SCX: Stateless KV-Cache Encoding for Cloud-Scale Confidential Transformer Serving

Mu Yuan
The Chinese University of Hong Kong
Hong Kong SAR, China
muyuan@cuhk.edu.hk

Lan Zhang*
University of Science and Technology of China
Hefei, China
zhanglan@ustc.edu.cn

Liekang Zeng, Siyang Jiang
Bufang Yang, Di Duan
The Chinese University of Hong Kong
Hong Kong SAR, China

Guoliang Xing*
The Chinese University of Hong Kong
Hong Kong SAR, China
glxing@ie.cuhk.edu.hk

ABSTRACT

Transformer models have revolutionized fields like natural language processing and computer vision but face privacy concerns in sensitive applications such as medical diagnostics. Existing confidential serving methods, including cryptography-based, memory isolation-based, and access control-based, offer trade-offs between privacy and efficiency but often struggle with high latency or hardware dependencies. This work proposes stateless KV-cache encoding (SCX), a novel framework that encodes the intermediate key-value cache during Transformer inference using user-controlled keys. SCX ensures that the cloud can neither recover the input nor independently complete the next token prediction, effectively preserving privacy. By introducing efficient encoding and decoding schemes, SCX addresses communication complexity and attack vulnerabilities while ensuring zero loss of inference quality. Experiments on large Transformer models demonstrate that SCX achieves lower latency (e.g., 36ms for LLaMA-7B), outperforming state-of-the-art cryptography and memory isolation methods by orders of magnitude. Moreover, SCX can complementarily work with advanced KV-cache management techniques to further enhance KV-cache communication efficiency by 85%, marking a significant step toward practical, privacy-preserving large Transformer serving.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Security and privacy** → **Distributed systems security**; • **Computing methodologies** → **Distributed artificial intelligence**.

KEYWORDS

Model Inference, Large Language Model, KV-Cache, Confidential Computing, Device-Cloud Collaboration

*Guoliang Xing and Lan Zhang are corresponding authors.

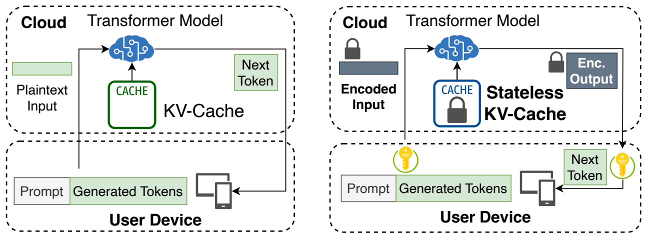
ACM Reference Format:
Mu Yuan, Lan Zhang, Liekang Zeng, Siyang Jiang, Bufang Yang, Di Duan, and Guoliang Xing. 2025. SCX: Stateless KV-Cache Encoding for Cloud-Scale Confidential Transformer Serving. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3718958.3750509

1 INTRODUCTION

Transformer models [4, 38] have emerged as a cornerstone of modern artificial intelligence, revolutionizing numerous fields such as language modeling [17, 37], computer vision [23], time-series forecasting [3], and molecular biology [34].

While the transformative potential of these models has been widely recognized, their current plaintext-based services [30] raise data privacy concerns in sensitive applications, ranging from medical diagnostics [36] to financial forecasting [40]. Taking large language models (LLMs) as an example, Open Worldwide Application Security Project [33] has identified sensitive information disclosure as the second most critical risk of LLMs in 2025.

Towards confidential Transformer serving, existing work has mainly explored three technical directions:
(1) Cryptography-based. These approaches leverage advanced cryptographic techniques, including secure multi-party computation [8], homomorphic encryption [31], and secret sharing-based methods [14], to enable secure inference without exposing sensitive data or model parameters. The primary advantage of cryptography-based methods is their rigorous security guarantees, as they rely on well-established mathematical principles to ensure confidentiality. However, these methods often suffer from significant computational and communication overheads (e.g., 200s latency and 1.8GB traffic for Llama-7b model, see Tab. 6), as well as potential inference quality loss due to the approximation of non-linear layers. These constraints make cryptography-based methods challenging to deploy at scale.
(2) CPU memory isolation-based. These methods rely on trusted execution environments (TEE) [16] such as Intel SGX [42] and TDX [15] to provide on-cloud secure enclaves for running computations. TEE creates isolated regions of memory that are inaccessible to external processes, thereby protecting sensitive data and models during inference. However, the reliance on CPU-based enclaves results in high latency compared to GPU-accelerated inference,



(a) Baseline: plaintext serving of Transformer model with valid KV-cache.

(b) SCX: confidential serving of Transformer model with encoded stateless KV-cache.

Figure 1: SCX protects user data with stateless KV-cache encoding using user-controlled keys.

particularly for large Transformer models [26]. For example, experimental results show that a full-TEE deployment of the Llama-7b model has a throughput of only 0.3 tokens per second (Fig. 10).

(3) Access control-based. These approaches leverage specialized hardware and software stacks to enforce strict access control policies during model serving. For instance, Apple Private Cloud Compute (PCC) [1] and Nvidia Confidential Computing (CC) [28] allow computations to be performed securely within hardware-isolated environments. Compared to full cryptographic approaches, access control-based solutions typically offer lower performance overhead in single-device or compute-bound settings. However, they can incur non-negligible latency in distributed inference scenarios due to the need for encrypting inter-node communication. Additionally, they require specialized hardware (e.g., NVIDIA H100 GPUs) and tightly integrated software stacks, which may not be readily available or compatible with all cloud environments.

New idea: stateless KV-cache encoding. This work aims to achieve efficient inference (e.g., <50ms latency for 7B-level models) while ensuring privacy preservation in a hardware-agnostic manner. Our core idea is inspired by recent advancements in distributed KV-cache management [21, 24]. KV-cache [22] refers to the key-value pairs stored during Transformer inference, representing intermediate attention states. In distributed KV-cache management, long-context generation cannot be completed on a single device due to dependencies on KV-cache stored across multiple devices. To address this, these technologies compress the KV-cache to improve communication efficiency, thereby reducing inference latency. Building upon this concept, we explore whether user data privacy can be protected by encoding the KV-cache using user-controlled keys. We term this approach *stateless KV-cache encoding (SCX)*. Here, the "stateless" refers to the inability of the on-cloud KV-cache (attention state) to independently complete the inference. As illustrated in Fig. 1, SCX encodes input data with a user-controlled key on the device. This operation effectively "locks" the KV-cache computed on the cloud. Tab. 1 compares SCX and existing confidential serving methods. Developing an effective framework for stateless KV-cache encoding involves two key challenges:

Challenge-1: communication complexity. The KV-cache resides in the attention module of every Transformer block [38]. A naive approach that requires interaction with the user for every KV-cache access during inference would result in a communication

Table 1: Comparison of SCX and complementary confidential Transformer serving methods.

Method	User Data Protection	Math Equi.	7b Gen. <50ms	Hardware Agnostic
HE+MPC	E2E Encrypt.	✗	✗	✓
Intel TDX	Mem. Isolation	✓	✗	✓
Nvidia CC Apple PCC	Access Control	✓	✓	✗
SCX	KV-Cache Enc.	✓	✓	✓

overhead of $O(T \cdot L)$, where T is the number of generated tokens and L is the number of Transformer layers. This communication cost becomes prohibitive for large models, where L typically ranges from 32 to 80 and T can easily reach up to 10K. Consequently, designing an encoding and decoding scheme that minimizes communication complexity is essential.

Challenge-2: attack vulnerability. While optimizing communication efficiency, ensuring robust privacy protection remains critical. Various attacks exist, including brute-force vocabulary matching, LLM-assisted [39] sequence recovery, and fitting-based inversion attacks [11]. Simplified encoding schemes are often vulnerable to such attacks. Our analysis and experiments reveal that existing solutions are insufficient to resist certain known attack methods, highlighting the need for a more careful design.

SCX design. Overcoming challenges posed by stateless KV-cache encoding requires multiple technical advances. First, we propose a customized block-level encoding scheme for the prefill stage of Transformer inference. This scheme incorporates three key operations: token permutation, redundant embedding, and Laplace noise injection. To demonstrate its privacy protection capability, we prove a differential privacy [10] guarantee and demonstrate its resistance to common attack methods through both theoretical analysis and experiments. Second, to address communication complexity, we introduce a "keys sharing" stage, where the user uploads encoding keys for intermediate Transformer blocks (from the 2nd to the $(L-1)$-th layer) to the cloud. This allows the cloud to recover valid KV-cache values for intermediate layers. This design reduces the communication complexity from $O(T \cdot L)$ to $O(T + L)$. Third, we design a decoding scheme tailored for the Transformer generation stage, detailing how to decode the KV-cache for the first and last Transformer blocks and how to perform next-token prediction. We theoretically prove the mathematical equivalence of the decoded output to the plaintext inference output, ensuring no degradation in inference quality.

System deployment. SCX operates as a two-party system involving the user device and the cloud server. On the user side, the encoder module introduces negligible computational overhead, making it suitable for deployment on resource-constrained devices like smartphones or edge devices. During inference, the user leverages the on-cloud TEE to interact with the cloud GPU for KV-cache encoding-decoding operations. This design minimizes communication overhead while taking advantage of cloud GPU acceleration. Through these designs, SCX is a practical solution for confidential

Transformer serving, as it can be easily deployed in existing cloud environments with minimal modifications.

Contributions of this work are summarized as follows:

- We propose a new concept for cloud-scale confidential Transformer serving: stateless KV-cache encoding, which introduces a novel trade-off between privacy protection and efficiency, complementing existing techniques.
- We present SCX, a framework with carefully designed encoding and decoding schemes tailored for Transformer models. We prove the mathematical equivalence of the decoded output to plaintext inference, ensuring no quality loss. We theoretically analyze the protection capability of SCX based on differential privacy and evaluate its robustness against commonly known attacks.
- We implement SCX and evaluate it on four representative large Transformer models of varying sizes (7/13/47/70B). Experimental results show that SCX significantly outperforms SOTA confidential serving approaches [8, 15] in efficiency. For Llama-7B, SCX reduces latency from 17s to just 36ms. Moreover, we show that SCX can be seamlessly combined with KV-cache compression techniques [21, 24] to further reduce 85% memory movement latency.

