descendant nodes in the prefix tree, avoiding repeated evaluations for blocks with inherited sensitivity.

This asynchronous and pipelined detection strategy ensures that privacy guarantees are enforced without compromising inference responsiveness. It enables SafeKV to remain both performant and privacy-preserving in multi-user, real-time LLM environments

VI. SafeKV-Cache: Privacy-Aware Cache Management

In this section, we present the memory system design of SafeKV . The cache layer must enforce strict privacy boundaries while supporting high-throughput prefix matching, multi-tier storage coordination, and efficient memory reclamation. To meet these requirements, SafeKV built a unified radix-tree–based cache index with privacy-aware access control, path-aware memory optimizations for private entries, and a progressive eviction strategy.

A. Unified Privacy-Preserving Cache Index

To balance memory efficiency and privacy isolation, SafeKV introduces a unified KV-cache management mechanism based on an extended radix tree index. Inspired by the prefix-matching architecture in SGLang, SafeKV enhances this structure to support both global (public) and user-specific (private) KV-cache blocks within a single scalable and privacy-aware hierarchy.

As shown in Figure 4, all KV-cache entries—regardless of their visibility—are organized under a unified radix tree in HBM/DRAM/SSD. Each node is annotated with two key metadata fields: `private_tag` (0 for public, 1 for private) and `creator_id` (user identifier). These annotations enforce fine-grained access control during cache lookups: public entries are accessible to all users, while private entries are only visible to their creator.

- **Insert.** New KV-cache blocks are incrementally inserted into the radix tree after decoding.

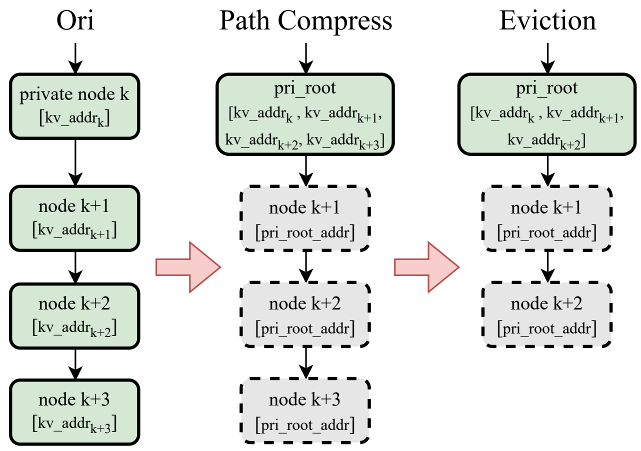


Fig. 5: An Example of *Path Compression* and *Progressive Eviction*.

- **Search.** Lookups traverse the radix tree, verifying each node's `private_tag`. Public nodes are accessible to all users, whereas private nodes require the querying user's identity to match the node's stored `creator_id`. This ensures private entries remain logically isolated even if they share prefixes with public entries.
- **Evict.** Eviction follows an LRU-based policy, removing least recently used leaf nodes first. Private nodes are incrementally pruned to avoid prematurely removing entire user-specific branches, while public nodes rely on reference counting to preserve frequently reused paths.

B. Private Tree Optimization and Progressive Eviction

While the unified radix tree structurally consolidates all KV-cache entries, SafeKV further optimizes private subtrees by leveraging their linear structure and user-specific access patterns. Unlike public blocks, which frequently branch and interleave, private blocks generally form linear, single-user paths, enabling two key optimizations: path compression and progressive eviction.

1) Path Compression: Figure 5 illustrates the path compression process. For private nodes forming single-user, non-branching paths, SafeKV compresses them into a single node. The root node of the compressed path (*pri_root*) aggregates descendant *kv_cache_addresses* into a single list and is flagged as *is_compressed = true*, which instructs the search engine to terminate further traversal upon reaching this node and directly use the cached address list for inference.

To enable reversible and trackable compression, all descendant nodes are retained in the tree and updated as follows: (1) their *after_compress* flag is set to true, indicating that they are inactive and serve only metadata roles, and (2) their *kv_cache_address* is replaced with a reference to the *pri_root* node. These references allow the system to locate the *pri_root* quickly during subsequent eviction, ensuring that memory cleanup affects the correct storage entries. SafeKV also updates the metadata of the *pri_root* to reflect the cumulative KV-cache memory usage of the compressed subtree.

2) Progressive Eviction: SafeKV adopts a bottom-up eviction strategy, targeting leaf nodes with the oldest access epochs, tracked via a global *epoch_counter*. In compressed subtrees, inactive leaf nodes (*after_compress* = *true*) are prioritized. Upon eviction, the corresponding entry in the parent node's (*pri_root*) compressed address list is removed, gradually pruning the subtree while preserving reusable upstream prefixes. Only after all descendant nodes have been evicted does *pri_root* itself become eligible for removal, preventing premature loss of reusable private contexts.

3) Epoch-Based LRU with Privacy-Aware Priority.: To manage eviction order efficiently, SafeKV uses an epoch-based LRU approximation. Each node stores its most recent *access_epoch*, and a global *epoch_counter* advances periodically. Nodes with the largest *epoch_delta* (i.e., oldest usage) are prioritized for eviction. Among nodes with identical age, SafeKV evicts public nodes first—based on the assumption that private blocks have lower access frequency. This eviction policy strikes a balance between maximizing memory availability and minimizing potential privacy violations.

Together, these optimizations enable efficient compression and controlled eviction of private KV-cache entries, aligning memory usage with the dynamic requirements of multi-tenant LLM inference systems.

VII. Evaluation

In this section, we comprehensively evaluate SafeKV across a diverse set of state-of-the-art LLMs, including Phi-4 [13], Qwen3-30B-A3B [20], Qwen3-32B [21], Qwen3-235B-A22B [19], Llama-3.3-70B [12] and DeepSeek-R1 [17]. Our evaluation aims to answer the following research questions:

- **[RQ1] Effectiveness**: How reliably does SafeKV mitigate timing side-channel attacks and prevent leakage of sensitive content?
- **[RQ2] Cost**: What are the system costs, including model deployment overhead and latency introduced by privacy detection?
- **[RQ3] Performance**: Compared to full isolation-based cache management, what performance advantages does SafeKV offer under realistic workloads?

Experimental Setup. At the time of this writing, although vLLM [16] has introduced KV-cache sharing, its implementation remains preliminary. In contrast, SGLang [15] offers a fully featured stack with support for KV-cache sharing, fine-grained eviction, and batching strategies. As such, we implemented SafeKV within the SGLang framework and conducted all experiments on a server equipped with 8× NVIDIA H20 96GB GPUs. For privacy-related evaluation, we adopt the `pii-masking` dataset from ai4privacy [4], which includes multilingual samples covering 54 categories of personally identifiable information (PII).