This work does not raise any ethical issues.

2 BACKGROUND

This section introduces relevant preliminaries on Transformer architecture and KV-cache in incremental generation.

2.1 Transformer Architecture

The Transformer architecture [38] is built upon a sequence of self-attention and feedforward layers, which enable the model to capture global dependencies within input sequences efficiently. The attention, specifically the multi-head self-attention module, is central to the Transformer. For an input sequence of tokens, each token is represented as a vector, and the self-attention computes weighted dependencies between all token pairs. This is achieved through three learned projections: the query (Q), key (K), and value (V) matrices. The attention output for each token is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k represents the dimensionality of the key vectors. By stacking multiple attention heads, the model can capture diverse relationships between tokens. The feedforward module complements the attention by applying two fully connected layers with a non-linear activation function in between. Together, these modules allow the Transformer to model both local and long-range dependencies effectively. Scaling laws and architectural refinements [4, 37] have demonstrated the ability of Transformer-based models to generalize well across a wide range of tasks.

2.2 KV-Cache in Attention

In auto-regressive generation, such as text generation tasks, the model generates tokens sequentially, predicting the next token based on previously generated tokens. While the Transformer excels at modeling sequences, a naive implementation of auto-regressive generation suffers from inefficiencies that make it prohibitively expensive at scale.

In each step of auto-regressive generation, the attention recomputes the full attention matrix over all past tokens for every new token. This recomputation grows quadratically with the sequence length, leading to significant overhead in both computation and memory as the sequence progresses. For example, at step t, the model computes attention over all t tokens, even though the results for earlier tokens remain unchanged. This redundant computation is especially pronounced in large-scale deployments, where serving latency and resource utilization are critical concerns.

To address this, the KV-cache is employed. Instead of recomputing the key and value projections for all past tokens, the KV-cache stores these projections from previous steps. During incremental generation, only the new token's key and value projections are computed and appended to the cache. This allows the attention to retrieve the cached keys and values without recomputation, reducing the per-step computational complexity of incremental generation from $O(t^2)$ to $O(t)$. With the KV-cache, the attention at step t only requires computing:

$$\text{Attention}(Q_t, [K_1, ..., K_{t-1}, K_t], [V_1, ..., V_{t-1}, V_t]),$$

where $K_{1:t-1}$ and $V_{1:t-1}$are retrieved from the cache, K_t and V_t are newly calculated projections of the latest token. KV-cache is particularly useful in cloud-scale settings, where large Transformer models are served to handle long-context, high-throughput, low-latency workloads [21, 24].

3 STATELESS KV CACHING

In this section, we formalize the concepts of stateful and stateless KV-cache. Then we explain the design space and discuss two key technical challenges in detail.

3.1 Formalization

We begin by discussing the stateful nature of the original KV-cache, then introduce a definition of statelessness inspired by Apple PCC [1], and conclude by outlining how our proposed SCX encoding scheme satisfies this definition.

The KV-cache in Transformers is inherently stateful. To understand this, we first define a stateful function [32]:

Definition 1 (Stateful Function). A function f is stateful if its output depends on both new inputs and an internal state that evolves over time. Formally, $f(x_t, S_{t-1}) \mapsto (y_t, S_t)$, where x_t is the new input, S_{t-1} is the internal state from the previous step, y_t is the output, and S_t is the updated state.

In the context of auto-regressive generation with KV-cache, at each generation step t, the attention uses new query, key, and value projections (Q_t, K_t, V_t), as well as cached keys and values $(K_{1:t-1}, V_{1:t-1})$ from prior steps. These cached projections serve as the internal state S_{t-1}, which evolves as new tokens are generated. Thus, the ordinary KV-cache satisfies the definition of statefulness because the next token depends on the cached state.

Inspired by Apple PCC [1], we introduce the notion of a stateless KV-cache. PCC operates by caching data encrypted with a key provided by the user's device and wipes its own copy of the key after

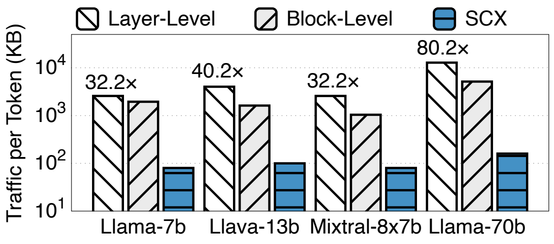


Figure 2: Communication costs per token in incremental generation stage using different encoding strategies.

processing a request. This ensures that the cloud cannot independently regenerate or reuse the cached data without the user's input. Following this principle, we define the statelessness of a KV-cache as follows:

Definition 2 (Statelessness). The on-cloud KV-cache is stateless if the probability that the cloud can generate the next token without access to the user's key is sufficiently low. Let k_u denote the user's key. Formally, the KV-cache is stateless if: $P_{cloud}[f(x_t, S_{t-1}) = f(x_t, S_{t-1}|k_u)] \leq \epsilon$, where $\epsilon \in [0, 1]$.

In essence, the statelessness condition ensures that the cached data in the cloud is encoded such that it is computationally infeasible for the cloud to generate the next token without access to the user's key. In Sec. 5.4, we will formally prove that SCX encoding scheme satisfies the definition of statelessness given above.

3.2 Design Space

Scope. This work focuses on the serving of an end-to-end Transformer model in a client-server setting. Scenarios involving additional parties, such as retrieval-augmented generation (RAG) providers [7] or third-party services like web search engines, are outside the scope of this work.

Threat model. We adopt the classic semi-honest two-party setting [8, 14, 31], which is widely used in privacy-preserving inference research. In this setting:

- User (data owner): The user owns sensitive input and output data, which must remain confidential.
- Cloud (model owner): The cloud hosts the proprietary Transformer model and performs computations.

Both parties are assumed to be semi-honest, meaning that they follow the prescribed protocol but may attempt to infer additional information from obtained messages.

Goals. Our goal is to design a serving scheme that satisfies the following properties under this threat model:

- Data privacy: Recovering the user's original input and output from the revealed information must be computationally infeasible for the cloud.
- Model confidentiality: The user must not gain full access to the proprietary Transformer model.
- Practical efficiency: The quality of service must be sufficient for production environments, achieving performance benchmarks such as serving a 7B large model with a latency of less than 50ms.

Table 2: Attack resistance enhanced by three transformations in SCX encoding scheme.

	Brute-Force	LLM Recovery	Fitting
Plaintext	✗	✗	✗
+Token Perm.	✓	✗	✗
+Redundant Emb.	✓	✓	✗
+One-Time Noise	✓	✓	✓

These assumptions align with practical constraints in cloud confidential computing at scale.

3.3 Challenges

In achieving an effective stateless KV-cache encoding, we encounter two key technical challenges and propose corresponding insights to address them.

Challenge-1: communication complexity. To protect the user's input and output privacy, the most direct idea is to encode all intermediate states using user-controlled keys through a layer-level encoding scheme. However, even if we limit encoding to only the K and V states, since KV-cache acts on each attention block, the overhead during decoding remains prohibitive. For a model with L layers and T generated tokens, both layer-level and block-level encoding schemes incur a communication complexity of $O(T \cdot L)$ between the cloud and the user. In LLM-based text generation tasks, T can reach 10K, while L can range from 32 (for LLaMA-7B) to 80 (for LLaMA-70B). This high communication cost leads to unacceptable delays, as illustrated in Fig. 2.

▹ *Insight 1: Sharing keys for intermediate blocks.*

The forward inference process in Transformers is sequential, meaning that as long as the KV-cache of the first and last blocks remains "locked" the cloud cannot complete the decoding process independently. Leveraging this observation, we propose a decoding scheme that significantly reduces the communication complexity from $O(T \cdot L)$ to $O(T + L)$. Experimental results demonstrate that this design effectively reduces generation latency, enabling us to decode a 7B model with a latency of less than 30ms, making it suitable for real-time production environments.

Challenge-2: attack vulnerability. User data is vulnerable to several known attacks, including brute force, LLM-assisted recovery, and fitting-based inversion through deep model fitting. Simple encoding mechanisms often fail to defend against these attacks:

- Brute Force: Encoding schemes that deploy the embedding layer on the user side (e.g., split inference) are susceptible to brute force attacks, where the cloud can match encoded inputs against the embeddings of the entire vocabulary.
- LLM-Assisted Recovery: Token-level random permutation schemes are vulnerable to LLM-assisted recovery, particularly when the context length is short. Our experiments on GPT-4o confirm the feasibility of such attacks.

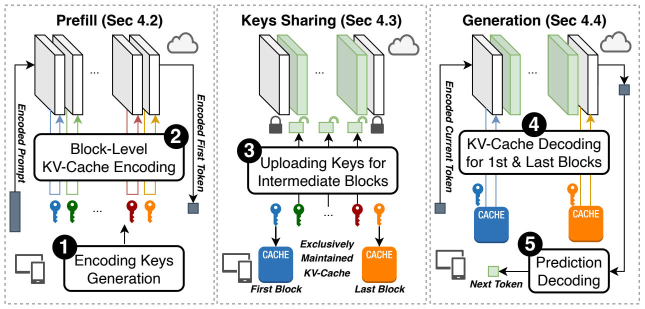


Figure 3: SCX Overview

- Fitting-based inversion: Fixed-key encoding schemes are highly susceptible to inversion attacks via deep model fitting, as attackers can train surrogate models to approximate the encoding mechanism.

▹ *Insight 2: Fixed keys for prefill, one-time keys for generation.*

To address these vulnerabilities, we exploit the two-stage nature of autoregressive inference and adopt distinct encoding strategies for the prefill and generation phases:

- Prefill Phase: During this phase, we use three transformations: token permutation, redundant embedding, and Laplace noise injection. These transformations rely on fixed keys that remain consistent within a single inference session. This balances efficiency with protection against brute force and LLM-assisted recovery attacks.
- Generation Phase: For the decoding of each new token, we employ one-time keys. This ensures that the encoding varies dynamically, making it resistant to fitting-based inversion and other known attacks.

Tab. 2 summarizes the protection capabilities provided by the three Transformations we adopted. Our design is grounded in the theoretical principles of differential privacy, which guarantee bounded information leakage. Experimental evaluations show that the SCX encoding scheme effectively resists common attacks while maintaining practical privacy protection capabilities.

4 SCX DESIGN

Building on the two insights introduced above, this section details the design of SCX across its three stages: prefill, key sharing, and generation. Without loss of generality, our introduction in this section is based on the Transformer of the Llama [37] architecture, and we will discuss support for representative variants in Sec. 6.2.

4.1 Overview

Fig. 3 provides an overview of the SCX framework, which operates in three stages to ensure efficient and privacy-preserving Transformer serving.

In the prefill stage, the user device ❶ generates encoding keys locally. The input embedding is computed on the user device and encoded using these keys. The user then sends the encoded embedding to the cloud. To enable valid inference computations, the cloud interacts with the user for ❷ KV-cache encoding at each Transformer block, i.e., a block-level encoding scheme.

After the prefill stage, the user ❸ uploads the encoding keys for the intermediate Transformer blocks to the cloud. These keys allow the cloud to "unlock" the encoded KV-caches and use them in subsequent computations. However, the keys for the first and last Transformer blocks remain exclusively on the user's device, ensuring the statelessness of the on-cloud inference function.

During the generation stage, the user first encodes the new token embedding using locally maintained one-time keys and sends the encoded token to the cloud. Since the intermediate blocks already have valid KV-caches (unlocked during the key-sharing stage), the cloud only needs to interact with the user for ❹ KV-cache decoding at the first and last Transformer blocks. The cloud computes the encoded output (next-token prediction) and sends it back to the user. The user device then ❺ decodes the prediction to obtain the actual next token.

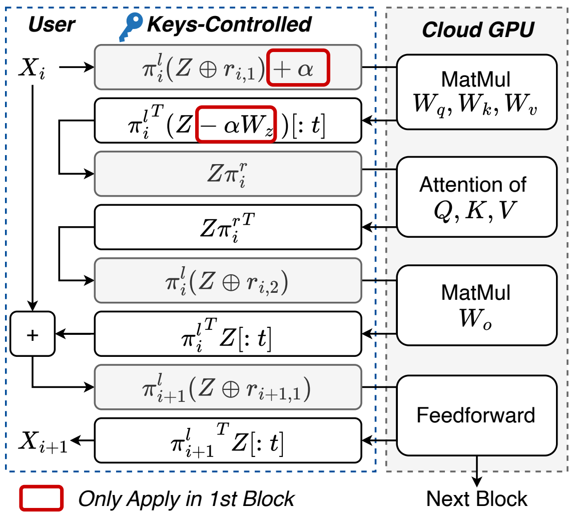


Figure 4: Prefill stage: block-level encoding.

4.2 Prefill Stage

❶ **Encoding keys generation.** Given an L-layer Transformer model, SCX employs three transformations to encode user data. For each transformation, the user generates the following keys locally:

- Token/feature-level permutation matrices: $\{\pi_i^l, \pi_i^r | i \in [L]\}$.
- Redundant embedding: $\{r_{i,1}, r_{i,2} | i \in [L]\}$.
- Laplace noise: $\alpha = \text{Lap}(\mu = 0, s)$, where μ is the expectation of the Laplace distribution and s is the scale parameter.

The user first performs tokenization, token embedding, and normalization locally to obtain the input embeddings. Let $X = [x_1, ..., x_t]$ represent the embeddings for t input tokens after these preprocessing steps. Tab. 3 summarizes the notations used in this design.

❷ **Block-level kv-cache encoding.** As shown in Fig. 4, SCX performs block-level KV-cache encoding as follows:

Table 3: Summary of Notions

Encoding Keys	
$\pi_i^l \in \mathbb{R}^{t\times t}/\pi_i^r \in \mathbb{R}^{d\times d}$	Permutation in token/feature level
$r_i \in \mathbb{R}^{t\times d}$	Redundant embedding
$\alpha \in \mathbb{R}^{t\times d}$	Random Laplace Noise
Operators	
$\oplus$	Concatenation
$[i:j]$	Indexing

(1) **Initial embedding encoding.** The user encodes input embeddings X using the formula: $\pi_i^l(X \oplus r_i) + \alpha$, where $\oplus$ denotes the concatenation. Encoded embeddings are then sent to the cloud.
(2) **Cloud-side computation.** Cloud-Side Computation: The cloud processes the encoded embeddings by performing matrix multiplication with the model weights W_q, W_k, W_v. These weights correspond to the query, key, and value projections, respectively.
(3) **User-side recovery.** The user recovers these projections using the following formula: ${\pi_i^l}^T(Z - \alpha W_{q/k/v})[:t]$, where Z denotes the projection received from the cloud and [] denotes the indexing operator. For the involved multiplication of α and the weight matrix, in SCX implementation, the user does multiplication offline before performing inference. Therefore, user runtime only involves light addition. After recovery, the user performs positional encoding for Q and K.
(4) **Feature permutation.** The user applies feature-level permutation to Q, K, V using π_i^r, resulting in $Q\pi_i^r, K\pi_i^r, V\pi_i^r$. These values are then sent back to the cloud to proceed with the attention computation.
(5) **Attention and re-encoding.** The cloud computes the attention and sends it to the user. The user recovers the output and re-encodes it using the formula: $\pi_i^l(Z \oplus r_{i,2})$. The cloud then performs the linear layer with the weight W_o.
(6) **Residual connection and feedforward.** The user recovers the result, adds it to the initial embedding (residual connection), and re-encodes the result: $\pi_{i+1}^l(Z \oplus r_{i+1,1})$. The cloud processes the feedforward module, and the user recovers the result before proceeding to the next block.

Use of Laplace noise in the first block. As highlighted by the red rectangle in Fig. 4, the Laplace noise α is applied only in the first block. This design choice is crucial for mitigating brute-force attacks: In the first block, the input embeddings X do not yet contain inter-token correlations, making them vulnerable to brute-force attacks (e.g., matching embeddings against the entire vocabulary). The addition of Laplace noise ensures that brute-force attacks on the initial embeddings are computationally infeasible. In subsequent blocks, the embeddings already incorporate inter-token attention, which introduces exponential complexity and makes brute-force attacks (traversing all possible sequences across the vocabulary) infeasible without the user's keys.

4.3 Keys Sharing Stage

❸ **Uploading keys for intermediate blocks.** After applying block-level encoding in the prefill stage, the cloud does not possess valid KV-caches for any Transformer block. If we proceed directly to generation decoding at this point, every Transformer block would require repeated user-cloud interactions, as was done during the prefill stage. This results in a communication complexity of $O(T \cdot L)$, T is the number of generated tokens. Such high complexity is impractical for real-time generation tasks.

To mitigate this communication overhead, SCX introduces a key-sharing mechanism. Specifically, instead of keeping all Transformer blocks encoded, we relax the protection range to include only the first l_s layers and the last l_e layers. The user uploads the encoding keys for all intermediate Transformer blocks, i.e., $\{\pi^l_{l_s+1:L-l_e}\}$, to the cloud. With these keys, the cloud can independently recover valid KV-caches for the intermediate layers using the same recovery operation previously performed by the user: ${\pi_i^l}^\top Z[:t],\ i \in \{l_s+1, ..., L-l_e\}$. This design eliminates the need for user-cloud interactions in the intermediate blocks during the generation stage, significantly reducing the communication complexity from $O(T\cdot L)$ to $O(T+L)$. Our experiments show that setting $l_s = 1$ or 2, and $l_e = 1$, can provide strong privacy protection while enabling efficient generation.

4.4 Generation Stage

❹ **KV-cache decoding for 1st & last blocks.** As shown in Fig. 5, KV-cache decoding is only required for the first and last Transformer blocks.

(1) **First block decoding.** For the first block, the input token embedding x_1 is encoded by adding a one-time Laplace noise α_1 and then sent to the cloud. The cloud processes the encoded embedding by performing matrix multiplications with the model weights. The user then recovers the query, key, and value using the operation $z - \alpha_1 W_z$. Then the user concatenates the key (k_t) and value (v_t) with its local valid KV cache and obtains K_t and V_t. The user re-encodes q_t, K_t, V_t using π_1^r (feature-level permutation), and sends them back to the cloud for attention computation. After receiving the attention output, the user recovers it using $z{\pi_1^r}^T$ and executes the linear layer using W_o locally. Finally, the user sends the exact activations to the cloud, which completes the feedforward module computation.
(2) **Intermediate blocks.** Since the cloud already has the shared intermediate keys, it can independently handle the computations for the 2nd through $(L-1)$-th Transformer blocks without requiring further user interaction.
(3) **Last block decoding.** For the last Transformer block, the cloud first computes the query, key, and value. The cloud sends the value v_t to the user, meanwhile, the cloud performs the attention of q_t and K_t^c based on the on-cloud encoded cache. Specifically, $K_t^c = \pi_L^l(K_t \oplus R)$, where R denotes the redundant embeddings. Therefore, $q_t {K_t^c}^T = q_t(K_t^T \oplus R^T){\pi_L^l}^T$. The user recovers the valid attention using its keys via $q_t K_t^T = q_t {K_t^c}^T \pi_L^l[:t]$. Then the user finishes the subsequent attention operations. For the feedforward module, the user adds a linear constant α_L to the activations before sending