Attacker Model. Our threat model focuses on timing-based side-channel attacks in multi-tenant LLM serving. We do not model the internal candidate generation strategies of attackers. Instead, we assume the adversary has prior knowledge of the non-sensitive prefix portion of a target query and seeks to recover the sensitive suffix tokens. Candidate tokens are drawn from the same benchmark dataset. The attacker infers the correctness of each guess based on the observed Time-to-First-Token (TTFT) latency differences. The effectiveness of SafeKV is measured by the reduction in attacker success rate when attempting to reconstruct privacy-sensitive tokens.

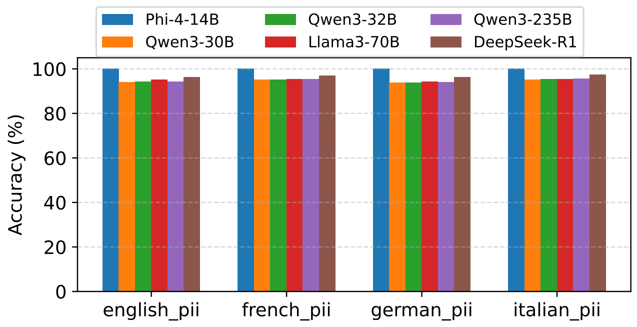


Fig. 6: Accuracy of Protecting Private Information from Timing Side-Channel Attacks

A. *RQ1 Effectiveness Evaluation*

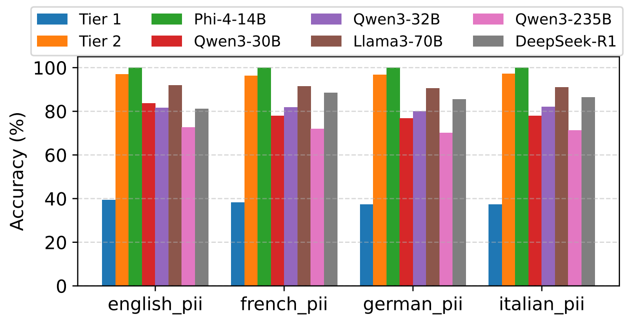
*a) **Overall Defense Effectiveness.**:* We first evaluate SafeKV 's ability to defend against timing-based side-channel attacks. In our experimental setup, both benign users and adversaries access deployed LLM models (e.g., Qwen3-235B-A22B, DeepSeek-R1) via OpenAI-compatible APIs. Adversaries attempt to infer privacy-sensitive tokens from victim prompts by measuring Time-to-First-Token (TTFT) latency, as described in our attacker model.

Figure 6 presents the defense success rates of SafeKV across different model backbones. Across all evaluated models, SafeKV consistently achieves high effectiveness, exceeding 94% in blocking timing-based inferences. Notably, the defense accuracy improves with more powerful in-servce LLMs. For example, under DeepSeek-R1, SafeKV achieves defense rates of 96.30%, 96.90%, 96.24%, and 97.26% across privacy-sensitive datasets in four languages, demonstrating strong robustness and generalization.

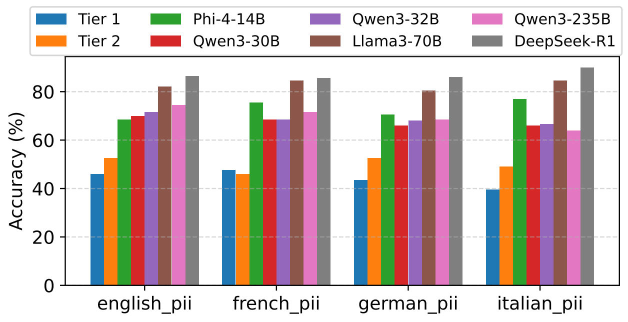
Additionally, the Microsoft Phi-4 deployment achieve near-complete coverage against adversarial probes. We attribute this to Phi-4's rigorous safety-oriented post-training, which includes supervised fine-tuning on a wide range of security-sensitive categories. These findings highlight a promising trend: as foundation models grow in capability, SafeKV 's defense effectiveness also improves. This indicates that our approach is future-proof and benefits from underlying model advancements. .

*b) **Accuracy of Multi-Tier Detection.**:* We now evaluate the accuracy and effectiveness of SafeKV 's hybrid privacy detection pipeline, which operates in three stages: Tier-1 rule-based matching, Tier-2 lightweight general privacy detector, and Tier-3 context-aware validation.

Tier-1: Rule-Based Pattern Matching. As outlined in Section V-B, the first stage performs fast keyword and



(a) Accuracy



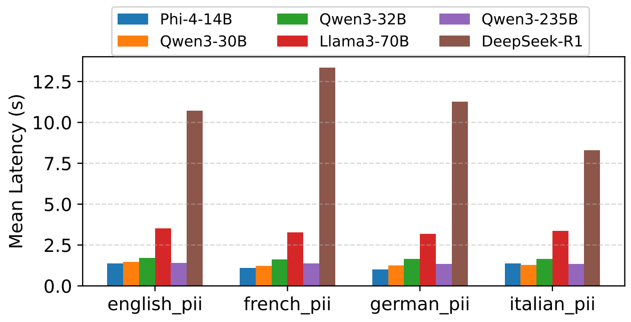
(b) Accuracy with Complex Prompts

Fig. 7: Comparison of the Accuracy of Multi-Tier Privacy Detectors under simple/complex requests

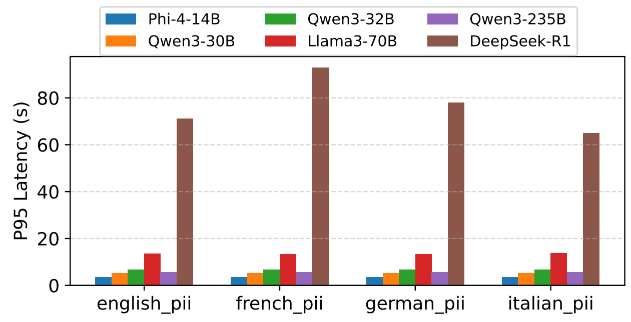
regular expression matching to capture structured and explicit sensitive content. By default, SafeKV includes a comprehensive rule set covering the PII types listed in Table IV, which can also be extended via a configurable file (`privacy_pattern_config.json`) or API interface. On multilingual test sets, the Tier-1 engine achieves detection accuracies of 39.24%, 38.09%, 37.35%, and 37.33% respectively (as shown in Figure 7(a)). While lightweight and highly efficient, this tier lacks semantic understanding and demonstrates limited accuracy, particularly for obfuscated or implicitly expressed privacy content.

Tier-2: General Privacy Detector. To compensate for Tier-1's limitations, employs a compact transformer-based language model to detect privacy-sensitive content that escapes pattern matching. Based on our benchmarking in Appendix A, we adopt **Llama-3.2-1B** as the default Tier-2 detector, offering a favorable trade-off between detection accuracy, inference latency, and GPU resource usage. As shown in Figure 7(a), Llama-3.2-1B achieves strong and consistent accuracy across all tested languages, reaching 96.85%, 96.30%, 96.64%, and 97.15% respectively.