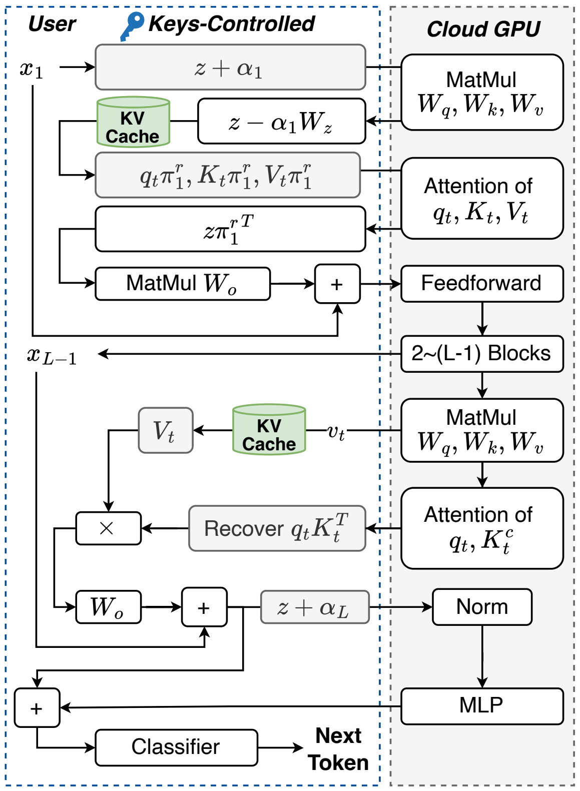


Figure 5: Token generation stage: unlocking intermediate Transformer blocks with configuration $l_s = 1, l_e = 1$ to reduce communication.

them to the cloud. The cloud performs the normalization and multi-layer perceptron (MLP) inference, while the user applies the final residual addition step locally.

❺ **Prediction decoding.** At the end of the generation stage, the user performs the classifier operation locally. Specifically, the user multiplies the output with a linear weight, applies the SoftMax activation, and samples the next token. By default, SCX utilizes one-time keys $(\alpha_1, \pi_1^r, \alpha_L)$ in every decoding step. In Sec. 5.2, we analyze that if we relax this requirement and reuses π_1^r across decoding steps, SCX can potentially outperforms the Nvidia CC solution in multi-GPU deployment.

For token prediction decoding during the prefill stage, the cloud performs the classifier operation. To reduce communication overhead, the cloud sends only the top-k logits of each row instead of the full logits matrix. Since only the user knows which row corresponds to the actual last row, the user can correctly recover the final token prediction on their device without leaking sensitive information.

5 ANALYSIS

This section provides a theoretical analysis of SCX , covering equivalence to ensure lossless inference accuracy, privacy bounds based on differential privacy, and statelessness for KV-cache encoding.

5.1 Inference Equivalence

To guarantee that SCX does not degrade inference quality, we prove the mathematical equivalence of the encoded inference process to the original unencoded computation.

Lemma 1 (Equivalent Inference). *Let Enc, Dec denote SCX encoding and decoding functions, and f denote the inference function. For any input embedding(s) X, the inference output satisfies:*

$$Dec(f(Enc(X))) = f(X).$$

See Appendix A.1 for the proof. This lemma ensures that SCX achieves lossless inference quality. Our experimental results (Tab. 4) validate this theoretical result, demonstrating that our prototype implementation achieves identical accuracy to the original unencoded Transformer model.

5.2 Overhead in Multi-GPU Inference

In this subsection, we illustrate the advantages of SCX over the NVIDIA Confidential Computing (CC) solution in terms of additional overheads for multi-GPU inference, by analyzing boundary conditions in the decoding phase.

In multi-GPU inference of large language models using tensor parallelism [29, 35], attention heads are partitioned across devices, and attention outputs must be synchronized during each auto-regressive decoding step. The standard collective communication primitive for this synchronization is `all_gather`, which collects each GPU's partial attention output and replicates the complete result to every device. Modern communication libraries such as NCCL [27] employ a ring-based `all_gather` implementation by default: each of the N GPUs sends its local fragment to the next GPU in a logical ring, requiring $N-1$ hops per invocation.

Let S_{attn} denote the size (in bytes) of the attention output segment that each GPU must exchange per token, and let B_{aes} (bytes/s) represent the hardware throughput of AES-GCM encryption/decryption on CC-enabled GPUs (e.g., NVIDIA H100). Under the CC threat model, interconnects such as NVLink are considered untrusted [20], and therefore, every GPU-to-GPU transfer must be encrypted and authenticated. The total CC-induced additional latency for the token-level all-gather is thus:

$$T_{\text{cc}} \;=\; (N-1)\,\frac{S_{\text{attn}}}{B_{\text{aes}}}\,. \tag{1}$$

Relaxed reuse of transformation for decoding efficiency. In contrast, SCX eliminates the need for data encryption in GPU-to-GPU communication, as the attention outputs have already been transformed into a privacy-preserving representation. While the default design of SCX enforces one-time keys, i.e., generating a fresh transformation matrix π^r for each decoding step, we consider a practical variant that permits reusing the same π^r across multiple steps. Under this relaxed setting, the decoding process no longer requires transferring the entire transformed key and value caches ($K_t\pi^r$ and $V_t\pi^r$) back to the GPU. Instead, for each decoding step, the SCX CPU-side enclave only needs to return the current token's transformed key and value vectors, $k_t\pi^r$ and $v_t\pi^r$, which are both of size S_{attn}.

These vectors are transmitted in plaintext over PCIe, as the host is within the trusted boundary. Let B_{pcie} denote the effective PCIe

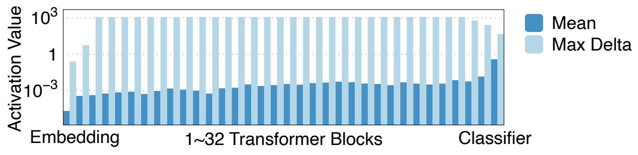


Figure 6: Activation ranges of each layer in Llama2-7b model using WikiText-2 dataset.

bandwidth (in bytes/s). Then the additional latency incurred by SCX per token is:

$$T_{\text{scx}} = \frac{S_k + S_v}{B_{\text{pcie}}} = \frac{2S_{\text{attn}}}{B_{\text{pcie}}}. \quad (2)$$

The condition under which SCX outperforms the CC-secured all-gather becomes:

$$T_{\text{scx}} < T_{\text{cc}} \iff N > 2 \cdot \frac{B_{\text{aes}}}{B_{\text{pcie}}} + 1. \quad (3)$$

Example and practical GPU count ranges. Suppose $B_{\text{pcie}} = 128\,\text{GB/s}$, $B_{\text{aes}} = 200\,\text{GB/s}$. Then the inequality becomes $N > 2 \cdot \frac{200}{128} + 1 = 4.125$, implying that SCX is potentially more efficient when $N \geq 5$. This GPU count is well within the range used by current large-scale deployments (e.g., 16 H100 GPUs for the 671B DeepSeek-R1 model). As model sizes and sequence lengths continue to grow, the required number of GPUs will increase, making such thresholds increasingly common.

5.3 Privacy Guarantee

SCX incorporates differential privacy through the addition of Laplace noise during the encoding process to protect user embeddings. The scale parameter for the Laplace noise is defined as $s = \frac{\Delta f}{\epsilon}$, where $\Delta f = \max|f(x) - f(y)|$ represents the maximum absolute difference of activation values, and ϵ is the privacy parameter. To provide an intuitive understanding of Δf, Fig. 6 shows the mean and maximum absolute differences of activation values across layers of the LLaMA2-7B model. Smaller values of ϵ indicate stronger privacy guarantees but also introduce larger noise. Adding Laplace noise $\text{Lap}(\mu = 0, s)$ ensures $(\epsilon, 0)$-differential privacy.

While adding noise can degrade utility in traditional differential privacy schemes, SCX eliminates this limitation. As shown in Lemma 1, the added noise is equivalently removed during user-side decoding, ensuring that the inference quality remains unaffected. In practice, setting $\epsilon \leq 1$ is generally considered a strong privacy guarantee. From our experiments on the Llama2-7B model, the maximum difference of embedding values is only 0.25. Setting the noise scale s to values like 5 or 10 provides a robust privacy guarantee, making it computationally infeasible for the cloud to infer the original embeddings. Our experiments using distance correlation further confirm that noisy embeddings and randomly generated vectors are statistically indistinguishable, demonstrating that SCX provides strong privacy protection.

5.4 KV-Cache Statelessness

SCX ensures statelessness (Definition 2), meaning that the cloud cannot independently generate the correct next tokens without access to user-provided keys. We formalize this property in the following lemma:

Lemma 2 (Stateless KV-Cache). *Let $x \in \mathbb{R}^d$ denote the input embedding, S the on-cloud encoded KV-cache, and V the vocabulary size. The probability that the cloud generates the correct next token without the user's key k_u is bounded by:*

$$P_{cloud}[f(x, S) = f(x, S|k_u)] \leq max(\frac{1}{d!}, \frac{exp(\epsilon)}{V})$$

.

The proof is provided in Appendix A.2. For LLMs, d (the embedding dimension) typically ranges from 1k to 8k, making $1/(d!)$ an astronomically small value. V, the vocabulary size, is usually around 30k. $\epsilon = \Delta x/s$, which reflects the normalized difference in embedding values and noise scale, can be set to a small value (e.g., 0.01). Given these parameters, the resulting probability bound is extremely small, ensuring that it is computationally infeasible for the cloud to infer the correct next token without the user's key. This guarantees that SCX satisfies the definition of statelessness and prevents unauthorized token generation by the cloud.

6 IMPLEMENTATION

We implemented SCX in Python using PyTorch and HuggingFace `transformers` (open-sourced at https://github.com/yuanmu97/scx). To implement token and feature-level permutation efficiently, we employ a simple indexing-based approach: rather than constructing full 2D permutation matrices and performing matrix multiplications, we randomly shuffle a 1D index array and apply it directly via the `torch.Tensor` indexing operator. For portability across different TEE platforms, users can package the runtime using Confidential Containers [6], which avoids any dependency on proprietary TEE subsystems or vendor-specific SDKs.

6.1 TEE Integration

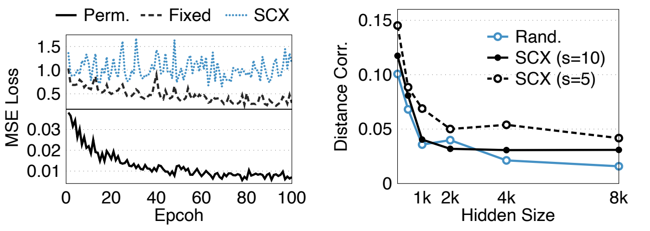
To ensure data privacy while minimizing communication overhead, we integrate SCX with Trusted Execution Environments (TEE). The TEE handles intermediate decoding and re-encoding operations. By deploying these operations inside the TEE, we replace cross-device communication with local PCIe memory movements within the same cloud host, reducing latency and improving throughput.

Clarification on TEE usage. While SCX can leverage a TEE to isolate sensitive decoding and re-encoding steps, it does not strictly depend on one. Traditional TEEs incur overhead through full memory encryption/decryption on every transfer, which SCX avoids by only moving pre-encoded data between secure and non-secure domains. In deployments where a TEE is unavailable or undesirable, SCX can fall back to device-cloud communication. Thus, SCX remains compatible with both TEE-based and non-TEE environments, allowing system integrators to choose the trust model and performance trade-off that best fits their deployment.

Comparison with full-TEE solutions. In full-TEE (CPU) approaches, both model weights and all data reside entirely within the enclave, leading to significant performance bottlenecks due to the limited CPU resources. In contrast, SCX confines only the lightweight decoding and re-encoding logic to the TEE, if used, while offloading the heavy model inference onto the GPU. The

Table 4: Serving normal Llama2-7b, Mixtral-8x7b models, quantized TinyLlama-1.1b and Qwen-7b models with SCX shows lossless inference.

Prec. Bits	Max Abs. Diff. 7b	Max Abs. Diff. 8x7b	Acc.	Prec. Bits	Max Abs. Diff. 1.1b	Max Abs. Diff. 7b	Acc.
FP32	4.5e-5	5.6e-5	1.0	INT8	0.012	0.023	1.0
FP16	5.1e-3	7.2e-3	1.0	INT4	0.016	0.025	1.0



(a) Fitting-based Attack **(b) Distance Correlation**

Figure 7: SCX's privacy protection performance.

encoded KV-caches exchanged between the CPU TEE and GPU do not require additional encryption during transit, further simplifying implementation and preserving high throughput without sacrificing privacy.

6.2 Transformer Variants

SCX is designed to be flexible and can seamlessly support a variety of Transformer architectures, including multi-modal models and mixture-of-experts (MoE) designs. Below, we describe how SCX integrates with key variants.

Multi-modality. Vision Transformers (ViT) [9] divide an image into non-overlapping patches, which are then linearly embedded to create a sequence of token embeddings. These token embeddings are fed into a Transformer model for processing. Since SCX operates on token embeddings and does not depend on the preprocessing of the original data, it can seamlessly support ViT without requiring modifications. LLaVA [23] processes both text and images as inputs. It uses a ViT to embed the image and then combines the visual embeddings with text embeddings through a linear projection. To integrate LLaVA with SCX: (1) The ViT computations are performed using SCX. (2) The linear projection for visual embeddings is executed within the TEE. (3) The concatenated visual and text embeddings are then processed by SCX.

Mixture-of-experts. Mixtral [17] incorporates MoE into Transformers by constructing multiple feedforward modules (experts) in parallel. A router (or gating layer) determines the weights for these experts via $g(x) = xW_g$, where $W_g \in \mathbb{R}^{d\times e}$ and e represents the number of experts. SCX naturally supports MoE models because the encoded data passed to the feedforward module is equivariant to linear multiplication.

Our experiments on both LLaVA and Mixtral models demonstrate that SCX flexibly supports these Transformer variants.

7 EVALUATION

7.1 Setup

Models and datasets. We use six open-source and representative large Transformer models with varying sizes: TinyLlama-1.1b-GPTQ [41], Llama2-7b [37], Qwen-7b-GPTQ [18], Llava-13b [23], Mixtral-8x7b [17], Llama2-70b [37]. These models cover text and image modalities, dense and MoE architectures, and full-precision and quantized representations. For text input, we use the WikiText-2 [25] and NarrativeQA [2] datasets, while for image input, we use the VQA [13] dataset.

Devices. User device: MacBook Air, Apple M3 CPU. Cloud server: (1) 4x NVIDIA A100 GPUs server; (2) 4x NVIDIA 4090 GPUs server. Unless otherwise stated, models by default are loaded in FP16 precision and tested on the A100 server.

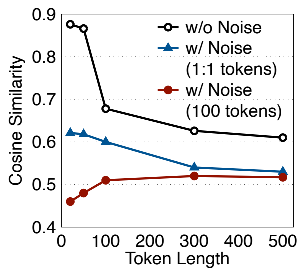
Baselines. We compare SCX with three Transformer serving approaches: (1) Plaintext, running in the cloud GPU; (2) Puma [8], SOTAcryptography-based protocol for Llama models that combines MPC and HE primitives; (3) Full-TEE, running in the cloud CPU TEE. And we further integrate SCX with two KV-cache compression approaches: (4) InfiniGen [21] and (5) CacheGen [12].

Metrics. We evaluate SCX based on the following metrics: For output quality, we consider (1) token prediction accuracy and (2) perplexity (PPL) scores. For quantification of privacy leakage, we consider (3) the cosine similarity of tokens and (4) the distance correlation of vectors. For efficiency, we consider (5) Time-To-First-Token / TTFT during prefill, (6) latency and throughput during generation, and (7) communication cost.

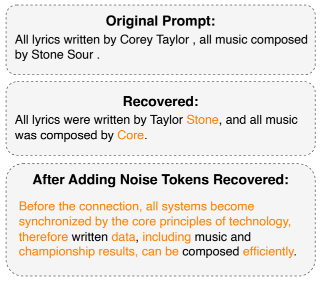
7.2 Inference Output Quality

Lossless inference. We evaluate the accuracy and perplexity (PPL) of four Transformer models on the WikiText-2 dataset in both plaintext mode and SCX mode. Results for the Llama2-7b and Mixtral-8x7b models are shown in Tab. 4. Experimental results confirm our mathematical proof of equivalence: The PPL difference (Δ PPL) is zero and the accuracy is 100%, with SCX achieving identical predictions to the plaintext model. The accuracy here is calculated by treating the plaintext model's output as the ground truth, ensuring that SCX preserves inference quality.

Sensitivity to data representations. While SCX outputs are theoretically equivalent, minor differences emerge due to hardware-level representation errors. We measured the maximum absolute difference in output logits on Wikitext-2 under various numerical precisions. Specifically, we compared standard FP32 and FP16 formats, as well as quantized INT8 and INT4 versions of the TinyLlama-1.1B and Qwen-7B models. Results are summarized in Tab. 4. We observed that using reduced-precision formats slightly increases numerical discrepancies. For FP16, the maximum absolute differences ranged from 4.5×10^{-5} to 7.2×10^{-3}, which are negligible given the logits range of -17.5 to $+30.9$. For quantized models, the differences were slightly higher: up to 0.012 and 0.016 for INT8 and INT4 on TinyLlama-1.1B, and 0.023 and 0.025 respectively on Qwen-7B. Despite these differences, all tested precisions led to identical token predictions, i.e., 100% accuracy. This confirms the robustness of SCX to low-precision and quantized representations.



(a) Cosine Similarity



Original Prompt:
All lyrics written by Corey Taylor , all music composed by Stone Sour .

Recovered:
All lyrics were written by Taylor Stone, and all music was composed by Core.

After Adding Noise Tokens Recovered:
Before the connection, all systems become synchronized by the core principles of technology, therefore written data, including music and championship results, can be composed efficiently.

(b) Recovery Sample

Figure 8: LLM-assisted token sequence recovery.

7.3 Privacy Protection

Fitting-based attacks. We evaluated SCX's resistance to fitting-based attacks using a simple surrogate model composed of two linear layers. The surrogate model attempts to reconstruct embeddings by minimizing the mean squared error (MSE) over 100 epochs. We compared three transformations: permutation-only, fixed noise, and SCX's one-time noise. Permutation-only embeddings were easily fitted, with MSE decreasing rapidly. Fig. 7a shows that fixed noise embeddings showed a steady decrease in MSE, indicating vulnerability to fitting. SCX's one-time noise prevented fitting, with loss values fluctuating throughout training, demonstrating robust resistance

Distance correlation. We measured the distance correlation between SCX-encoded embeddings and the original embeddings across different hidden sizes. Experimental results (see Fig. 7b) show that, with a scale parameter $s = 5, 10$, SCX-encoded embeddings are statistically indistinguishable from randomly generated vectors. This strong privacy protection ensures that encoded embeddings reveal no meaningful information about the original data.

LLM-assisted recovery. Recent advanced attacks against token-level permutations involve using large language models (LLMs) to recover the original token sequence from shuffled tokens. We evaluated this attack using GPT-4o on the WikiText-2 dataset. We tested SCX with and without redundant token embeddings. Redundant embeddings are appended to increase resistance to recovery. We consider the cosine similarity between recovered tokens and the original tokens as the metric. As shown in Fig. 8a, without redundant tokens, GPT-4o achieved high recovery accuracy for short sequences, e.g., cosine similarity exceeding 0.87 for sequences of length 20. Adding 100 redundant tokens significantly reduced similarity to 0.46, making recovery much harder. This demonstrates that SCX's redundant token strategy effectively enhances resistance to LLM-assisted recovery, especially for shorter sequences. A specific recovery example is shown in Fig. 8b.