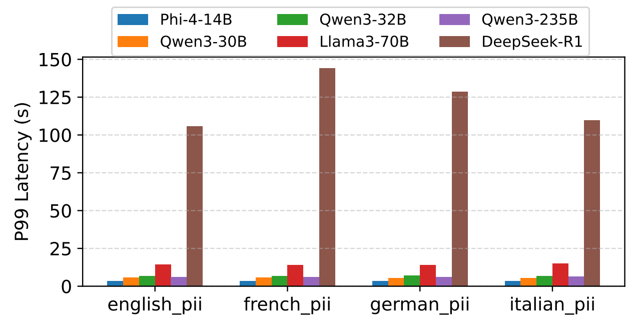
Tier-3: Context-Aware Validation. Despite the effectiveness of Tiers 1 and 2, certain complex inputs remain challenging, particularly those where sensitive information is embedded within long-range context or multi-turn conversations. In such cases, the limited capacity of compact models like Llama-3.2-1B leads to degraded detection quality. As illustrated in Figure 7(b), its accuracy falls to around 50% when processing privacy cues that rely on broader conversational context. To address this, Tier-3 invokes a more capable model such as



(a) Mean Latency



(b) P95 Latency



(c) P99 Latency

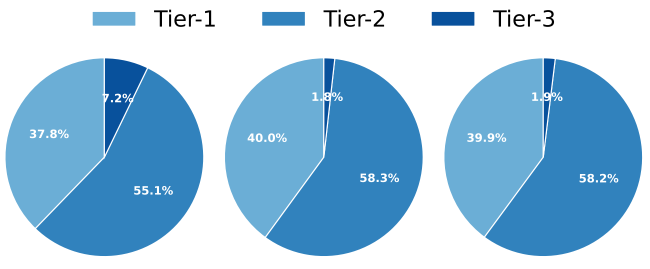
Fig. 8: The Latency of PII detection of Large Scale LLMs

DeepSeek-R1, which achieves over 90% accuracy under the same conditions. This significant improvement underscores the importance of Tier-3 in handling nuanced, context-dependent privacy risks that lightweight detectors may miss.

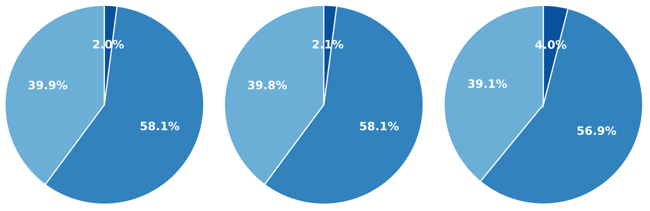
B. *RQ2 Cost Evaluation*

*a) **Overhead of Multi-Tier Detection**:* In Section VII-A, we demonstrated the effectiveness of SafeKV in mitigating timing side-channel attacks and quantified the detection accuracy of each tier individually. Here, we further evaluate the overall runtime overhead of the complete multi-tier privacy detection pipeline across different LLM backends. Specifically, Figure 8 presents the average, P95, and P99 latency of SafeKV when deployed using various foundation models. Given that Tier-3 performs context-aware validation using the underlying LLM itself, its latency is inherently dependent on the model size and inference complexity.

Although the mean latency depicted in Figure 8 may initially appear high, it predominantly results from Tier-



(a) Phi-4-14B (b) Qwen3-30B-A3B (c) Qwen3-32B



(d) LLama-3.3-70B (e) Qwen3-235B-A22B (f) DeepSeek-R1

Fig. 9: Workload Ratio of Multi-Tier Detectors under different Large Scale LLMs.

3's computationally intensive validation (further analyzed in Section VII-B0b). Importantly, this overhead is significantly mitigated in practice. First, as illustrated in Figure 9, more than 92% of requests are resolved by the lightweight and low-latency Tier-1 and Tier-2 stages, ensuring that only a small subset of queries proceed to the more expensive Tier-3. Second, SafeKV employs an asynchronous detection pipeline that effectively decouples privacy classification from the critical inference path. Consequently, privacy checks do not block token generation, minimizing any impact on latency-sensitive serving. Overall, by integrating multi-tier detection with asynchronous processing, SafeKV delivers robust privacy enforcement with minimal overhead. A comprehensive analysis of system performance improvements is detailed in Section VII-C.

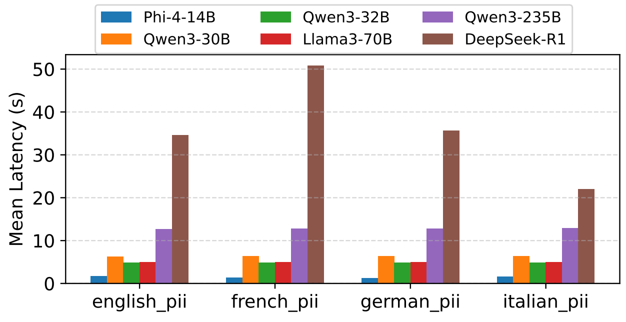
*b) **Per-Tier Detection Cost.**:* We next quantify the detection overhead separately for each tier. As shown in Table VI, the rule-based detection in Tier-1 completes within 0.2 ms per prompt, while Tier-2, which utilizes the compact Llama-3.2-1B model, incurs an average latency below 125 ms. For Tier-3, Figure 10 illustrates how detection latency scales with different base model sizes, clearly indicating a latency increase correlated with larger model parameters and associated inference complexity.

Additionally, model latency is influenced not only by model size but also by the prompt length and contextual complexity. Due to space limitations, detailed data and analysis on the impact of prompt length can be found in Appendix B.

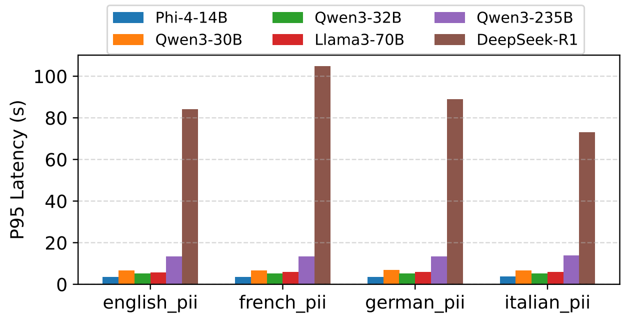
C. RQ3 Performance Evaluation

In this section, we evaluate the performance of SafeKV by comparing it against three representative KV-cache management strategies:

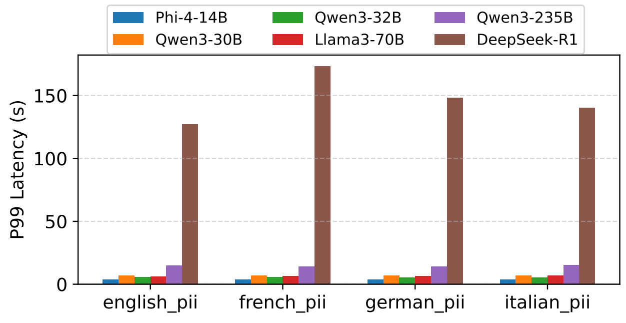
- **SGLang [15]:** Full global cache sharing with no privacy protection, achieving maximum efficiency.



(a) Mean Latency



(b) P95 Latency



(c) P99 Latency

Fig. 10: The Latency breakdown of Tier-3 PII detection across four languages.

- **Cache Partitioning [49]:** Strict per-user isolation using separate radix-trees, ensuring strong privacy but no cross-user reuse.
- **Public System Prompt:** A hybrid that allows reuse of shared system prompts atop isolated user data.

We evaluate these methods across three representative workload types:

- **Single Request PII:** Short privacy-sensitive queries.
- **Multi-Turn Chat:** Conversational sessions with embedded PII.
- **System Prompt:** Requests with shared system prompts and user-specific PII.

*a) **LLM Inference Latency**:* Our primary evaluation metric is *inference latency*, particularly focusing on *Time-To-First-Token (TTFT)*, which is a key indicator of responsiveness in LLM serving systems. Given that KV-cache reuse primarily accelerates the *prefill* stage, TTFT effectively captures performance differences among cache management strategies.

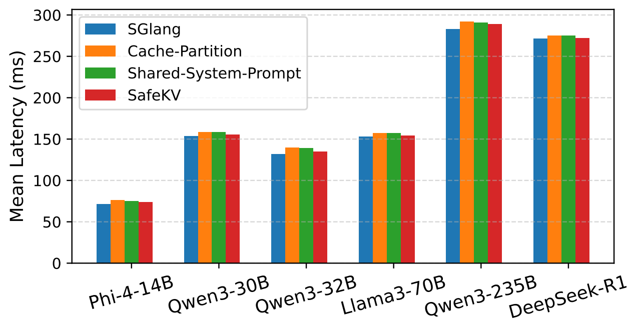
TABLE VI: Latency breakdown of Tier-1 and Tier-2 across four languages.

	english_pii			french_pii			german_pii			italian_pii		
Tier	Mean(ms)	P95(ms)	P99(ms)	Mean(ms)	P95(ms)	P99(ms)	Mean(ms)	P95(ms)	P99(ms)	Mean(ms)	P95(ms)	P99(ms)
Tier-1	0.10	0.16	0.20	0.11	0.17	0.20	0.11	0.17	0.21	0.11	0.17	0.20
Tier-2	124.93	152.14	171.85	113.03	125.68	127.17	113.91	126.31	129.54	116.12	128.77	140.77

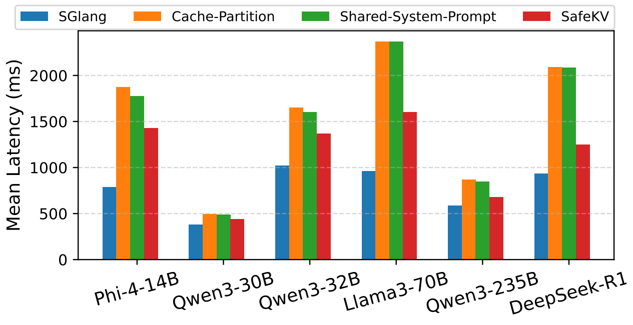
As depicted in Figure 11, all four methods exhibit comparable TTFT on single-request PII workloads, primarily due to the limited opportunity for cache reuse with short, unique queries.However, in multi-turn conversational scenarios (constructed using SharedGPT [1] dialogues augmented with pii-masking [4] queries to embed privacy-sensitive information), substantial opportunities for prefix reuse emerge, resulting in notable performance variations. Specifically, under larger models such as Qwen-235B-A22B and DeepSeek-R1, SafeKV significantly reduces the latency overhead associated with full cache isolation—from 50.41% and 118% to just 11.74% and 34.28%, respectively. Finally, in the system prompt scenario (pii-masking requests with a uniform system prompt(approximately 8192 tokens)), SafeKV continues to perform effectively, although slightly behind the Public System Prompt method explicitly optimized for this scenario. Overall, these results demonstrate that SafeKV consistently achieves superior latency performance compared to Cache Partitioning and remains adaptable across diverse real-world use cases.

*b) **LLM Inference Throughput**:* In addition to latency, throughput is critical for evaluating a system's capability to handle sustained high-volume requests. To measure this, we conduct throughput benchmarking at a fixed load of 16 requests per second (RPS), reporting token throughput as the total number of tokens divided by the end-to-end inference time. Test workloads are derived from three representative datasets: `pii_masking [4]`, `ShareGPT [1]`, and the requests with uniform system prompt.

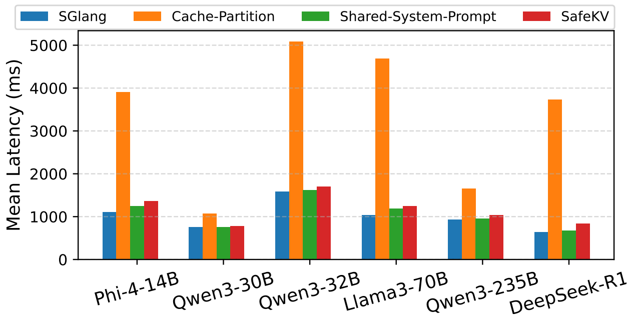
As shown in Figure 12, SafeKV substantially outperforms Cache Partitioning by selectively isolating only sensitive KV-cache entries, thus preserving reuse opportunities for non-sensitive content. This selective isolation results in throughput improvements ranging from 1.36× to 2.66× across the evaluated workloads. Performance gains are particularly pronounced when using larger foundation models such as DeepSeek-R1. These findings highlight that SafeKV effectively balances rigorous privacy enforcement with significant throughput enhancements, making it well-suited for practical, high-traffic LLM deployments.



(a) Single Request Pii



(b) Multi-Turn



(c) System Prompt

Fig. 11: Comparison of TTFT of SafeKV with SGLang, Cache-Partition, and Systme-Prompt-Sharing in different working scenarios

VIII. Discussion & Limitations

A. *Protection Scope and Assumptions*

SafeKV targets *timing-based prompt leakage* in shared KV-cache systems. It assumes an adversary with black-box knowledge of the LLM—no access to model internals, parameters, or user-level metadata beyond API-visible timing. Under this setting, SafeKV 's hybrid detection and entropy-aware runtime isolation substantially reduce the risk of cache-probing attacks.