Impact of sharing intermediate keys in decoding. To explicitly evaluate whether sharing intermediate keys with the cloud affects privacy protection during token decoding, we measure both token-level perplexity and embedding similarity on the WikiText-2 and NarrativeQA datasets. We consider four decoding strategies: (1) *Normal*, which uses the original (plaintext) KV-cache; (2) *ALL-zeros*, which replaces all KV-caches with zeros; (3) *ALL-encoded*, which uses SCX-encoded KV-caches in all layers; and (4) *SCX-NearestReuse*, in which layers $\{0, 1, -1\}$ are SCX-encoded, but during decoding, we simulate an adversarial policy by reusing KV-caches from the nearest available layers, i.e., $\{2, 2, -2\}$. Experimental results are shown in Tab. 5. The original model achieves low token-level perplexity (1.33 on WikiText-2 and 2.61 on NarrativeQA). In contrast, both the ALL-zeros and ALL-encoded settings lead to extremely high perplexity (over 40,000), indicating that the model output becomes meaningless when valid KV-caches are unavailable. Sharing intermediate-layer keys allows the cloud to reconstruct those layers' KV-caches and reduces the perplexity accordingly. However, even under the SCX-NearestReuse attack, where the cloud attempts to bypass SCX protection by reusing KV-caches from neighboring layers, the perplexity remains exceedingly high (around 20,000), suggesting that the model still fails to generate coherent output and that user privacy remains strongly protected. To further quantify the effectiveness of SCX, we also compare the cosine similarity between the final-layer embeddings produced under attack and those obtained under normal decoding. For SCX-NearestReuse, the similarities are 0.49 on WikiText-2 and -0.117 on NarrativeQA, both indicating weak correlation with the true embeddings. In comparison, the ALL-zeros setting yields similarities of 0.37 and -0.011, respectively. These results confirm that SCX encoding effectively prevents meaningful KV-cache recovery, even when partial keys are shared and reuse attacks are attempted.

Table 5: Effect of KV-cache stateless encoding on LLaMA2-7B model performance across Wikitext-2 and LongBench NarrativeQA datasets. SCX-NearestReuse encodes {1,2, -1} layers and reuse KV-cache in {2, 2, -2} layers, respectively.

Dataset	KV-Cache	Token PPL	Emb. Sim.
Wikitext-2	Normal	1.33	1.00
	ALL-Zeros	57396	0.37
	ALL-Encoded	73566	0.23
	SCX-NearestReuse	19108	0.49
NarrativeQA	Normal	2.61	1.00
	ALL-Zeros	73363.1	-0.011
	ALL-Encoded	40017.8	0.025
	SCX-NearestReuse	21057.8	-0.117

Table 6: Prefill stage of serving Llama-7b/70b models with context length 8.

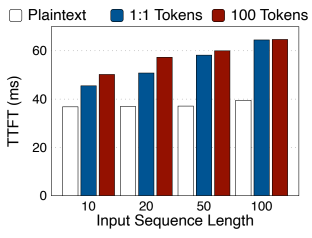
Method		TTFT		Comm. Cost	
		7b	70b	7b	70b
Unprotected	Plaintext	10ms	112ms	18B	18B
Protected	Puma	200s	N/A	1.8GB	N/A
	Full-TEE	17s	182s	18B	18B
	SCX	**36ms**	**324ms**	72KB	144KB

7.4 Efficiency

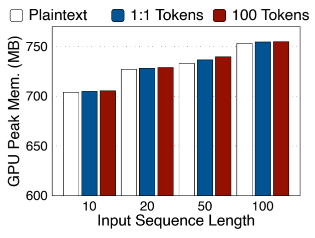
TTFT in the prefill stage. Tab. 6 compares the Time-To-First-Token (TTFT) in the prefill stage for four different approaches:

Table 7: One more token generation of serving Llama-7b/70b models with context length 8.

Method		Latency 7b	Latency 70b	Comm. Cost 7b	Comm. Cost 70b
Unprotected	Plaintext	28ms	202ms	2B	2B
Protected	Puma	164s	N/A	2.0GB	N/A
	Full-TEE	4s	26s	2B	2B
	SCX	**30ms**	**208ms**	8KB	16KB



(a) TTFT of Llama-1.1b-int4



(b) GPU Peak Memory

Figure 9: Sensitivity to redundant embeddings with different ISL on TinyLlama-1.1b-Int4 model.

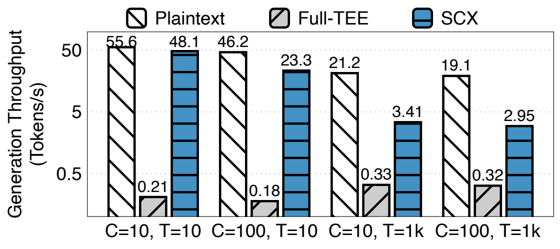
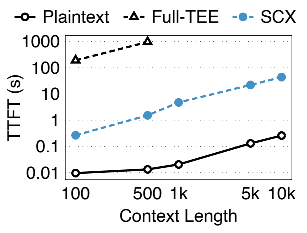
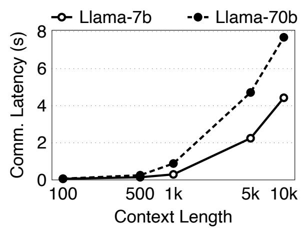


Figure 10: Generation throughput, llama-7b, C denotes the context length, T denotes the number of new generated tokens.



(a) TTFT of Llama-7b



(b) Communication Latency

Figure 11: Context length sensitivity in prefill stage.

unprotected plaintext, Puma, full-TEE, and SCX. The experiments were conducted on both the Llama2-7b and Llama2-70b models with a context length of 8. The results demonstrate that Puma and full-TEE incur significant delays, with TTFT reaching 200s and 17s, respectively. In contrast, SCX achieves a dramatic reduction in TTFT to just 36ms. Compared to plaintext serving, which achieves a TTFT of 10ms, SCX provides a practical trade-off between efficiency and privacy protection. These results underline the effectiveness of SCX in maintaining low latency during the prefill stage while preserving strong privacy guarantees.

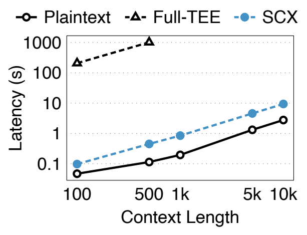
Overheads introduced by additional embeddings. As discussed above, when the input sequence length (ISL) is short (e.g., less than 100 tokens), SCX encoding requires the concatenation of redundant noise embeddings to defend against LLM-based token recovery attacks. However, this inevitably introduces additional computational and memory overheads. We evaluate this overhead by measuring the time-to-first-token (TTFT) and GPU peak memory usage under different noise injection policies and ISL values ranging from 10 to 100. Specifically, we compare two policies: (1) *1:1 Tokens*, where the number of redundant tokens equals the number of input tokens; and (2) *Fixed 100 Tokens*, where a constant 100 redundant tokens are added regardless of ISL. As shown in Fig. 9, both policies incur moderate overhead: TTFT increases by approximately 13–25 ms, and GPU memory consumption increases by less than 10 MB. Given that such redundant noise is only required in scenarios with very short input sequences, we consider the overhead to be minor and acceptable in practice.

Latency in the generation stage. Tab. 7 reports the generation latency for the same four approaches. Puma and full-TEE solutions exhibit substantial delays, with latencies of 164s and 4s, respectively. SCX, however, achieves a latency of only 30ms, which is remarkably close to the plaintext latency of 28ms. This result highlights the efficiency of SCX in the generation stage, delivering near-plaintext performance while ensuring privacy protection. Achieving such low latency is crucial for real-time applications, making SCX a highly practical solution compared to cryptography-based or full-TEE methods.

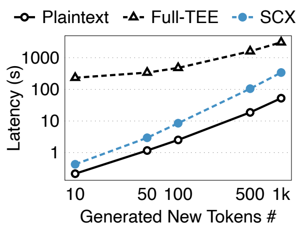
Communication cost. The user-cloud communication cost for SCX is reported in Tab. 6 and Tab. 7. SCX incurs only KB-level communication cost per generated token, which is affordable even for devices with limited bandwidth, such as mobile phones. This low communication overhead makes SCX a scalable and efficient solution for privacy-preserving LLM serving.

Generation throughput. Fig. 10 illustrates the generation throughput (tokens per second) of the Llama2-7b model using different context lengths (C) and numbers of generated tokens (T). The results compare plaintext, full-TEE, and SCX approaches. When the context length is small ($C = 10$), SCX achieves a throughput of 48.1 tokens/s, which is close to plaintext throughput (55.6 tokens/s). As the context length increases ($C = 100$), SCX maintains a throughput of 3.41 tokens/s, significantly outperforming the full-TEE solution, which achieves only 0.33 tokens/s. This represents a 10.3× speedup, demonstrating the scalability of SCX for longer contexts and its advantage over full-TEE deployments.

Sensitivity to context length. Handling long context lengths is a critical requirement for serving large language models. We evaluated the TTFT of the Llama2-7b model in the prefill stage with context lengths ranging from 100 to 10K. Fig. 11 shows that even for a 10K context, SCX achieves a practical balance between efficiency and privacy protection. SCX is over 1000× faster than full-TEE deployment while being only around 10× slower than



(a) Generation Latency



(b) Context Length=100

Figure 12: Generation using Llama2-7b with varying context length and number of generated tokens.

unprotected plaintext deployment. However, the communication cost increases with both the context length and the model size, as shown in Fig. 11b. This is because SCX must transmit encoded embeddings to the cloud, which scales with the input size. Despite this, the trade-off remains practical for serving large models.

Generation for different lengths. We also evaluated SCX's generation latency for the Llama2-7b model with context lengths ranging from 100 to 10K. Fig. 12a shows that SCX achieves near-plaintext latency even for long contexts. For example, with a 10K context length, SCX achieves a generation latency of 9.3s, compared to 2.7s for plaintext. Additionally, we tested SCX's performance when generating varying numbers of new tokens. As shown in Fig. 12b, with a context length of 100 and 1K new tokens, SCX's generation latency is 330s, compared to 53s for plaintext. These results highlight the effectiveness of SCX's key-sharing design, which significantly reduces the computational complexity during the generation stage. This optimization is critical for maintaining practical performance in real-world applications requiring long contexts and large-scale token generation.