For adversaries exploiting non-timing side channels (e.g., GPU resource contention, speculative execution, shared-memory leakage), we regard these as orthogonal and mitigate them via confidential computing (e.g., Intel TDX for CPUs and NVIDIA H100 Confidential Computing for GPUs); a full treatment is deferred to separate work. SafeKV assumes tokenizer alignment between attacker and victim; when tokenization mismatches or prompt obfuscation arise, SafeKV can incorporate input normalization (e.g., canonicalization,

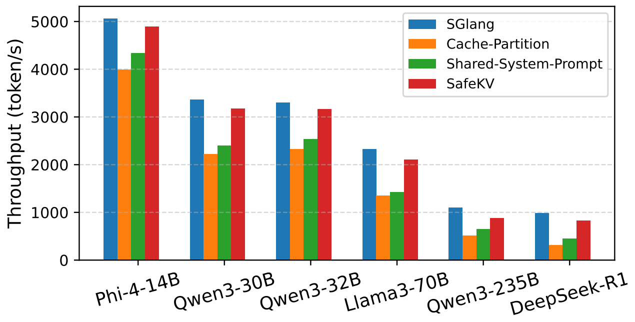


Fig. 12: Comparison of Throughput of SafeKV with SGLang, Cache-Partition, and Systme-Prompt-Sharing

Unicode folding, whitespace/punctuation regularization) to improve robustness. Alignment of tokenization schemes is orthogonal to our focus and is not discussed further.

B. *Detection Robustness and Safeguards*

Although our hybrid detector attains high empirical accuracy, novel PII formats may still induce detection errors. In such cases, false negatives (i.e., private data misclassified as public) can transiently enter the shared cache. SafeKV mitigates this via runtime entropy monitoring that flags and quarantines anomalously accessed entries. The time to detect and revoke a leaked cache block depends on traffic volume and user distribution, yielding a bounded but nonzero exposure window; in practice, this window can be further reduced by caching and batching, limiting residual risk.

C. *Deployment and Integration Considerations*

SafeKV introduces predictable memory and compute requirements due to multi-tier privacy detection and private-block isolation. Detection is decoupled from the critical inference path and supports asynchronous classification; operators should provision capacity to sustain target throughput (Tier-2 detector: Llama-3.2-1B, $\sim$2.6 GB GPU HBM). Tier-3 verification leverages the running LLM and adds a small amount of inference work. In cost-focused environments, lighter detectors or rule-only configurations can be selected, with a corresponding trade-off in protection strength.

SafeKV integrates with existing LLM inference stacks (e.g., vLLM [41], SGLang [71]) by extending their cache subsystems with private/public block indexing, per-user origin tags, and entropy-based access tracking. These extensions are modular, preserve existing scheduling/batching and API surfaces, and can be reused across deployments with minimal disruption to serving logic.

D. *Generality and Transferability*

Our detection pipeline targets privacy-sensitive inputs in natural-language prompts. Extending SafeKV to multimodal LLMs (e.g., image+text) may require modality-aware detection strategies; however, the unified cache-management layer is KV-cache–centric and model-agnostic, and thus remains directly compatible with multimodal serving stacks.

IX. Related Works

A. *Multi-tenant Security*

Side-channel attacks have long threatened multi-tenant systems due to shared resource usage. It can be broadly categorized into classical system-level attacks and emerging model-level side channels.

Classical side-channel attacks exploit shared hardware or OS abstractions in multi-tenant systems. Cross-VM attacks recover sensitive data between co-located virtual machines by monitoring CPU caches or memory access patterns [56], [61], [68], [69]. Other works leverage shared OS resources, such as OS data structure or public file systems, to launch cross-application attacks in Unix, Android, or iOS environments [28], [36], [58], [64], [66], [67]. Recent studies further reveal that multiple WebAssembly modules isolated in the same runtime are vulnerable to cross-module attacks [39], [46].

Complementing these, recent work highlights a new class of attacks targeting LLM inference backends. These attacks exploit timing differences caused by shared KV-cache reuse to infer user inputs [54], [60], [73]. For example, PromptPeek [54] and InputSnatch [73] reconstruct user prompts via TTFT measurements in black-box settings. Unlike classical channels, these attacks are unique to LLM-serving pipelines, where performance optimizations inadvertently expose privacy risks. Our work builds on these findings by proposing a practical, multi-tier defense system that addresses this emerging attack surface.

B. *Defenses Against KV-Cache Side Channels*

To mitigate KV-cache side-channel attacks, researchers and practitioners have proposed a range of defenses.

The most straightforward solution is **User-level cache isolation**, it prevents cross-user sharing by allocating distinct cache namespaces per user. This containment strategy eliminates cache-based interactions between users and is adopted by some LLM providers(OpenAI [48] and DeepSeek [31]) and researchers [49] for prefix caching. While effective, such strict isolation sacrifices memory efficiency and undermines the performance benefits of cache reuse.

Rate Limiting serves as a complementary defense by throttling the frequency of user queries, thereby impeding rapid probing required for cache-timing attacks. For instance, OpenAI enforces rate limits to prevent abuse and ensure infrastructure stability [47]. However, rate limiting must be carefully tuned to avoid degrading service quality for benign users.

A third line of defense is **Timing Obfuscation**, which aims to eliminate observable latency differences between cache hits and misses. Prior work in model extraction has shown that response time can correlate with internal architecture details [26], [32], [33], motivating similar countermeasures in LLM serving. Two common strategies are: (i) enforcing constant-time execution [45] or injecting random delays to flatten latency variance [27], and (ii) disabling token-level streaming, which removes fine-grained timing signals from observable outputs. While these approaches can mask timing patterns, they often incur latency penalties or reduce the interactivity of real-time systems.

X. Conclusion

This paper presents SafeKV , a privacy-preserving KV-cache management framework for LLM serving systems, designed to mitigate timing side-channel attacks arising from shared cache entries. SafeKV combines a hybrid privacy detection pipeline, comprising rule-based matching, lightweight LLM detectors, and context-aware validation, with a sensitivity-aware KV-cache manager that supports efficient reuse through a unified radix tree index.

Our multi-tiered detection achieves accurate privacy classification with minimal latency, while the cache manager enables fine-grained isolation without compromising performance. Extensive evaluations show that SafeKV mitigates over 94% of timing-based attacks across multiple LLM backbones and improves throughput by up to 2.66× compared to per-user cache isolation.These results demonstrate that selective KV-cache sharing, guided by efficient privacy detection, provides a practical balance between security and scalability.

References

[1] "anon8231489123/sharegpt_vicuna_unfiltered," https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2023.

[2] "Bellegroup/multiturn_chat_0.8m," https://huggingface.co/datasets/BelleGroup/multiturn_chat_0.8M, 2023.

[3] "ab-ai/pii_model," https://huggingface.co/ab-ai/pii_model, 2024.

[4] "ai4privacy/pii-masking-200k," https://huggingface.co/datasets/ai4privacy/pii-masking-200k, 2024.

[5] "beki/flair-pii-distilbert," https://huggingface.co/beki/flair-pii-distilbert, 2024.

[6] "iiiorg/piiranha-v1-detect-personal-information," https://huggingface.co/iiiorg/piiranha-v1-detect-personal-information, 2024.