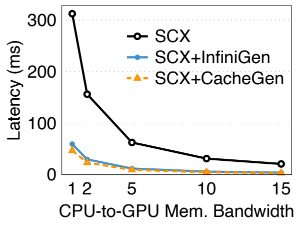
7.5 SCX with KV-Cache Compression

Latency breakdown. To better understand the performance bottlenecks, we report the latency breakdown of the Llava-13b model in Tab. 8. Experimental results reveal that memory movement between the cloud GPU and the CPU-based TEE dominates the latency, accounting for 94% in the prefill stage and 86% in the generation stage. This highlights memory movement as the primary contributor to overall latency.

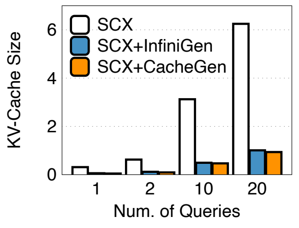
Memory movement bandwidth. For the Llama2-7b model, we analyzed the proportion of latency caused by memory movement compared to computation on the TEE CPU and GPU. Experiments show that memory movement dominates the runtime latency on the GPU, while it occupies a much smaller share of the runtime latency on the CPU. To investigate the cause of this discrepancy, we measured the CPU-GPU memory bandwidth of our cloud server: CPU-to-GPU bandwidth: 17.96 GB/s and GPU-to-CPU bandwidth: 2.89 GB/s. The results show that the GPU-to-CPU bandwidth is significantly lower than the CPU-to-GPU bandwidth, making GPU-to-CPU memory movement a critical bottleneck. This motivates the integration of KV-cache compression techniques to further reduce latency.

Table 8: Latency breakdown of Llava-13b model with context length=1k.

	Emb.	Comm.	Comp.	Mem.	Clf.	Total
Prefill	2.1	298	21	**4728**	1.9	5051 ms
Gen.	0.04	0.3	76	**481**	0.4	558 ms



(a) Mem. Movement Latency



(b) Concurrent Queries

Figure 13: Combines SCX with KV-cache compression approaches to further optimize the efficiency.

Integrating KV-cache compression. To address the memory movement bottleneck, we integrated SCX with state-of-the-art KV-cache compression techniques, including CacheGen and InfiniGen. We evaluated the effect of these techniques on memory movement latency under varying bandwidth conditions. As shown in Fig. 13a, integrating CacheGen and InfiniGen significantly reduces the CPU-to-GPU memory movement latency. Naturally, the optimization effect diminishes as the available bandwidth increases. Additionally, we examined the KV-cache size under different numbers of concurrent queries. Fig. 13b demonstrates that applying CacheGen and InfiniGen effectively reduces the KV-cache size, which can grow very large under highly concurrent workloads. This reduction is critical for maintaining efficient memory movement and avoiding excessive latency in scenarios with high query concurrency.

8 RELATED WORK

KV-cache management. Recent research in LLM KV-cache management [22] has focused on optimizing efficiency, particularly in long context generation. InfiniGen [21] conducts minimal rehearsals with current layer inputs and parts of the query weight from the next layer, allowing for selective prefetching of essential KV cache entries rather than fetching all entries. CacheGen [24] splits long contexts into manageable chunks, encoding each separately at varying compression levels based on available bandwidth, akin to video streaming techniques. CachedAttention [12] employs hierarchical KV caching, scheduler-aware management, and techniques to handle context window overflow, significantly improving inference efficiency.

SCX can work complementarily with these approaches, further reducing the communication overheads involved by our encoding-decoding scheme.

Confidential model serving. Existing approaches fall into three categories: (1) Cryptography-based methods [8, 14, 31] offer strong privacy via techniques like MPC and HE, but incur high communication overheads and accuracy loss due to nonlinear approximations. (2) TEE-based methods [16, 26, 42] isolate computation in secure enclaves but suffer from low throughput on large models due to CPU bottlenecks. (3) Access-control-based methods, such as Apple PCC [1] and Nvidia CC [28], combine high-end specialized hardware with policy enforcement for efficient secure inference, but require costly and customized deployment.

SCX introduces a new trade-off via stateless KV cache encoding, offering a scalable and efficient alternative that complements existing privacy-preserving mechanisms.

9 DISCUSSION

Support for RAG and third-party components. SCX can be extended to support Retrieval-Augmented Generation (RAG) while maintaining privacy guarantees. In a typical RAG pipeline, the user query is first embedded and matched against a large-scale vector database to retrieve semantically relevant documents, which are then fed as context into the language model for generation. However, both the user embeddings and the retrieved documents can leak sensitive semantic information if processed in untrusted environments, as similarity scores and retrieval results directly reflect user intent and content.

To address this, we can execute the entire retrieval phase securely within a Trusted Execution Environment (TEE). Specifically, the user embedding is sent into the TEE, which then performs a search over the pre-loaded vector index that resides in TEE memory. Recent TEE technologies, such as Intel TDX, support large memory capacities (e.g., 256 GB–1 TB) [15], which are sufficient to accommodate large-volume RAG workloads. The retrieval output, typically a small set of top-*k* plaintext documents or passages, is not exposed outside the TEE. Inside the TEE, these retrieved passages are concatenated with the original user prompt to form the final input context, following standard RAG workflows. This combined input is then processed through the standard SCX pipeline: it is first encoded, transformed into a privacy-preserving representation, and finally sent to the untrusted cloud for inference. By treating the augmented prompt as a unified input, SCX naturally supports RAG without changing the core system architecture.

Moreover, SCX does not rely exclusively on TEE. Recent privacy-preserving retrieval protocols such as RemoteRAG [5] provide cryptographic alternatives that protect both embeddings and document access patterns. SCX can flexibly integrate these as drop-in replacements for the TEE-based retrieval module, allowing system designers to choose between hardware-based or protocol-based protection based on trust assumptions and performance requirements.

SCX under disaggregated prefill/decode/caching architectures. Recent large-scale LLM serving systems increasingly adopt disaggregated architectures that separate the prefill, decode, and caching stages [19, 43]. In such designs, caching components are responsible for managing and serving KV caches across decoding sessions, potentially on remote nodes.

Our SCX framework can extend to this setting: instead of relying solely on a CPU-side TEE colocated with each GPU, a more general deployment model can delegate the caching layer to centralized trusted hardware in the cloud (e.g., Intel TDX or GPU-side secure enclaves such as NVIDIA H100 CC). This architecture enables secure caching disaggregation, where KV caches are transformed, stored, and served from secure memory, while inference can still run efficiently on untrusted accelerators.

Such a direction aligns with the broader trend of decoupling computation and memory in LLM systems and opens new opportunities for privacy-preserving inference. We leave a full system design along this line as future work.

10 CONCLUSION

This work addresses the critical challenge of achieving privacy and efficiency in serving Transformer models by introducing stateless KV-cache encoding. Through innovative encoding and decoding schemes, SCX minimizes communication complexity and resists known attack methods, while maintaining mathematical equivalence to plaintext inference. Extensive evaluations on Transformer models of varying sizes demonstrate that SCX significantly improves upon existing solutions in efficiency and scalability.

ACKNOWLEDGMENTS

We thank the anonymous SIGCOMM reviewers and shepherd for their constructive comments. The research is partially supported by the China National Natural Science Foundation with No.623B2093, No.62441228, National Key R&D Program of China 2021YFB2900103, RGC Theme-based Research Scheme T43-513/23-N, and Strategic Topics Grant STG1/E-403/24-N.

REFERENCES

[1] Apple. 2025. Private Cloud Compute: A new frontier for AI privacy in the cloud. https://security.apple.com/blog/private-cloud-compute/

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 3119–3137. https://doi.org/10.18653/v1/2024.acl-long.172

[3] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. 2023. Accurate medium-range global weather forecasting with 3D neural networks. *Nature* 619, 7970 (2023), 533–538.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Yihang Cheng, Lan Zhang, Junyang Wang, Mu Yuan, and Yunhao Yao. 2024. RemoteRAG: A Privacy-Preserving LLM Cloud RAG Service. *arXiv preprint arXiv:2412.12775* (2024).

[6] Confidential Containers Maintainers. 2025. Confidential Containers. https://github.com/confidential-containers/confidential-containers. Accessed: 2025-06-13.

[7] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 719–729.

[8] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Cheng. 2023. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533* (2023).

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR* (2021).

[10] Minxin Du, Xiang Yue, Sherman SM Chow, Tianhao Wang, Chenyu Huang, and Huan Sun. 2023. Dp-forward: Fine-tuning and inference on language models with differential privacy in forward pass. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2665–2679.

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1322–1333.

[12] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 111–126.

[13] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[14] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. 2024. Sigma: Secure gpt inference with function secret sharing. In *2024 Privacy Enhancing Technologies Symposium*. 61–79.

[15] Intel. 2025. Intel Trust Domain Extensions.

[16] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.

[17] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[18] et al. Jinze Bai. 2023. Qwen Technical Report. *arXiv preprint arXiv:2309.16609* (2023).

[19] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[20] Jonghyun Lee, Yongqin Wang, Rachit Rajat, and Murali Annavaram. 2025. Characterization of GPU TEE Overheads in Distributed Data Parallel ML Training. *arXiv preprint arXiv:2501.11771* (2025).

[21] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 155–172.

[22] Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2024. A Survey on Large Language Model Acceleration based on KV Cache Management. *arXiv preprint arXiv:2412.19442* (2024).

[23] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26296–26306.

[24] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia). 38–56.

[25] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]

[26] Apoorve Mohan, Mengmei Ye, Hubertus Franke, Mudhakar Srivatsa, Zhuoran Liu, and Nelson Mimura Gonzalez. 2024. Securing AI Inference in the Cloud: Is CPU-GPU Confidential Computing Ready?. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. IEEE, 164–175.

[27] NVIDIA. 2025. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl. Accessed: 2025-06-12.