[7] "meta-llama/llama-2-13b-hf," https://huggingface.co/meta-llama/Llama-2-13b-hf, 2024.

[8] "meta-llama/llama-2-70b-hf," https://huggingface.co/meta-llama/Llama-2-70b-hf, 2024.

[9] "meta-llama/llama-3.1-8b," https://huggingface.co/meta-llama/Llama-3.1-8B, 2024.

[10] "meta-llama/llama-3.2-1b," https://huggingface.co/meta-llama/Llama-3.2-1B, 2024.

[11] "meta-llama/llama-3.2-3b," https://huggingface.co/meta-llama/Llama-3.2-3B, 2024.

[12] "meta-llama/llama-3.3-70b-instruct," https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct, 2024.

[13] "microsoft/phi-4," https://huggingface.co/microsoft/phi-4, 2024.

[14] "omshikhare/dbert-pii-detection-model," https://huggingface.co/omshikhare/dbert-pii-detection-model, 2024.

[15] "The sglang source code," https://github.com/sgl-project/sglang, 2024.

[16] "vllm, easy, fast, and cheap llm serving for everyone," https://github.com/vllm-project/vllm?tab=readme-ov-file, 2024.

[17] "deepseek-ai/deepseek-r1-0528," https://huggingface.co/deepseek-ai/DeepSeek-R1-0528, 2025.

[18] "Qwen/qwen3-0.6b," https://huggingface.co/Qwen/Qwen3-0.6B, 2025.

[19] "Qwen/qwen3-235b-a22b," https://huggingface.co/Qwen/Qwen3-235B-A22B, 2025.

[20] "Qwen/qwen3-30b-a3b," https://huggingface.co/Qwen/Qwen3-30B-A3B, 2025.

[21] "Qwen/qwen3-32b," https://huggingface.co/Qwen/Qwen3-32B, 2025.

[22] "Qwen/qwen3-4b," https://huggingface.co/Qwen/Qwen3-4B, 2025.

[23] "Qwen/qwen3-8b," https://huggingface.co/Qwen/Qwen3-8B, 2025.

[24] Anthropic, "Prompt caching with claude." 2024, accessed: 2025-07-17. [Online]. Available: https://www.anthropic.com/news/prompt-caching

[25] F. Bang, "Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings," in *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, 2023, pp. 212–218.

[26] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 515–532.

[27] J. Breier, D. Jap, X. Hou, and S. Bhasin, "A desynchronization-based countermeasure against side-channel analysis of neural networks," in *International Symposium on Cyber Security, Cryptology, and Machine Learning*. Springer, 2023, pp. 296–306.

[28] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it:{UI} state inference and novel android attacks," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[29] K. Chu, Z. Shen, D. Xiang, and W. Zhang, "SafeKV: Safe KV-cache sharing in LLM serving," in *Machine Learning for Computer Architecture and Systems 2025*, 2025. [Online]. Available: https://openreview.net/forum?id=jhDsbd5eXL

[30] deepmind, "Gomini," 2025, https://deepmind.google/technologies/gemini/.

[31] DeepSeek, "Deepseek api docs: Deepseek api introduces context caching on disk, cutting prices by an order of magnitude." 2024, accessed: 2025-07-17. [Online]. Available: https://api-docs.deepseek.com/news/news0802/

[32] G. Dong, P. Wang, P. Chen, R. Gu, and H. Hu, "Floating-point multiplication timing attack on deep neural network," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2019, pp. 155–161.

[33] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[34] V. Gallego, "Configurable safety tuning of language models with synthetic preference data," 2024.

[35] N. Ho, S. Bae, T. Kim, H. Jo, Y. Kim, T. Schuster, A. Fisch, J. Thorne, and S.-Y. Yun, "Block transformer: Global-to-local language modeling for fast inference," *Advances in Neural Information Processing Systems*, vol. 37, pp. 48 740–48 783, 2024.

[36] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012.

[37] N. Kandpal, B. Lester, C. Raffel, S. Majstorovic, S. Biderman, B. Abbasi, L. Soldaini, E. Shippole, A. F. Cooper, A. Skowron, J. Kirchenbauer, S. Longpre, L. Sutawika, A. Albalak, Z. Xu, G. Penedo, L. B. Allal, E. Bakouch, J. D. Pressman, H. Fan, D. Stander, G. Song, A. Gokaslan, T. Goldstein, B. R. Bartoldson, B. Kailkhura, and T. Murray, "The common pile v0.1: An 8tb dataset of public domain and openly licensed text," 2025. [Online]. Available: https://arxiv.org/abs/2506.05209

[38] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[39] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[40] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[41] ——, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

[42] H. Li, Y. Li, A. Tian, T. Tang, Z. Xu, X. Chen, N. Hu, W. Dong, Q. Li, and L. Chen, "A survey on large language model acceleration based on kv cache management," *arXiv preprint arXiv:2412.19442*, 2024.

[43] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.

[44] J. Lucas and R. Harang, "Structuring applications to secure the kv cache," https://developer.nvidia.com/blog/structuring-applications-to-secure-the-kv-cache/, Apr. 2025, accessed: 2025-05-01.

[45] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky nets: Recovering embedded neural network models and inputs through simple power and

timing side-channels—attacks and defenses," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 079–12 092, 2021.

[46] S. Narayan, C. Disselkoen, D. Moghimi, S. Cauligi, E. Johnson, Z. Gang, A. Vahldiek-Oberwagner, R. Sahita, H. Shacham, D. Tullsen *et al.*, "Swivel: Hardening {WebAssembly} against spectre," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[47] OpenAI, "Openai developer platform, rate limits." 2024, accessed: 2025-07-17. [Online]. Available: https://platform.openai.com/docs/guides/rate-limits/what-are-the-rate-limits-for-our-api

[48] ——, "Prompt caching: Reduce latency and cost with prompt caching." 2024, accessed: 2025-07-17. [Online]. Available: https://platform.openai.com/docs/guides/prompt-caching

[49] Z. Pang, W. Wang, and Y. Liao, "Cache partitioning for mitigating timing side-channel attacks in llm serving systems," in *2024 6th International Conference on Frontier Technologies of Information and Computer (ICFTIC)*. IEEE, 2024, pp. 1238–1245.

[50] R. Qin, Z. Li, W. He, J. Cui, F. Ren, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot," in *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, 2025, pp. 155–170.

[51] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv e-prints*, 2019.

[52] Y. Shen, Z. Ji, J. Lin, and K. Koedginer, "Enhancing the de-identification of personally identifiable information in educational data," *arXiv preprint arXiv:2501.09765*, 2025.

[53] M. Soleimani, G. Jia, I. Gim, S.-s. Lee, and A. Khandelwal, "Wiretapping llms: Network side-channel attacks on interactive llm services," *Cryptology ePrint Archive*, 2025.

[54] L. Song, Z. Pang, W. Wang, Z. Wang, X. Wang, H. Chen, W. Song, Y. Jin, D. Meng, and R. Hou, "The early bird catches the leak: Unveiling timing side channels in llm serving systems," *arXiv preprint arXiv:2409.20002*, 2024.

[55] X. Sun, G. Liu, Z. He, H. Li, and X. Li, "Deprompt: Desensitization and evaluation of personal identifiable information in large language model prompts," *arXiv preprint arXiv:2408.08930*, 2024.

[56] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift, "A placement vulnerability study in {Multi-Tenant} public clouds," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[57] S. Wang, Y. Zhao, Z. Liu, Q. Zou, and H. Wang, "Sok: Understanding vulnerabilities in the large language model supply chain," *arXiv preprint arXiv:2502.12497*, 2025.

[58] Z. Wang, J. Guan, X. Wang, W. Wang, L. Xing, and F. Alharbi, "The danger of minimum exposures: Understanding cross-app information leaks on ios through multi-side-channel learning," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[59] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[60] G. Wu, Z. Zhang, Y. Zhang, W. Wang, J. Niu, Y. Wu, and Y. Zhang, "I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving," in *Proceedings of the 2025 Network and Distributed System Security (NDSS) Symposium*, San Diego, CA, USA, 2025.

[61] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops:{Cross-VM} row hammer attacks and privilege escalation," in *25th USENIX security symposium (USENIX Security 16)*, 2016.

[62] L. Ye, Z. Tao, Y. Huang, and Y. Li, "Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition," *arXiv preprint arXiv:2402.15220*, 2024.

[63] W. Zeng, Y. Dong, J. Zhou, J. Ma, J. Tan, R. Wang, and M. Li, "MPCache: MPC-friendly KV cache eviction for efficient private LLM inference," 2025. [Online]. Available: https://openreview.net/forum?id=QliOktBcy3

[64] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems." in *USENIX Security Symposium*, 2009.

[65] T. Zhang, G. Saileshwar, and D. Lie, "Time will tell: Timing side channels via output token count in large language models," *arXiv preprint arXiv:2412.15431*, 2024.

[66] X. Zhang, X. Wang, X. Bai, Y. Zhang, and X. Wang, "Os-level side channels without procfs: Exploring cross-app information leakage on ios," in *Proceedings of the Symposium on Network and Distributed System Security*, 2018.

[67] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on arm and their implications for android devices," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[68] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[69] ——, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[70] J. Zhao, Z. Fang, S. Li, S. Yang, and S. He, "Buzz: Beehive-structured sparse kv cache with segmented heavy hitters for efficient llm inference," *arXiv preprint arXiv:2410.23079*, 2024.

[71] L. Zheng, L. Yin, Z. Xie, J. Huang, C. Sun, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Efficiently programming large language models using sglang," *arXiv preprint arXiv:2312.07104*, 2023.

[72] L. Zheng, L. Yin, Z. Xie, C. L. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Sglang: Efficient execution of structured language model programs," *Advances in Neural Information Processing Systems*, vol. 37, pp. 62 557–62 583, 2024.

[73] X. Zheng, H. Han, S. Shi, Q. Fang, Z. Du, X. Hu, and Q. Guo, "Inputsnatch: Stealing input in llm services via timing side-channel attacks," *arXiv preprint arXiv:2411.18191*, 2024.

[74] Z. Zheng, X. Ji, T. Fang, F. Zhou, C. Liu, and G. Peng, "Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching," *arXiv preprint arXiv:2412.03594*, 2024.

Appendix A
Tier-2 Detection Accuracy and Efficiency.

As discussed in Section V-B, Tier-2 serves as the primary detection path for most incoming requests, complementing Tier-1 by providing more accurate privacy classification with minimal latency and resource overhead. To meet this goal, we evaluate six lightweight LLMs from the Qwen3 and Llama3 families as Tier-2 candidates, seeking a favorable trade-off between detection accuracy and runtime efficiency.

We first analyze their accuracy under varying output lengths. As shown in Figure 13, Llama3 models maintain high detection accuracy even with short outputs ($max\_output\_length = 10$), while Qwen3 models require significantly longer outputs (≥ 75) to match this performance, likely due to their reliance on intermediate reasoning.

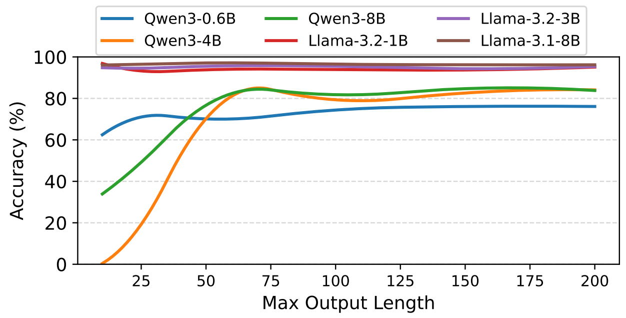


Fig. 13: Accuracy vs Max Output Length on english-pii-43k.

We then benchmark all six models on multilingual privacy datasets, using the best-performing output lengths per family (Qwen3: `75`; Llama3: `10`). As shown in Figure 14, **Llama-3.2-1B** consistently achieves the highest accuracy across all tasks, reaching 96.85%, 96.30%, 96.64%, and 97.15%. Moreover, as shown in Figure 15, its shorter output requirement also results in significantly lower average, P95, and P99 latencies compared to Qwen3 models. These results make Llama-3.2-1B a strong default choice for Tier-2 detection.

Given its strong accuracy, fast response time, and low memory footprint (**Llama-3.2-1B** only requires 2.6GB GPU memory), **Llama-3.2-1B** is selected as the default Tier-2 detector in SafeKV .

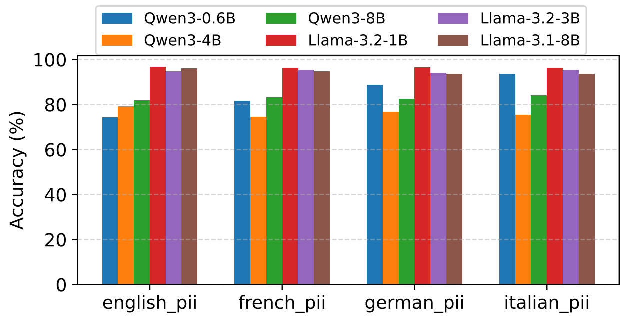


Fig. 14: Accuracy of lightweight genral LLM models for PII detection

Appendix B
PII Detection Overhead vs. Varies Prompt Length

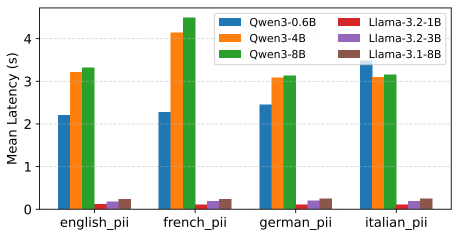
In addition to model size, LLM inference latency is heavily influenced by the length of the input prompt. To quantify this effect, we measured the privacy detection latency under varying input lengths (assuming a fixed output length of 4096 tokens and allowing the model to perform multi-step reasoning), as shown in Figure 16. The results reveal an approximately linear increase in latency with respect to prompt length, which aligns with the computational characteristics of LLM serving. Specifically, longer prompts introduce additional overhead in the prefill phase due to increased token processing, and also result in more extensive KV-cache computations during the decoding phase.

We further observe that large models, such as DeepSeek-R1, experience substantial latency increases under long input prompts. On our testbed, privacy detection times for input lengths of 512, 1K, 2K, and 4K tokens reached 49.1s, 59.2s, 61.6s, and 70.4s, respectively—ultimately surpassing the system's timeout threshold at the higher end. Consequently, latency measurements for DeepSeek-R1 with long prompts are omitted from Figure 16. These findings underscore the importance of restricting Tier-3 detection to only high-risk or uncertain cases, thereby maintaining system responsiveness.

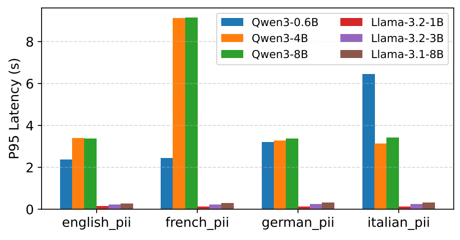
Appendix C
P95/P99 Latency of SafeKV Performance.

As presented in Section § VII-C, SafeKV notably reduces average time-to-first-token (TTFT) compared to the Cache-Partition baseline, particularly in the Multi-Turn Chat and System Prompt scenarios. To further assess tail latency, we report the P95 and P99 TTFT across different models and scenarios in Figure 17. The results reveal three key observations:

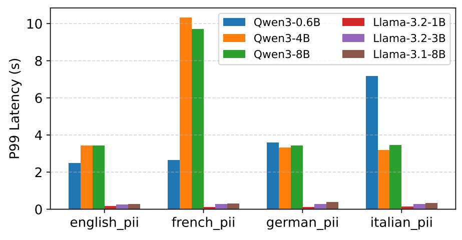
- (1) Across all evaluated settings, SafeKV consistently achieves lower P95 and P99 TTFT than Cache-Partition, indicating its effectiveness in mitigating long-tail latency.
- (2) Although the Shared System Prompt approach reaches performance close to SGLang in the System Prompt scenario, it does not yield tail latency improvements in the other two workloads.
- (3) In the System Prompt setting, while SafeKV delivers significant improvements in mean TTFT, its asynchronous detection mechanism—where KV-cache entries are initially isolated and later reclassified—introduces additional delay for some requests, resulting in slightly elevated P95 and P99 TTFT in these cases.



(a) Mean Latency

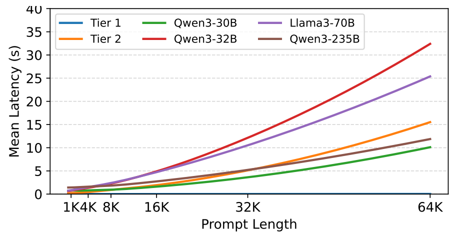


(b) P95 Latency

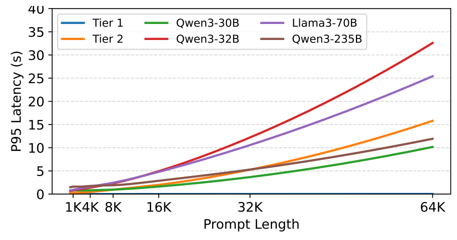


(c) P99 Latency

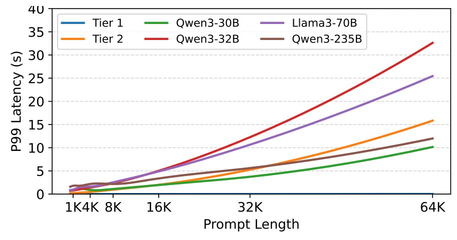
Fig. 15: Latency of lightweight genral LLM models for PII detection



(a) Mean Latency

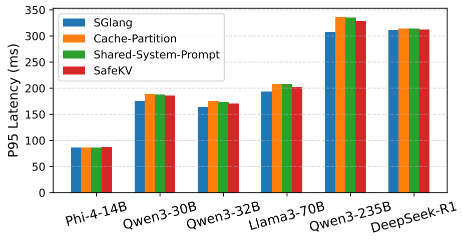


(b) P95 Latency

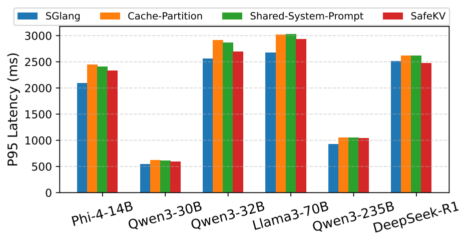


(c) P99 Latency

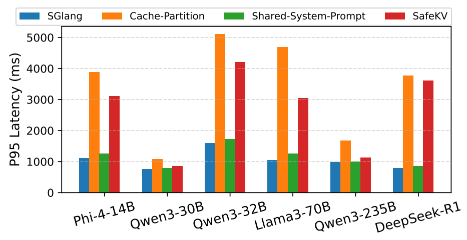
Fig. 16: The Latency of PII detection vs different prompt length



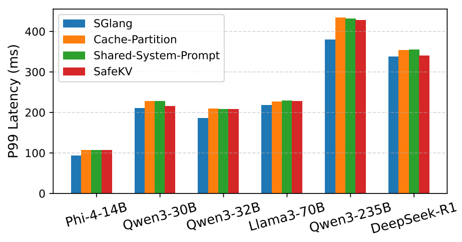
(a) Single Request PII P95 Latency



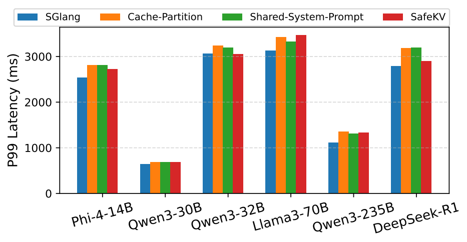
(b) Multi-Turn Chat P95 Latency



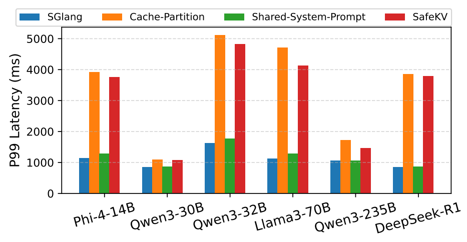
(c) System Prompt P95 Latency



(d) Single Request PII P99 Latency



(e) Multi-Turn Chat P99 Latency



(f) SYstem Prompt P99 Latency

Fig. 17: Comparison of TTFT (P95 & P99) of SafeKV with SGLang, Cache-Partition, and Systme-Prompt-Sharing in different working scenarios