[28] NVIDIA. 2025. NVIDIA Confidential Computing: Secure Data and AI Models in Use. https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/

[29] NVIDIA Corporation. 2025. TensorRT-LLM: An open-source library for optimizing Large Language Model inference. https://github.com/NVIDIA/TensorRT-LLM. Accessed: 2025-07-23.

[30] OpenAI. 2025. ChatGPT. https://openai.com/blog/chatgpt.

[31] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4753–4771.

[32] Dusko Pavlovic. 2023. Stateful Computing. In *Programs as Diagrams: From Categorical Computability to Computable Categories*. Springer Nature, 145–164. https://doi.org/10.1007/978-3-031-34827-3_7

[33] Open Worldwide Application Security Project. 2025. OWASP Top 10 for LLM Applications 2025. *whitepaper* (2025).

[34] Jerret Ross, Brian Belgodere, Vijil Chenthamarakshan, Inkit Padhi, Youssef Mroueh, and Payel Das. 2022. Large-scale chemical language representations capture molecular structure and properties. *Nature Machine Intelligence* 4, 12 (2022), 1256–1264.

[35] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[36] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine* 29, 8 (2023), 1930–1940.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[39] Shenao Yan, Shen Wang, Yue Duan, Hanbin Hong, Kiho Lee, Doowon Kim, and Yuan Hong. 2024. An LLM-assisted easy-to-trigger backdoor attack on code completion models: injecting disguised vulnerabilities against strong detection. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 101, 18 pages.

[40] Linyi Yang, Jiazheng Li, Ruihai Dong, Yue Zhang, and Barry Smyth. 2022. Numhtml: Numeric-oriented hierarchical transformer model for multi-task financial forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 11604–11612.

[41] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. arXiv:2401.02385 [cs.CL]

[42] Wei Zheng, Ying Wu, Xiaoxue Wu, Chen Feng, Yulei Sui, Xiapu Luo, and Yajin Zhou. 2021. A survey of Intel SGX and its applications. *Frontiers of Computer Science* 15 (2021), 1–15.

[43] Pengfei Zuo, Huimin Lin, Junbo Deng, Nan Zou, Xingkun Yang, Yingyu Diao, Weifeng Gao, Ke Xu, Zhangyu Chen, Shirui Lu, et al. 2025. Serving Large Language Models on Huawei CloudMatrix384. *arXiv preprint arXiv:2506.12708* (2025).

Appendices are supporting material that has not been peer-reviewed.

A PROOFS

A.1 Proof of Lemma. 1

To prove that the output after applying SCX encoding and decoding equals the original inference output, we just need to prove that in each round of user-cloud interactions, the intermediate results are equivalent.

First, we consider the prefill stage.

Proof. For the 1st round interaction, take the computation of Q as an example:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec((\pi^l(X \oplus r) + \alpha)W_q)\\
&= Dec(\pi^l(XW_q + \alpha W_q) \oplus (rW_q + \alpha W_q))\\
&= {\pi^l}^T \pi^l((XW_q + \alpha W_q) \oplus (rW_q + \alpha W_q) - \alpha W_q)[:t]\\
&= (XW_q \oplus rW_q)[:t]\\
&= XW_q\\
&= f(X).
\end{aligned}$$

K, V same as above. For the 2nd round interaction:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec(Attn(Q\pi^r, K\pi^r, V\pi^r))\\
&= Dec(\text{softmax}\left(\frac{Q\pi^r {\pi^r}^T K^T}{\sqrt{d_k}}\right)V\pi^r)\\
&= Dec(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\pi^r)\\
&= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\pi^r {\pi^r}^T\\
&= \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\pi^r\\
&= f(X).
\end{aligned}$$

For the 3rd round interaction:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec(\pi^l(X \oplus r)W_o)\\
&= {\pi^l}^T \pi^l(X \oplus r)W_o[:t]\\
&= (XW_o \oplus rW_o)[:t]\\
&= XW_o\\
&= f(X).
\end{aligned}$$

For the 4th round interaction:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec((\text{silu}(\pi^l(X \oplus r)W_1) \cdot \pi^l(X \oplus r)W_2)W_3)\\
&= Dec(\pi^l(\text{silu}((XW_1 \oplus rW_1)) \cdot (XW_2 \oplus rW_2))W_3)\\
&= {\pi^l}^T(\pi^l(\text{silu}((XW_1 \oplus rW_1)) \cdot (XW_2 \oplus rW_2))W_3)[:t]\\
&= (\text{silu}(XW_1) \cdot (XW_2)W_3)\\
&= f(X).
\end{aligned}$$

Therefore, the prefill stage returns the equivalent inference output. □

Then, we consider the generation stage.

Proof. For the 1st round interaction, take the computation of q as an example:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec((X + \alpha)W_q)\\
&= XW_q + \alpha W_q - \alpha W_q\\
&= XW_q\\
&= f(X).
\end{aligned}$$

k, v same as above. For the 2nd round interaction:

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec(Attn(Q\pi^r, K\pi^r, V\pi^r))\\
&= Dec(\text{softmax}\left(\frac{q\pi^r {\pi^r}^T [K,k]^T}{\sqrt{d_k}}\right)[V,v]\pi^r)\\
&= Dec(\text{softmax}\left(\frac{q[K,k]^T}{\sqrt{d_k}}\right)[V,v]\pi^r)\\
&= \text{softmax}\left(\frac{q[K,k]^T}{\sqrt{d_k}}\right)[V,v]\pi^r {\pi^r}^T\\
&= \text{softmax}\left(\frac{q[K,k]^T}{\sqrt{d_k}}\right)[V,v]\\
&= f(X).
\end{aligned}$$

The 3rd round interaction is the same as the 2nd round. For the 4th round interaction,

$$\begin{aligned}
&Dec(f(Enc(X)))\\
&= Dec(MLP(Norm(X + \alpha_l)))\\
&= MLP(Norm(X))\\
&= f(X),
\end{aligned}$$

where we apply the linear-scaling invariance property of the Norm function. Therefore, the generation stage returns the equivalent inference output. □

Since both prefill and generation stages return the equivalent inference output, SCX encoding and decoding scheme has equivalent output as the original inference function.

A.2 Proof of Lemma. 2

Let $x \in \mathbb{R}^d$ denote the input embedding, S the on-cloud encoded KV-cache, and V the vocabulary size. The probability that the cloud can generate the correct next token without the user's key k_u is bounded by:

$$P_{\text{cloud}}[f(x,S) = f(x,S \mid k_u)] \leq \max\left(\frac{1}{d!}, \frac{\exp(\epsilon)}{V}\right).$$

Intuition. For the cloud to generate the correct next token, it must restore the valid KV-cache. However, without access to the user's keys (k_u), the cloud faces two challenges:

(1) Feature-level Permutation Guessing: The KV-cache is encoded using a feature-level permutation matrix (π^r). The cloud must guess the correct permutation to recover the original KV-cache.

(2) Noise Elimination: Laplace noise ($\text{Lap}(\mu = 0, s)$) is added to the embedding. To restore the KV-cache, the cloud must eliminate the influence of this noise, which is difficult without knowing the exact noise parameters.

The cloud has two potential strategies to restore the KV-cache: (1) Guessing the correct feature-level permutation (π^r); (2) Eliminating the noise to infer the correct embedding. Now we prove that both approaches are statistically challenging, as shown below.

Proof. Step 1: Probability of guessing the correct feature-level permutation.

The feature-level permutation matrix π^r defines a random reordering of the d-dimensional embedding. There are $d!$ possible permutations of the embedding features. Without knowledge of the user's key, the cloud can only guess the permutation. The probability of guessing the correct permutation is:

$$P_{\text{perm}} = \frac{1}{d!}.$$

For large embeddings (e.g., $d = 1,000$ or $d = 8,000$), $d!$ is astronomically large, making P_{perm} negligibly small.

Step 2: Probability of eliminating the influence of Laplace noise.

To eliminate the Laplace noise and infer the correct embedding, the cloud needs to identify the original embedding values from their noisy counterparts. Recall that the Laplace noise added to the embedding is drawn from $\text{Lap}(\mu = 0, s)$, where $s = \Delta f / \epsilon$.

Differential privacy guarantees that the probability of distinguishing two embeddings x and x' based on their noisy versions is bounded by:

$$\frac{\Pr[\text{Lap}(x)]}{\Pr[\text{Lap}(x')]} \leq \exp(\epsilon).$$

For the cloud to infer the embedding and generate the correct next token, it must match the noisy embedding to the correct token in the vocabulary. There are V possible tokens in the vocabulary. The probability of successfully eliminating the influence of noise and inferring the correct token is therefore:

$$P_{\text{noise}} = \frac{\exp(\epsilon)}{V}.$$

Step 3: Combining the two probabilities.

To restore the KV-cache and generate the correct next token, the cloud must succeed in either guessing the correct feature-level permutation or eliminating the influence of noise. The overall probability of success is therefore bounded by the maximal of the two independent probabilities:

$$P_{\text{cloud}} \leq \max\left(P_{\text{perm}}, P_{\text{noise}}\right).$$

Substituting the expressions for P_{perm} and P_{noise}:

$$P_{\text{cloud}} \leq \max\left(\frac{1}{d!}, \frac{\exp(\epsilon)}{V}\right).$$

□

This proof shows that the probability of the cloud restoring the KV-cache and generating the correct next token without access to the user's key is bounded by the maximal of solving the two challenges: guessing the correct feature-level permutation ($1/d!$) or eliminating the Laplace noise ($\exp(\epsilon)/V$). For practical values of d, V, and ϵ, this probability is vanishingly small, ensuring the statelessness of SCX.

B DETAILED EXPERIMENTAL SETUP

The prompt we used for LLM-assisted recovery in Sec. 7.3: I have used a tokenizer to tokenize normal sentences into tokens and randomly permuted the order of tokens. Please try to recover the original sentence as much as possible based on the given randomly permuted tokens. Note that only output the recovered sentence, and do not output anything else. Below is the list of tokens: