YOU CAN'T READ MY MIND: NON-INTERACTIVE SECURE AUTOREGRESSIVE LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) excel at autoregressive generation but typically require server-side deployment, raising the risk of sensitive-data leakage. Fully Homomorphic Encryption (FHE) enables secure LLM inference directly over ciphertexts. However, autoregressive inference under FHE remains underexplored and faces challenges in both efficiency and generation quality. Token-by-token generation requires expensive vocabulary-wide operations, while approximate token selection over vocabulary introduces semantic errors and unintended mixtures of token embeddings. These operations recur at every decoding step, making their computational costs and approximation errors a bottleneck to a practical inference under FHE. To address these challenges, we propose **TELEX**, a non-interactive framework for secure autoregressive LLM inference. We introduce a novel inference flow: the server performs reasoning over ciphertexts, while a lightweight client model generates the answer locally with the returned reasoning latents. This design keeps vocabulary-dependent operations outside FHE evaluation. Fixed-length reasoning in encrypted latent space further reduces encrypted decoding from an average of 26.7 steps to 5 on GSM8K-Aug. To accelerate the decoding stage under FHE, we introduce a layout-compatible KV-cache decoding algorithm that efficiently reuses prefill KV-cache without packing-format conversion. We implement TELEX as an end-to-end system that achieves near-lossless generation quality and substantial efficiency gains. TELEX achieves an **85.91**$\times$ per-step decoding speedup compared to MOAI. On a single H200 GPU, we report an amortized time of 49.9 seconds per query for decode stage and an end-to-end time of 14.8 minutes per query. TELEX is the first open-source system to support end-to-end, non-interactive autoregressive LLM inference under FHE.

1 INTRODUCTION

Large Language Models (LLMs) (OpenAI, 2024; Anthropic, 2024; Google, 2024) have rapidly advanced the state of the art across a wide range of generative tasks. These models commonly generate text autoregressively, predicting one token at a time conditioned on preceding tokens. Due to their substantial computational and memory requirements, LLMs are typically deployed on the server side, which easily lead to privacy information leakage (Lam et al., 2021; Fan et al., 2023).

To address the privacy leakage concerns, Fully Homomorphic Encryption (FHE) has emerged as a promising cryptographic primitive for secure LLM inference, as it enables computation directly over encrypted data. However, its substantial computational overhead remains a major obstacle to practical deployment. Autoregressive LLM inference typically consists of two stages: prefill and decode. The prefill stage processes the entire input sequence, whereas the decode stage generates output token-by-token. Prior works have explored FHE-based systems on non-autoregressive models (Moon et al., 2025; Park et al., 2025; Zimerman et al., 2023; 2024; Rovida & Leporati, 2024; Lim et al., 2025), such as the BERT family, or on a prefill stage of an autoregressive model (Zhang et al., 2026a; De Castro et al., 2025; Yang et al., 2025b; Zhang et al., 2024). Recent works extend inference to the decoding stage (Park et al., 2026; Yu et al., 2026b;a). Sylph (Park et al., 2026) and Rossetta (Yu et al., 2026a) reduce nonlinear-operation costs, while Cachemir (Yu et al., 2026b) results in suboptimal prefill efficiency. However, all of the above leave the vocabulary-dependent bottleneck unresolved.

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

Under autoregressive decoding, the output hidden state is projected onto the vocabulary, a token is discretely selected, and its ID directly indexes the embedding table for the next decoding step. However, neither discrete token selection nor indexing is naturally compatible with FHE. Under the CKKS (Cheon et al., 2017) scheme commonly used in Transformer inference, nonlinear selection operations such as Argmax are implemented via homomorphic polynomial approximations over the entire vocabulary. The resulting approximation error makes it difficult to obtain an exact encrypted one-hot vector corresponding to a discrete token ID. Meanwhile, embedding retrieval cannot be performed via direct indexing and instead be typically realized as a dense vector-matrix multiplication over the full vocabulary embedding table. (Kim et al., 2024; Avitan et al., 2025)

This creates two fundamental challenges. First, large vocabularies in modern LLMs make vocabulary projection and embedding retrieval slow by incurring vocabulary-scale homomorphic computation. Second, approximate token selection error corrupts generation quality by both selecting wrong token leading to semantically wrong answer, and input mixture of unrelated token embedding which results in unstable or collapsed generation (Lim et al., 2026; Lin et al., 2024; Rho et al., 2025; Zhang et al., 2026b). Appendix A demonstrates both failure modes of approximate token selection with the Argmax algorithm in NEXUS (Zhang et al., 2024). These vocabulary-dependent operations are repeated for every decoding step and therefore become a critical bottleneck for end-to-end secure autoregressive LLM inference.

To address this challenge, we introduce **TELEX**, an FHE-based non-interactive secure autoregressive LLM inference framework. As shown in Figure 1(a), we delegate heavy and complex reasoning to server to compute under FHE. The encrypted reasoning latents are then returned to the client. The client uses a light-weight model to generate the answer with the decrypted reasoning latents.

To achieve this, we propose an **FHE-friendly latent reasoning transfer** method in Figure 1(c), enabling a lightweight client model to recover the server's performance using the transferred reasoning latents. This design allows for non-interactive secure autoregressive inference, where the server performs multiple steps entirely within the encrypted latent space, removing vocabulary-dependent operations from the homomorphic critical path.

To further mitigate the computational bottleneck in the decode stage, we propose a **layout-compatible key–value (KV) cache** in Figure 1(e). This mechanism exploits the structure of online softmax to seamlessly combine MOAI's state-of-the-art column-packed KV cache with our row-packed representation, preserving prefill efficiency while enabling efficient KV cache decoding under FHE.

To the best of our knowledge, TELEX is the first open-source end-to-end system for non-interactive secure autoregressive LLM inference under FHE. The anonymized source code is available here[1].

In summary, our contributions are as follows:

- **Efficient non-inteactive secure autoregressive LLM inference.** By eliminating vocabulary-dependent operations from the FHE path and reducing encrypted decoding steps from 26.7 to 5, TELEX achieves a **21.9**$\sim$**22.7**$\times$ decoding speedup while retaining at least 99% of the server-side model's task performance under ciphertext inference.
- **FHE-optimized layout-compatible KV cache.** We propose an FHE-based KV-cache decoding method that preserves the optimized prefill efficiency without layout conversion. By efficiently utilizing the KV cache under FHE, TELEX delivers an **85.91**$\times$ speedup over MOAI for a single decoding step.
- **End-to-end CPU/GPU implementation.** We implement TELEX end-to-end on both CPU and GPU platforms and evaluate the inference pipeline. On an H200 GPU running SIM-CoT-Llama-3.2-3B, we achieve an amortized decoding latency of **49.9 seconds per query** and an end-to-end latency of **14.8 minutes per query**, including prefill stage.

1.1 Related Work

Secure Transformer and LLM Inference Existing secure Transformer and LLM inference generally follow either an interactive or a non-interactive paradigm. Interactive methods commonly

[1] https://anonymous.4open.science/r/TELEX-C3DC

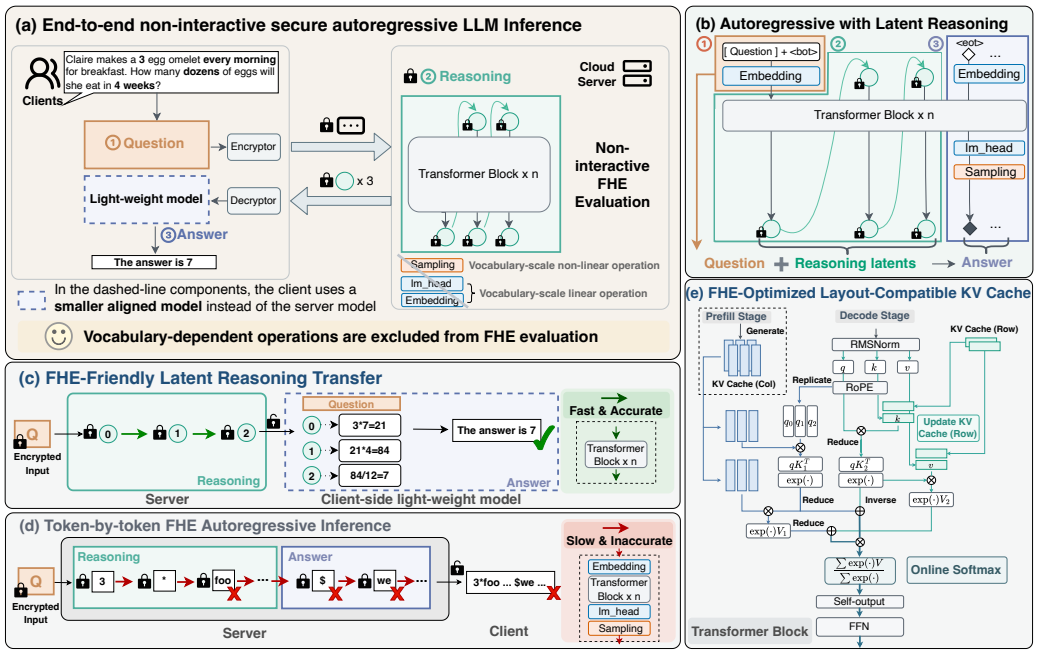


Figure 1: **Overview of TELEX.** (a) End-to-end non-interactive secure autoregressive LLM inference. (b) Latent reasoning decouples reasoning from token generation. (c) FHE-friendly latent reasoning transfer enables a lightweight client to generate answers from server reasoning latents, bypassing costly vocabulary-dependent FHE operations present in (d)'s token-by-token decoding approach. (e) Optimized Layout-compatible KV cache for efficient FHE decoding.

employ MPC, optionally combined with HE, to reduce computation at the communication cost. Early works such as THE-X (Chen et al., 2022) demonstrated secure Transformer inference, followed by numerous protocol and model optimizations (Hao et al., 2022; Hou et al., 2026; Liang et al., 2024; Pang et al., 2024; Wu et al., 2024; Lu et al., 2023; Dong et al., 2025; Jha & Reagen, 2025; Zeng et al., 2025; Zhang et al., 2025; Xu et al., 2025a). However, such approaches can incur substantial communication overhead and require data owners to remain online (Rho et al., 2024).

These limitations motivate non-interactive inference based on FHE. Prior FHE-based Transformer methods primarily target non-autoregressive models or the prefill stage of autoregressive LLMs (Moon et al., 2025; Park et al., 2025; Zimerman et al., 2023; 2024; Rovida & Leporati, 2024; Lim et al., 2025; Zhang et al., 2026a; 2024; De Castro et al., 2025; Yang et al., 2025b), with optimizations for ciphertext packing, homomorphic matrix operations, and non-linear evaluations. Recent works extend FHE-based LLM inference to autoregressive decoding (Park et al., 2026; Yu et al., 2026b). Sylph (Park et al., 2026) reduces non-linear evaluation costs through outlier mitigation, while ROSETTA (Yu et al., 2026a) combines CKKS and TFHE for efficient non-linear evaluation. Cachemir (Yu et al., 2026b) trades off prefill efficiency for decoding optimization. However, vocabulary-dependent operations required at each decoding step remain largely unaddressed by these approaches. Modern LLMs typically operate over large vocabularies; for example, Llama-3.2 contains 128,256 tokens (Meta, 2024). Consequently, vocabulary projection and embedding retrieval involve large vocabulary-wide linear operations, while encrypted token selection requires nonlinear approximation over the same vocabulary space.

Latent reasoning, continuous representations Early work internalizes explicit reasoning through distillation (Deng et al., 2023) and progressive removal of reasoning tokens (Deng et al., 2024). Other approaches explore looped computation (Giannou et al., 2023; Zhu et al., 2025; Jeddi et al., 2026), auxiliary soft thoughts (Xu et al., 2025b), and autoregressive latent compression (Tan et al., 2026). Some work explore autoregressive latent reasoning (Xu et al., 2025b; Tan et al., 2026). Coconut chains continuous hidden states as input embeddings, bypassing discrete token interface (Hao et al., 2025). CODI advances this direction by trajectory-level distillation (Shen

et al., 2025). SIM-CoT (Wei et al., 2026) introduces step-level supervision and achieves accuracy comparable to supervised chain-of-thought (CoT) method on GSM8k-Aug (Deng et al., 2024) with Llama-3.2-1B.

2 PRELIMINARY

Notation. We write $[n] = \{0, \ldots, n-1\}$ and use bold lowercase letters for row vectors and uppercase letters for matrices. For $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{v}^\top$ denotes its transpose. The symbols $\otimes$ and $\|$ denote elementwise multiplication and column concatenation, respectively. For $k \in \mathbb{Z}$, cyclic rotation is defined by $\mathrm{Rot}_k(\mathbf{v}) = [v_{(i+k) \bmod n}]_{i\in[n]}$, where positive and negative k specify left and right rotations. For $A = (a_{ij}) \in \mathbb{R}^{m\times d}$, $A_{:,j}$ denotes its j-th column. When $m = d$, its k-th cyclic diagonal is $\mathrm{Diag}_k(A) = [a_{i,(i+k) \bmod m}]_{i\in[m]}, k \in [m]$.

2.1 FULLY HOMOMORPHIC ENCRYPTION

We use CKKS (Cheon et al., 2017) to evaluate approximate arithmetic on encrypted vectors. With polynomial degree N, a power of two, CKKS provides $S = N/2$ complex slots per ciphertext. Packing values into these slots enables SIMD evaluation. Let $ct_{\mathbf{u}} = \mathrm{Enc}(\mathbf{u})$ and $ct_{\mathbf{v}} = \mathrm{Enc}(\mathbf{v})$ for $\mathbf{u}, \mathbf{v} \in \mathbb{C}^S$. The operation in CKKS scheme, with keys and encoding details omitted, are as follows:

$$\mathrm{Dec}(\mathrm{Add}(ct_{\mathbf{u}}, ct_{\mathbf{v}})) = \mathbf{u} + \mathbf{v}, \quad \mathrm{Dec}(\mathrm{Mult}(ct_{\mathbf{u}}, ct_{\mathbf{v}})) = \mathbf{u} \otimes \mathbf{v},$$
$$\mathrm{Dec}(\mathrm{PlainMult}(ct_{\mathbf{u}}, \mathbf{v})) = \mathbf{u} \otimes \mathbf{v}, \quad \mathrm{Dec}(\mathrm{Rot}_k^{\mathrm{HE}}(ct_{\mathbf{u}})) = \mathrm{Rot}_k(\mathbf{u}).$$

Here, PlainMult uses an encoded plaintext operand. At level ℓ, the ciphertext modulus is $Q_\ell = \prod_{i=0}^{\ell} q_i$, where the q_i are distinct primes. Fresh ciphertexts start at level L. Rescaling controls the scale by reducing both the scale and modulus, removing one prime per step. Bootstrapping refreshes ciphertexts to support further computation (Cheon et al., 2018).

2.2 LATENT REASONING

Latent reasoning represents intermediate computation as hidden states before generating an answer in tokens. For a prompt $x = (x_i)_{i\in[T]}$, we append the begin-of-thought token `<bot>` to enter the latent phase. The initial prefix $U^{(0)} \in \mathbb{R}^{(T+1)\times d}$ is $U^{(0)} = [e_\theta(x_i)]_{i\in[T]} \oplus e_\theta(\texttt{<bot>})$. Let $e_\theta : \mathcal{V} \to \mathbb{R}^d$ embed tokens from vocabulary $\mathcal{V}$, and let $H_\theta(U) \in \mathbb{R}^d$ return the last-layer hidden state at the final position of a prefix U. K latent transitions are written as $\mathbf{z}_k = H_\theta(U^{(k)}), U^{(k+1)} = U^{(k)} \oplus \mathbf{z}_k$, $k \in [K]$, where $\oplus$ appends a row along the sequence dimension. Each latent state is fed back as the next input embedding. After K latent steps, the end-of-thought token `<eot>` is appended to form the answer prefix $U^{\mathrm{ans}} = U^{(K)} \oplus e_\theta(\texttt{<eot>})$. The model then generates answer tokens autoregressively: $\mathbf{h}_t = H_\theta(U^{\mathrm{ans}} \oplus e_\theta(a_{<t}))$, $p_\theta(a_t \mid U^{\mathrm{ans}}, a_{<t}) = \mathrm{softmax}(\mathbf{h}_t {W_o}^\top)_{a_t}$.

3 NON-INTERACTIVE SECURE AUTOREGRESSIVE LLM INFERENCE

We first introduce the problem setting and overview of TELEX, then describe our FHE-Friendly latent reasoning transfer method and the end-to-end inference protocol.

3.1 PROBLEM SETUP AND FRAMEWORK OVERVIEW

We consider a client holding a private question and a lightweight answer generator, and a server hosting a large latent-reasoning model obtained by post-training a pretrained LLM (Hao et al., 2025; Shen et al., 2025; Wei et al., 2026). As illustrated in Figure 1(b), latent-reasoning models recurrently feed hidden states back into the model without converting into tokens, instead using them as reasoning latents. TELEX exploits this structure to restrict server-side FHE evaluation to latent reasoning, while the client generates the answer using an aligned lightweight model, avoiding vocabulary-dependent operations under FHE. Since answer decoding in the original latent-reasoning model is conditioned on the latent trajectory, we align the client model to condition answer generation on the question and the returned latents, thereby recovering the performance of the server-side model.

During inference, the client encrypts locally computed question embeddings and sends them to the server, which evaluates a fixed number of steps under FHE non-interactively and returns the encrypted latents. The client decrypts the latents and generates the final answer with the aligned lightweight model.

3.2 FHE-FRIENDLY LATENT REASONING TRANSFER

We present our methodology for latent reasoning transfer in Figure 1(c). First, we detail the cross-model bridging and dual-path training, which enable the client model to effectively consume server-side latents. Second, we design a dynamic selector that adaptively routes queries between step-by-step expansion and direct readout.

Learning to consume latent reasoning. To make server-side reasoning usable by the lightweight client, we learn a mapping between their representation spaces. Following the continuous communication interface of Interlat (Du et al., 2026), an adapter maps the transmitted trajectory into a client-side soft prefix: $Z = [\mathbf{z}_k]_{k\in[K]} \in \mathbb{R}^{K\times d}, P = A_\phi(Z) \in \mathbb{R}^{K\times d_c}$, where d_c is the client embedding dimension. The prefix supplies reasoning context alongside the question, yielding

$$p_{\eta,\phi}(y \mid x, Z) = \prod_{t\in[|y|]} p_\eta(y_t \mid x, P, y_{<t}), \tag{1}$$

where η and ϕ denote the client and adapter, respectively.

The client can consume the transmitted latent trajectory in two ways: directly read out an answer, or first expand the latents into explicit reasoning. Specifically, Hidden-Prefix Transfer (HP) learns $p_{\mathrm{HP}}(a \mid x, Z)$, whereas Latent-to-CoT Transfer (L2C) yields an explicit reasoning chain r and learns $p_{\mathrm{L2C}}(r, a \mid x, Z)$. Explicit expansion provides a step-by-step route for the client to use the reasoning latent, but also makes the answer depend on its own intermediate generated tokens. Consequently, errors introduced during expansion may accumulate and degrade answer accuracy. To mitigate this risk, we retain HP as a complementary direct readout path and adaptively selecting between the two routes for each question. Given a reference CoT r and answer a, the branches receive target-specific supervision:

$$y^{\mathrm{HP}} = a, \qquad y^{\mathrm{L2C}} = (r, a), \quad \mathcal{L}_{\mathrm{CE}}^{(b)} = -\frac{1}{|y^{(b)}|}\log p_b(y^{(b)} \mid x, Z), \tag{2}$$

where $b \in \{\mathrm{HP}, \mathrm{L2C}\}$ and p_b denotes the corresponding conditional model.

To encourage generation to benefit specifically from the matched trajectory, L2C additionally learns to prefer it over mismatched, zero, and absent latent conditions. We consider three control conditions indexed by $j \in [3]$: mismatched latents ($j = 0$), zero latents ($j = 1$), and removal of the latent interface ($j = 2$). For the same reference sequence $y = (r, a)$, let s_+ denote its mean token log-probability under matched latents, and s_j the corresponding score under the j-th control condition. We minimize $\mathcal{L}_{\mathrm{pref},j} = -\log\sigma(\beta_{\mathrm{pref}}(s_+ - s_j))$, where σ is the sigmoid function and $\beta_{\mathrm{pref}} > 0$ controls preference strength. A complementary separation loss encourages distinct predictions under matched and mismatched trajectories $\mathcal{L}_{\mathrm{sep}} = [\mu_{\mathrm{sep}} - \overline{D}_{\mathrm{JS}}]_+$, where $[u]_+ = \max(0, u)$ and $\overline{D}_{\mathrm{JS}}$ is the Jensen–Shannon divergence (Lin, 1991) between the two next-token distributions, averaged over reference target positions. The mismatched distribution is held fixed in this term. The final L2C objective is

$$\mathcal{L}_{\mathrm{L2C}} = \mathcal{L}_{\mathrm{CE}}^{(\mathrm{L2C})} + \lambda_{\mathrm{sep}}\mathcal{L}_{\mathrm{sep}} + \lambda_{\mathrm{pref}}\sum_{j\in[3]}\mathcal{L}_{\mathrm{pref},j}. \tag{3}$$

Task supervision rewards correct generation, separation encourages sensitivity to latents, and preference ties that sensitivity to the target response. Similarly, HP is supervised on final answer only with mismatch/zero preferences and separation losses.

Dynamic selection for explicit expansion. Rather than always choosing the L2C branch, the selector estimates whether explicit expansion improves the answer over direct readout. Let $\chi_{\mathrm{HP}}, \chi_{\mathrm{L2C}} \in \{0, 1\}$ indicate candidate correctness, and let $\mathbf{f}$ summarize the generated texts, confidence, and decoding features. The conditional accuracy gain from choosing L2C instead of HP is $\Delta(\mathbf{f}) = \Pr(\chi_{\mathrm{L2C}} = 1, \chi_{\mathrm{HP}} = 0 \mid \mathbf{f}) - \Pr(\chi_{\mathrm{HP}} = 1, \chi_{\mathrm{L2C}} = 0 \mid \mathbf{f})$.

Algorithm 1 TELEX End-to-end Inference given Private Question $x = (x_i)_{i\in[T]}$, Latent Budget K.

(1) Client:
1: $U^{(0)} \leftarrow [e_\theta(x_i)]_{i\in[T]} \oplus e_\theta(\texttt{<bot>})$
2: $ct_{\mathbf{U}}^{(0)} \leftarrow \mathrm{Enc}(U^{(0)})$
3: Send $ct_{\mathbf{U}}^{(0)}$ to the server

(2) Server: encrypted prefill
1: $ct_{\mathbf{U}}^{(0)}$ from Client.
2: $(ct_{\mathbf{z_0}}, ct_{\mathcal{C}}^{(0)}) \leftarrow \mathsf{Eval}(\mathsf{Prefill}, ct_{\mathbf{U}}^{(0)})$

(3) Server: latent decoding
3: **for** $k = 1, \ldots, K-1$ **do**
4: $\quad \mathsf{Eval}(\mathsf{Decode}, ct_{\mathbf{z_{k-1}}}, ct_{\mathcal{C}}^{(k-1)})$
5: $\quad$ Store result as $(ct_{\mathbf{z_k}}, ct_{\mathcal{C}}^{(k)})$
6: **end for**
7: Return $(ct_{\mathbf{z_k}})_{k\in[K]}$ to the client

(4) Client after server computation:
4: $\widetilde{\mathbf{z}_k} \leftarrow \mathrm{Dec}(ct_{\mathbf{z_k}})$ for $k \in [K]$
5: Generate a locally, conditioned on x and $(\widetilde{\mathbf{z}_k})_{k\in[K]}$

Ensure: Answer a

A learned estimate Δ_ψ determines the returned answer $\hat{a} = \hat{a}_{\mathrm{L2C}}$ if $\Delta_\psi(\mathbf{f}) > \tau$, and $\hat{a}_{\mathrm{HP}}$ otherwise, where τ is calibrated on development data. The selector thus balances the benefit of explicit expansion against the reliability of direct latent readout on each question.

3.3 End-to-End Inference Protocol

After one-time CKKS key setup, the client retains the secret key and provides the public and evaluation keys to the server. Algorithm 1 summarizes the end-to-end protocol. Here, $\mathcal{C}^{(k)}$ is the encrypted layer-wise KV cache after latent step k; Prefill initializes the cache, and Decode extends it while advancing the latent state. Tildes denote decrypted CKKS-approximate states. Notably, the online protocol executes in a single client–server round trip, achieving the desired non-interactive setting.

4 Efficient FHE decoding with layout-compatible KV cache

In this section, we first present the packing method and vector-matrix multiplication under ciphertext of TELEX. Then we introduce our layout-compatible KV cache method in Figure 1(e), followed by the details of non-linear function approximation.

4.1 Packing Method and Vector–Matrix Multiplication

Packing & Interleaved batching. We retain column packing for the prefill KV cache (Zhang et al., 2026a; Halevi & Shoup, 2014) and use blockwise row packing for current-step vectors. Let n denote the logical capacity per batch lane and $B = S/n$ the number of lanes. A hidden state of dimension d is partitioned into feature blocks of length n, whereas a per-head vector of dimension $d_h \leq n$ occupies one block. Each ciphertext interleaves corresponding feature blocks or cache columns across the B examples. All ciphertexts use the same batch ordering and stride B. Rotations by multiples of B rotate the logical coordinates within each lane without mixing examples (Zhang et al., 2026a; Yang et al., 2025b). Formal definitions are provided in Appendix B.

Vector–matrix multiplication We compute the query–key product directly from the column-packed key cache using the full-replication primitive of Halevi & Shoup (2014). Each query coordinate is replicated within its batch lane and multiplied with the corresponding encrypted key-cache column. Summing these ciphertext–ciphertext (ct–ct) products yields the query–key product without transposing or repacking the cache. The replication primitive and the column-packed multiplication procedure are formalized in Appendix C.

We evaluate linear projections using baby-step giant-step (BSGS) ciphertext–plaintext (ct–pt) multiplication on $n \times n$ plaintext weight blocks (Halevi & Shoup, 2021), preserving the (n, B) layout. Baby-step rotations of each input block are reused across diagonal groups and output blocks. For each output block, contributions from all input blocks are summed within each diagonal group before its giant-step rotation; the rotated sums are then added. We select the split according to the input and output ciphertext counts. Pre-rotated cyclic diagonals are repeated across lanes and encoded offline. The algorithm and rotation-count analysis are provided in Appendix D.

4.2 LAYOUT-COMPATIBLE KV CACHE FOR EFFICIENT FHE DECODING

As shown in Figure 1(e), we handle the column-packed prefill cache and row-packed decode KV through separate attention paths, merging their unnormalized statistics before normalization via online softmax (Dao et al., 2022; Milakov & Gimelshein, 2018).

Online softmax with shared slot geometry. We describe one attention head with dimension d_h and m prefill positions, with positional transformations already applied to queries and keys. For each batch lane $r \in [B]$, the query, keys, and values are row vectors in $\mathbb{R}^{d_h}$, and the prefill-cache matrices satisfy $C_K^{(r)}, C_V^{(r)} \in \mathbb{R}^{m \times d_h}$.

For a single prefill-sequence block, the cache columns and per-head vectors use a shared capacity $n \geq \max(m, d_h)$, with $nB = S$. They are padded to length n along the sequence and feature axes when necessary. Slot $tB + r$ represents sequence position t in a prefill-cache column and feature coordinate t in a per-head row-packed vector.

Let $\mathcal{P}$ and $\mathcal{D}$ denote the disjoint sets of prefill and visible decode positions. Omitting batch superscripts, define $s_i = \mathbf{q}\mathbf{k}_i^\top / \sqrt{d_h}$. For any subset $\mathcal{A}$ of visible positions, its unnormalized attention statistics are $\mathbf{u}_\mathcal{A} = \sum_{i \in \mathcal{A}} \exp(s_i)\mathbf{v}_i, \omega_\mathcal{A} = \sum_{i \in \mathcal{A}} \exp(s_i)$. These statistics are additive across disjoint subsets, giving $\mathbf{o} = (\mathbf{u}_\mathcal{P} + \mathbf{u}_\mathcal{D})/(\omega_\mathcal{P} + \omega_\mathcal{D})$. The accumulation is performed separately for each query; only KV states, not query-dependent attention statistics, persist across decoding steps.

Two compatible computation paths for KV cache. For the prefill-cache path, $\mathrm{Replicate}_{n,B}$ broadcasts individual query coordinates for multiplication with the corresponding key columns, leaving the original row-packed query unchanged. After exponentiation, each value-column reduction produces one numerator coordinate, broadcast across all n logical positions within each batch lane. Coordinate masks place these coordinates into their corresponding row-packed feature positions, while the denominator is broadcast within each batch lane. For the decode-KV path, the query and each stored key are multiplied directly in row packing. Feature reduction produces a broadcast score whose exponential is aligned with the row-packed value vector. Its weighted value and weight therefore update the same numerator and denominator accumulators without converting the decode KV pairs. The details of layout-compatible KV cache via online softmax are in Appendix E.

4.3 NON-LINEAR FUNCTION APPROXIMATION

We follow previous FHE implementations (Zhang et al., 2026a; 2024) for the repeated-squaring exponential $E(z) = (1 + z/128)^{128}$ and geometric-series reciprocal used in softmax. For RMSNorm, we adopt its inverse-square-root routine, using affine initialization followed by four Newton iterations and two Goldschmidt refinement iterations (Goldschmidt, 1964; Qu & Xu, 2022).

For SiLU, Park et al. (2026) report a degree-31 polynomial approximation and MOAI (Zhang et al., 2026a) uses a degree-63 polynomial approximation in their implementation. We retain the exact linear component $x/2$ and approximate the even remainder $\mathrm{SiLU}(x) - x/2$. Under this symmetry constraint, degree bounds 31 and 30 define the same polynomial family, since all odd-power coefficients above degree one vanish. Our approximation therefore has degree at most 30. Its coefficients are fitted by weighted least squares in a Chebyshev basis and an auxiliary uniform grid, then converted to the power basis for balanced-tree evaluation in 6 CKKS multiplicative levels.

5 EXPERIMENTS

We evaluate three questions: **RQ1:** Does transferring server latents improve client-side reasoning? **RQ2:** How much of the server's reasoning accuracy can the client recover under FHE? **RQ3:** How do latent reasoning and KV caching improve end-to-end inference efficiency? Finally, we quantify the decoding speedups achieved through efficient KV cache utilization.

Models and datasets. We use SIM-CoT-Llama-3.2-3B as the server and Llama-3.2-1B (Meta, 2024) as the client. We also evaluate Qwen3-0.6B (Yang et al., 2025a) as the client to demonstrate cross-model performance. Following SIM-CoT (Wei et al., 2026), we train and evaluate on GSM8K-Aug (Deng et al., 2024) as our in-domain dataset, which is an expanded version of GSM8K (Cobbe

Table 1: **Plaintext reasoning accuracy (%) on in-domain and out-of-domain benchmarks.** All methods use a 1B client; TELEX receives 6 latent states from the 3B server. Its selector chooses between Hidden-Prefix (HP) and Latent-to-CoT (L2C) outputs. Oracle counts a prediction as correct whenever either branch produces the correct answer.

Method	In-Domain	Out-of-Domain			
	GSM8K-Aug	GSM-Hard	MultiArith	SVAMP	OOD Avg.
No CoT	28.8	6.3	50.3	26.7	27.8
SFT-CoT	58.4	13.9	96.7	65.7	58.8
SIM-CoT	56.1	12.7	96.2	61.5	56.8
TELEX	**62.5**	**16.3**	**98.9**	**69.7**	**61.6**
TELEX(Oracle)	67.6	17.5	99.4	73.0	63.3

Table 2: **Recovery of server reasoning accuracy (%).** Recovery rate measures the average of GSM8K-Aug and OOD Avg. accuracies relative to the server model.

Method	GSM8K-Aug	OOD Avg.	Recovery rate
Server: SIM-CoT-Llama-3.2-3B	62.1	63.2	100%
Client: Llama-3.2-1B			
TELEX under Plaintext	62.5	61.6	99.0%
TELEX under FHE	62.5	61.5	99.0%
Client: Qwen3-0.6B			
TELEX under Plaintext	61.7	67.4	103.0%
TELEX under FHE	61.7	67.2	102.9%

et al., 2021). For out-of-domain (OOD) evaluation, we test on GSM-Hard (Gao et al., 2023), MultiArith (Roy & Roth, 2015), and SVAMP (Patel et al., 2021). OOD Avg. is the unweighted mean across the three OOD benchmarks. The selector is fitted and calibrated on GSM8K-Aug splitdev and applied unchanged to OOD data.

Implementation. We use CKKS with $N = 2^{16}$ and 128-bit security. Our CPU implementation uses Microsoft SEAL (Microsoft Research, 2023) and FHE-MP-CNN (Lee et al., 2022) on an AMD EPYC 9755 128-Core Processor; our GPU implementation uses PhantomFHE (Yang et al., 2024) on a single NVIDIA H200. More CKKS parameters and details appear in Appendix F.1.

5.1 Main Results

Client-side gains from server latents (RQ1). Table 1 compares TELEX with 1B client baselines that receive no server latents. Here, No-CoT is supervised fine-tuning (SFT) on direct answers only, SFT-CoT is supervised fine-tuning on CoT-annotated data, SIM-CoT (Wei et al., 2026) is a latent reasoning model. Conditioned on 6 latents from the 3B server, TELEX achieves 62.5% on GSM8K-Aug, outperforming SFT-CoT and SIM-CoT by 4.1 and 6.4 percentage points, respectively. It also outperforms SFT-CoT and SIM-CoT on all three OOD benchmarks, with an OOD average of 61.6%, compared with 56.8% for SIM-CoT and 58.8% for SFT-CoT. These results demonstrate the benefit of server-to-client latent transfer over the compared client-only reasoning methods, including CoT post-training. Oracle selection over the HP/L2C outputs reaches 67.6% on GSM8K-Aug and 63.3% OOD average, leaving 5.1 and 1.7 percentage points of selection headroom within the existing candidates.

Recovering server reasoning accuracy (RQ2). Table 2 evaluates whether TELEX preserves the server's reasoning capability when answer generation is carried out by smaller clients under FHE. With Llama-3.2-1B, TELEX retains 99.0% of the 3B server's average accuracy, while Qwen3-0.6B achieves a 102.9% recovery rate. Notably, the encrypted pipeline closely matches its plaintext counterpart, with no degradation on GSM8K-Aug while OOD. Avg. decreases by only 0.1 and 0.2 percentage points for Llama-3.2-1B and Qwen3-0.6B, respectively. These results show that TELEX effectively preserves server-level reasoning accuracy even when the transferred latent reasoning while moving answer generation to a smaller client. Appendix F.2 extends TELEX to additional latent

Table 3: **Efficiency ablations of TELEX.** The variants disable latent reasoning, KV caching, or both. # Tokens/Latents reports the total reasoning sequence length, including the first token/latent produced during prefill. Query time includes prefill stage.

Method	# Tokens/Latents	Decoding Time (s)	Query Time (min)
TELEX	6	49.9	14.8
w/o Latent	27.7	1092.6–1134.5	32.6–32.9
w/o KV Cache	6	4288.5	85.4
w/o Both	27.7	24901.2–25739.3	429.0–446.4

Table 4: Amortized time for the first decoding step of the Llama-3.2-3B transformer. Speedups are computed from unrounded times.

Operation	MOAI (s/input)	TELEX (s/input)	Speedup (×)
Input RMSNorm	0.35	0.01	41.3
Q/K/V projection + RoPE	76.40	0.80	95.7
Attention ($QK^\top$ + Softmax + AV)	105.13	3.98	26.4
Attention output projection	9.74	0.15	65.0
Post-attention RMSNorm	0.35	0.01	44.9
Gate/up projection	200.83	1.26	159.4
SiLU + elementwise product	52.57	0.05	1163.6
Down projection + residual	25.87	0.39	67.2
Bootstrapping	373.35	3.20	116.9
Total	**844.61**	**9.83**	**85.91**

reasoning methods, demonstrating its generality across heterogeneous server–client settings. The error analysis in Appendix F.4 (Figure 3) confirms stable multi-step latent reasoning under FHE. Specifically, errors remain zero-centered and narrow after step 3 without growing monotonically. Together with the retained task accuracy, these results validate our system's robustness.

End-to-end efficiency (RQ3). Table 3 quantifies the contributions of latent reasoning and KV caching. TELEX uses 6 latents using 5 decoding steps after prefill and requires 49.9 s for decode stage and 14.8 min per query, including prefill stage. Without latent reasoning, the reasoning sequence grows to 27.7 tokens, corresponding to an average of 26.7 decoding steps, and decoding time increases by 21.9–$22.7\times$; without KV caching, decoding time increases by $85.9\times$. Removing both raises query time to 429.0–446.4 min, yielding a 29.0–$30.2\times$ end-to-end speedup for TELEX. These gains combine a shorter reasoning trajectory with reuse of encrypted KV cache.

Sources of decoding speedup. Table 4 compares TELEX with MOAI (Zhang et al., 2026a) for the first decoding step of the 28-layer Llama-3.2-3B. TELEX reduces amortized computation from 844.61 to 9.83 s/input, an $85.91\times$ speedup. KV reuse avoids recomputing prefix representations, while compact current-token packing reduces ciphertext processing in projections and bootstrapping. Beyond the first step, Figure 4 in Appendix F.5 shows that per-step decoding time grows linearly with the step length, increasing from 9.83 to 10.12 s over five steps. The single-layer breakdown in Appendix F.5.2 (Table 6) further details the operation costs and ciphertext-level schedule.

6 Conclusion

We introduce TELEX, an end-to-end framework for non-interactive secure autoregressive LLM inference. Through a novel FHE-friendly latent reasoning transfer, we successfully eliminate vocabulary-dependent operations from the FHE critical path. To further accelerate the decode stage, we develop a layout-compatible KV cache that preserves prefill efficiency without costly packing-format conversions. TELEX achieves near-lossless generation quality and substantially outperforms the state-of-the-art HE-based Transformer inference work (Zhang et al., 2026a) in decoding efficiency. It reduces amortized decoding time to under 1 minute per query and end-to-end inference in about 15 minutes on a single H200 GPU.

AI USE STATEMENT

In this work, we used generative AI tools for debugging test function inputs, handling the extraction, storage, and loading of plaintext model weights, managing remote code repositories, and remotely monitoring the training process. We have not used generative AI tools for core algorithm design or implementation. Additionally, we used generative AI tools for sentence-level language polishing, but not to draft sections or generate large blocks of text. We have reviewed all AI-assisted work. All AI-assisted code was manually verified to ensure functional correctness. We take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our work, we have provided comprehensive details across the main text, appendix, and supplementary materials. The core design of TELEX, including the FHE-friendly latent reasoning transfer and layout-compatible KV cache, is detailed in Section 3.2 and Section 4.2. Complete mathematical definitions and derivations for packing, vector–matrix muliplication, layout-compatible KV cache algorithm are provided in Appendix B and Appendix E. Hardware configurations, CKKS configurations, and implementation details are documented in Appendix F. To facilitate reproducibility, our anonymized source code is publicly available at `https://anonymous.4open.science/r/TELEX-C3DC`.

REFERENCES

Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. In *Protecting privacy through homomorphic encryption*, pp. 31–62. Springer, 2022.

Anthropic. Claude 3.5 Sonnet, 2024. URL `https://www.anthropic.com/news/claude-3-5-sonnet`.

Matan Avitan, Moran Baruch, Nir Drucker, Itamar Zimerman, and Yoav Goldberg. Efficient decoding methods for language models on encrypted data. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 1777–1794, 2025.

Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 587–617. Springer, 2021.

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. The-x: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 3510–3520, 2022.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409–437. Springer, 2017.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29–May 3, 2018, Proceedings, Part I*, pp. 360–384. Springer, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Leo De Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. Encryptedllm: Privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption. In *ICML*, 2025.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, Wen-Guang Chen, et al. Puma: Secure inference of llama-7b in five minutes. *Security and Safety*, 4:2025014, 2025.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27106–27129, 2026.

Jiani Fan, Wenzhuo Yang, Ziyao Liu, Jiawen Kang, Dusit Niyato, Kwok-Yan Lam, and Hongyang Du. Understanding security in smart city domains from the ant-centric perspective. *IEEE Internet of Things Journal*, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Robert E Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

Google. Our next-generation model: Gemini 1.5, February 2024. URL https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/.

Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pp. 554–571. Springer, 2014.

Shai Halevi and Victor Shoup. Bootstrapping for helib. *Journal of Cryptology*, 34(1):7, 2021.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. *Advances in neural information processing systems*, 35:15718–15731, 2022.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling (COLM)*, 2025.

Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Jiawen Zhang, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. *IEEE Transactions on Dependable and Secure Computing*, 2026.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Nandan Kumar Jha and Brandon Reagen. Entropy-guided attention for private llms. *arXiv preprint arXiv:2501.03489*, 2025.

Jaeyun Kim, Saerom Park, Joohee Lee, and Jung Hee Cheon. Privacy-preserving embedding via look-up table evaluation with fully homomorphic encryption. In *ICML*, pp. 24437–24457, 2024.

Kwok-Yan Lam, Sananda Mitra, Florian Gondesen, and Xun Yi. Ant-centric iot security reference architecture—security-by-design for satellite-enabled smart cities. *IEEE Internet of Things Journal*, 9(8):5895–5908, 2021.

Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pp. 12403–12422. PMLR, 2022.

Zi Liang, Pinghui Wang, Ruofei Zhang, Nuo Xu, Shuo Zhang, Lifeng Xing, Haitao Bai, and Ziyang Zhou. Merge: Fast private text generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19884–19892, 2024.

Lawrence Lim, Vikas Kalagi, Divyakant Agrawal, and Amr El Abbadi. Tricycle: Private transformer inference with tricyclic encodings. *Cryptology ePrint Archive*, 2025.

Lawrence Lim, Jiaming Liu, Vikas Kalagi, Divyakant Agrawal, and Amr El Abbadi. Hyperion: Private token sampling with homomorphic encryption. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14150–14159, 2026.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

Zicheng Lin, Tian Liang, Jiahao Xu, Qiuzhi Lin, Xing Wang, Ruilin Luo, Chufan Shi, Siheng Li, Yujiu Yang, and Zhaopeng Tu. Critical tokens matter: Token-level contrastive estimation enhances llm's reasoning capability. *arXiv preprint arXiv:2411.19943*, 2024.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*, 2023.

Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`, 2024. Meta AI.

Microsoft Research. Microsoft SEAL (release 4.1). `https://github.com/microsoft/SEAL`, January 2023. Microsoft Research, Redmond, WA.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. Thor: Secure transformer inference with homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3765–3779, 2025.

OpenAI. Hello GPT-4o, May 2024. URL `https://openai.com/index/hello-gpt-4o/`.

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

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11090–11111, 2025.

Jaiyoung Park, Sejin Park, Jai Hyun Park, Jung Ho Ahn, Jung Hee Cheon, Guillaume Hanrot, Jung Woo Kim, Minje Park, and Damien Stehlé. Scaling up fhe-based privacy-preserving ml: Higher throughput, longer inputs for llama-3-8b, 2026. URL https://arxiv.org/abs/2601.18511.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Hongyuan Qu and Guangwu Xu. Improvements of homomorphic evaluation of inverse square root. *Available at SSRN 4258571*, 2022.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest K Ryu, and Jung Hee Cheon. Encryption-friendly llm architecture. *arXiv preprint arXiv:2410.02486*, 2024.

Donghwan Rho, Sieun Seo, Hyewon Sung, Chohong Min, and Ernest K Ryu. Traveling salesman-based token ordering improves stability in homomorphically encrypted language models. *arXiv preprint arXiv:2510.12343*, 2025.

Lorenzo Rovida and Alberto Leporati. Transformer-based language models and homomorphic encryption: An intersection with bert-tiny. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*, pp. 3–13, 2024.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of llm reasoning chains. *Advances in Neural Information Processing Systems*, 38:4646–4668, 2026.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. Sim-cot: Supervised implicit chain-of-thought. In *International Conference on Learning Representations*, volume 2026, pp. 56721–56742, 2026.

Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, Yinggui Wang, and Lei Wang. Ditto: Quantization-aware secure inference of transformers upon mpc. *arXiv preprint arXiv:2405.05525*, 2024.

Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. Breaking the layer barrier: Remodeling private transformer inference with hybrid {CKKS} and {MPC}. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2653–2672, 2025a.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. Phantom: A cuda-accelerated word-wise homomorphic encryption library. *IEEE Trans. Dependable Secur. Comput.*, 21(5):4895–4906, 2024. doi: 10.1109/TDSC.2024.3363900. URL https://doi.org/10.1109/TDSC.2024.3363900.

Linhan Yang, Jingwei Chen, Wangchen Dai, Shuai Wang, Wenyuan Wu, and Yong Feng. Arion: Attention-optimized transformer inference on encrypted data. *Cryptology ePrint Archive*, 2025b.

Jiangrui Yu, Baosheng Zhang, Liang Kong, Lin Ding, Yi Chen, Ye Yu, Mingzhe Zhang, and Meng Li. ROSETTA: Efficient and accurate privacy-preserving LLM decoding via hybrid CKKS/TFHE evaluation. Cryptology ePrint Archive, Paper 2026/2047, 2026a. URL `https://eprint.iacr.org/2026/2047`.

Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. Cachemir: Fully homomorphic encrypted inference of generative large language model with kv cache. *arXiv preprint arXiv:2602.11470*, 2026b.

Wenxuan Zeng, Ye Dong, Jinjin Zhou, Junming Ma, Jin Tan, Runsheng Wang, and Meng Li. Mpcache: Mpc-friendly kv cache eviction for efficient private large language model inference. *arXiv preprint arXiv:2501.06807*, 2025.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. *Cryptology ePrint Archive*, 2024.

Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. Moai: Module-optimizing architecture for non-interactive secure transformer inference. In *International Conference on Learning Representations*, volume 2026, pp. 27927–27953, 2026a.

Yancheng Zhang, Jiaqi Xue, Mengxin Zheng, Mimi Xie, Mingzhe Zhang, Lei Jiang, and Qian Lou. Cipherprune: Efficient and scalable private transformer inference. *arXiv preprint arXiv:2502.16782*, 2025.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *Advances in Neural Information Processing Systems*, 38:168990–169012, 2026b.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. Converting transformers to polynomial form for secure inference over homomorphic encryption. *arXiv preprint arXiv:2311.08610*, 2023.

Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Meiri, and Omri Soceanu. Power-softmax: Towards secure llm inference over encrypted data. *arXiv preprint arXiv:2410.09457*, 2024.

A Failure Modes of Approximate Token Selection: Token Errors and Embedding Divergence

We evaluate the released NEXUS (Zhang et al., 2024) Argmax on Llama-3.2-1B using 512 documents from the C4 validation set (Raffel et al., 2020), with one 128-token prefix per document and plaintext top-4096 candidate sets.

As shown in Figure 2, NEXUS changes the hard-selected token in only 4 of 512 cases (0.78%). However, the correct token attaining the maximum value does not guarantee the correctness of the FHE-approximated embedding fed into the next autoregressive step. Among the 508 cases with a correct hard selection, directly feeding back$\widetilde{\mathbf{e}} = \mathbf{b}^\top \mathbf{E}_C$, where $\mathbf{b}$ is the decrypted approximate selection vector and $\mathbf{E}_C$ contains the candidate embeddings, changes the subsequent greedy token in 149 cases (29.33%) relative to using the exact winner embedding.

This discrepancy exposes a distinct failure mode of approximate selection under FHE. An approximate Argmax may assign its maximum value to the correct token's position, while still producing

a selector $\mathbf{b}$ that deviates from an exact one-hot vector. Thus, residual approximation error in the selector is converted directly into a mixture of random embeddings.

This effect is particularly relevant to encrypted autoregressive generation because embedding retrieval is naturally implemented as a linear transformation of the encrypted selection vector. Unlike a plaintext table lookup, which returns exactly one embedding row, the linear encrypted lookup preserves residual nonzero coefficients in $\mathbf{b}$ and therefore propagates them into the recovered embedding. Consequently, approximate selection errors corrupt generation quality in two distinct ways. While large errors alter the hard token decision and lead to semantically wrong answers, the minor errors results in mixture of unrelated token embeddings representation passed to the next decoding step, which destabilizes subsequent hidden states and ultimately results in unstable or collapsed generation.

The 29.33% next-token divergence observed here therefore indicates that hard-selection accuracy alone can substantially underestimate the effect of approximate token selection on end-to-end autoregressive generation.

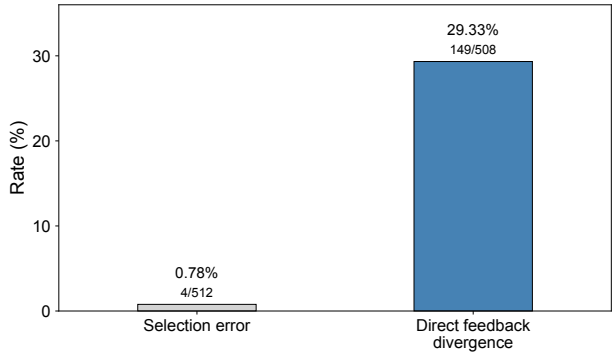


Figure 2: **Failure modes of approximate token selection.** *Selection error* is measured over all 512 test cases. *Direct-feedback divergence* isolates the 508 cases with a correct hard selection, measuring the frequency at which feeding back the approximated embedding—rather than the exact winner embedding—causes the model to predict a different greedy token at the next generation step.

B Details of Packing and Interleaved Batching.

This appendix formalizes the packing layouts and interleaved batching introduced in Section 4.1. We use $S = N/2$ CKKS slots, with $[q] = \{0, \dots, q-1\}$. For a plaintext vector, Rot_k denotes a cyclic left rotation by k positions over its logical length; $\mathrm{Rot}_k^{\mathrm{HE}}$ denotes the corresponding homomorphic rotation over the S ciphertext slots. The algebraic identities below describe ideal slotwise arithmetic, while decrypted CKKS outputs satisfy the corresponding relations approximately.

Column and row packing. We use column packing for matrix blocks and row packing for individual vectors (Zhang et al., 2026a; Halevi & Shoup, 2014). For a matrix block $X \in \mathbb{R}^{m \times p}$ with $m \le S$, column packing encrypts each column separately:

$$\mathrm{Enc}_{\mathrm{col}}(X) = \Big(\mathrm{Enc}\Big({X_{:,j}}^{\top} \parallel \mathbf{0}_{S-m}\Big)\Big)_{j \in [p]}, \tag{4}$$

where $\mathbf{0}_q$ denotes an all-zero vector of length q. For a row vector $\mathbf{x} \in \mathbb{R}^p$ with $p \le S$, row packing places its elements in a single ciphertext:

$$\mathrm{Enc}_{\mathrm{row}}(\mathbf{x}) = \mathrm{Enc}(\mathbf{x} \parallel \mathbf{0}_{S-p}). \tag{5}$$

Interleaved batching. Following Zhang et al. (2026a), we interleave multiple logical vectors to utilize the available slots. Let n be a divisor of S denoting the padded logical capacity per batch

lane, and let $B = S/n$. For $\mathbf{x}^{(r)} \in \mathbb{R}^n$, $r \in [B]$, define

$$\begin{aligned} \mathbf{x}_{\mathrm{I}} &= \mathcal{I}_B\Big((\mathbf{x}^{(r)})_{r\in[B]}\Big) \in \mathbb{R}^S, \\ (\mathbf{x}_{\mathrm{I}})_{tB+r} &= x_t^{(r)}, \qquad t \in [n],\ r \in [B]. \end{aligned} \tag{6}$$

The resulting ciphertext is $ct_{\mathbf{x}_{\mathrm{I}}} = \mathrm{Enc}(\mathbf{x}_{\mathrm{I}})$, where unused lanes are padded with zeros.

For any $k \in \mathbb{Z}$, this layout satisfies

$$(\mathrm{Rot}_{kB}(\mathbf{x}_{\mathrm{I}}))_{tB+r} = x^{(r)}_{(t+k) \bmod n}. \tag{7}$$

Indeed, adding kB to a slot index changes its logical coordinate but preserves its residue modulo B. Thus, $\mathrm{Rot}^{\mathrm{HE}}_{kB}$ rotates every logical vector by k positions without mixing batch lanes.

The capacity n is a packing-block parameter and is distinct from the model hidden dimension d. All projection blocks and attention operands use the same batch ordering and stride B. In our configuration,

$$m = d_h = n = 128, \qquad d = 3072, \qquad B = 256, \qquad S = nB = 32768.$$

Thus, a batched hidden state is represented by 24 interleaved row-packed ciphertexts, while a batched per-head vector occupies one ciphertext.

C Details of Replication primitive and Column-packed ct–ct Multiplication.

This appendix formalizes the replication primitive and Column-packed ct–ct multiplication introduced in Section 4.1.

Replication. We use the full-replication primitive of Halevi & Shoup (2014). For $\mathbf{v} \in \mathbb{R}^p$, $p \le S$, and $ct_{\mathbf{v}} = \mathrm{Enc}_{\mathrm{row}}(\mathbf{v})$, define

$$\begin{aligned} (ct_{\mathbf{r_i}})_{i\in[p]} &\leftarrow \mathrm{Replicate}_p(ct_{\mathbf{v}}), \\ \mathrm{Dec}(ct_{\mathbf{r_i}}) &\approx v_i \mathbf{1}_S, \qquad i \in [p], \end{aligned} \tag{8}$$

where $\mathbf{1}_S$ is the all-one row vector of length S. Each output therefore broadcasts one input coordinate across all ciphertext slots.

For interleaved input

$$\mathbf{v}_{\mathrm{I}} = \mathcal{I}_B\Big((\mathbf{v}^{(r)})_{r\in[B]}\Big), \qquad \mathbf{v}^{(r)} \in \mathbb{R}^n, \quad nB = S,$$

let $ct_{\mathbf{v}_{\mathrm{I}}} = \mathrm{Enc}(\mathbf{v}_{\mathrm{I}})$. The lane-wise variant is defined by

$$\begin{aligned} (ct_{\mathbf{r_i}})_{i\in[n]} &\leftarrow \mathrm{Replicate}_{n,B}(ct_{\mathbf{v}_{\mathrm{I}}}), \\ (\mathrm{Dec}(ct_{\mathbf{r_i}}))_{tB+r} &\approx v_i^{(r)}, \end{aligned} \tag{9}$$

for $i, t \in [n]$ and $r \in [B]$. This variant applies the logical replication operations independently to every lane, using lane-repeated masks and rotations by multiples of B. Consequently, each output broadcasts one coordinate within every lane without mixing different vectors.

Column-packed ct–ct multiplication. Let $C \in \mathbb{R}^{m\times p}$ with $m, p \le S$ and $\mathrm{Enc}_{\mathrm{col}}(C) = (ct_{\mathbf{c_i}})_{i\in[p]}$. For $\mathbf{v} \in \mathbb{R}^p$, multiplication by the transposed matrix satisfies

$$\mathbf{v}C^\top = \sum_{i\in[p]} v_i {C_{:,i}}^\top. \tag{10}$$

Given $ct_{\mathbf{v}} = \mathrm{Enc}_{\mathrm{row}}(\mathbf{v})$, we evaluate this product as

$$\begin{aligned} (ct_{\mathbf{r_i}})_{i\in[p]} &\leftarrow \mathrm{Replicate}_p(ct_{\mathbf{v}}), \\ ct_{\mathbf{y}} &\leftarrow \sum_{i\in[p]} \mathrm{Mult}(ct_{\mathbf{r_i}}, ct_{\mathbf{c_i}}). \end{aligned} \tag{11}$$

Algorithm 2 BSGS ct–pt vector–matrix multiplication

Require: $ct_{\mathbf{x_I}}$; plaintexts $\mathbf{p}_{j,i}$; $bg = n$, $nB = S$
Ensure: $ct_{\mathbf{y_I}}$ encrypting the interleaved products $(\mathbf{x}^{(r)}W)_{r\in[B]}$
Baby steps:
$ct_{\mathbf{u_0}} \leftarrow ct_{\mathbf{x_I}}$
for $i = 1, \ldots, b-1$ **do**
  $ct_{\mathbf{u_i}} \leftarrow \mathrm{Rot}^{\mathrm{HE}}_{iB}(ct_{\mathbf{x_I}})$
end for
Group accumulation:
for $j = 0, \ldots, g-1$ **do**
  $ct_{\mathbf{t_j}} \leftarrow \mathrm{PlainMult}(ct_{\mathbf{u_0}}, \mathbf{p}_{j,0})$
  for $i = 1, \ldots, b-1$ **do**
    $ct_{\mathbf{t_j}} \leftarrow \mathrm{Add}\big(ct_{\mathbf{t_j}}, \mathrm{PlainMult}(ct_{\mathbf{u_i}}, \mathbf{p}_{j,i})\big)$
  end for
end for
Giant steps:
$ct_{\mathbf{y_I}} \leftarrow ct_{\mathbf{t_0}}$
for $j = 1, \ldots, g-1$ **do**
  $ct_{\mathbf{y_I}} \leftarrow \mathrm{Add}\big(ct_{\mathbf{y_I}}, \mathrm{Rot}^{\mathrm{HE}}_{jbB}(ct_{\mathbf{t_j}})\big)$
end for
return $ct_{\mathbf{y_I}}$

Here, Mult denotes slotwise ciphertext–ciphertext multiplication. By Equation 8, the i-th product encrypts the i-th column of C, scaled by v_i, with zero padding in the remaining slots. Therefore,

$$\mathrm{Dec}(ct_{\mathbf{y}}) \approx (\mathbf{v}C^{\top}) \parallel \mathbf{0}_{S-m}. \tag{12}$$

This directly supports the query–key product $\mathbf{q}{C_K}^{\top}$ for a column-packed key cache C_K.

For the single-head query–key product, we instantiate $p = d_h$ and $C = C_K \in \mathbb{R}^{m\times d_h}$. In our configuration, $m = p = d_h = n = 128$.

Interleaved extension. The same computation applies to batched vectors and cache matrices when their lane layouts are aligned. Specifically, choose a common logical length n satisfying $n \mid S$ and $m, p \le n$, and set $B = S/n$. For each lane $r \in [B]$, let $\mathbf{v}^{(r)} \in \mathbb{R}^p$ and $C^{(r)} \in \mathbb{R}^{m\times p}$.

Pad each input vector and each matrix column to length n, then interleave them using the same lane ordering. Let $ct_{\mathbf{v_I}}$ denote the encrypted interleaved input and $ct_{\mathbf{c_{i,I}}}$ the encrypted interleaving of the padded i-th columns. We compute

$$\begin{aligned}(ct_{\mathbf{r_i}})_{i\in[n]} &\leftarrow \mathrm{Replicate}_{n,B}(ct_{\mathbf{v_I}}),\\ ct_{\mathbf{y_I}} &\leftarrow \sum_{i\in[p]} \mathrm{Mult}(ct_{\mathbf{r_i}}, ct_{\mathbf{c_{i,I}}}).\end{aligned} \tag{13}$$

Only the replicated outputs indexed by $i \in [p]$ are needed. For $t \in [n]$ and $r \in [B]$, the result satisfies

$$(\mathrm{Dec}(ct_{\mathbf{y_I}}))_{tB+r} \approx \begin{cases} \left(\mathbf{v}^{(r)}{C^{(r)}}^{\top}\right)_t, & 0 \le t < m,\\ 0, & m \le t < n.\end{cases} \tag{14}$$

Thus, each lane independently computes its vector–matrix product.

D Algorithm of Interleaved BSGS Multiplication

We describe multiplication by one plaintext weight block $W \in \mathbb{R}^{n\times n}$. For each batch lane $r \in [B]$, let $\mathbf{x}^{(r)} \in \mathbb{R}^n$ be the corresponding input feature block, where $nB = S$. Choose positive integers b and g such that $bg = n$.

For $k \in [n]$, define

$$\mathbf{d}_k = \mathrm{Diag}_k(W^\top), \qquad (\mathbf{d}_k)_t = w_{(t+k) \bmod n, t}.$$

For $j \in [g]$ and $i \in [b]$, define the plaintext slot vector $\mathbf{p}_{j,i} \in \mathbb{R}^S$ by

$$(\mathbf{p}_{j,i})_{tB+r} = \left(\mathrm{Rot}_{-jb}(\mathbf{d}_{jb+i})\right)_t, \qquad t \in [n],\ r \in [B].$$

These slot vectors are encoded offline. Algorithm 2 computes the products while preserving interleaved row packing. With rotation keys for the required offsets, it uses n ct–pt multiplications, $n-1$ additions, and $b + g - 2$ rotations. For a single weight block, a near-balanced split gives $O(\sqrt{n})$ rotations. For blockwise projections, we select the split using Eq. equation 23.

We prove that Algorithm 2 returns

$$\mathrm{Dec}(ct_{\mathbf{y}_\mathrm{I}}) \approx \mathcal{I}_B\Big((\mathbf{x}^{(r)}W)_{r\in[B]}\Big). \tag{15}$$

We first establish the identity in exact slot arithmetic. The approximate equality accounts for CKKS encoding and evaluation errors.

Rotation compatibility. Since $S = nB$, for any $k \in \mathbb{Z}$,

$$\begin{aligned}\left(\mathrm{Rot}_{kB}(\mathbf{x}_\mathrm{I})\right)_{tB+r} &= (\mathbf{x}_\mathrm{I})_{((t+k) \bmod n)B+r} \\ &= x^{(r)}_{(t+k) \bmod n}.\end{aligned} \tag{16}$$

Hence, a physical rotation by kB applies Rot_k independently to each logical row. Slotwise addition and multiplication also preserve these batch lanes.

Diagonal identity. Let $W = (w_{u,t})$. By definition,

$$(\mathbf{d}_k)_t = w_{(t+k) \bmod n, t}.$$

For each output coordinate $t \in [n]$,

$$\begin{aligned}\left(\sum_{k\in[n]} \mathrm{Rot}_k(\mathbf{x}^{(r)}) \otimes \mathbf{d}_k\right)_t &= \sum_{k\in[n]} x^{(r)}_{(t+k) \bmod n} w_{(t+k) \bmod n, t} \\ &= \sum_{u\in[n]} x^{(r)}_u w_{u,t} \\ &= (\mathbf{x}^{(r)}W)_t.\end{aligned} \tag{17}$$

BSGS decomposition. By rotation compatibility, $ct_{\mathbf{u_i}}$ represents $\mathrm{Rot}_i(\mathbf{x}^{(r)})$ in batch lane r. The logical row represented by $ct_{\mathbf{t_j}}$ in this lane is

$$\mathbf{t}^{(r)}_j = \sum_{i\in[b]} \mathrm{Rot}_i(\mathbf{x}^{(r)}) \otimes \mathrm{Rot}_{-jb}(\mathbf{d}_{jb+i}). \tag{18}$$

The giant-step rotation $\mathrm{Rot}^{\mathrm{HE}}_{jbB}$ applies Rot_{jb} to this logical row. Since rotations distribute over addition and Hadamard multiplication, the output in lane r equals

$$\begin{aligned}\sum_{j\in[g]} \mathrm{Rot}_{jb}(\mathbf{t}^{(r)}_j) &= \sum_{j\in[g]} \sum_{i\in[b]} \mathrm{Rot}_{jb+i}(\mathbf{x}^{(r)}) \otimes \mathbf{d}_{jb+i} \\ &= \sum_{k\in[n]} \mathrm{Rot}_k(\mathbf{x}^{(r)}) \otimes \mathbf{d}_k \\ &= \mathbf{x}^{(r)}W.\end{aligned} \tag{19}$$

The second equality follows from the unique decomposition $k = jb + i$, with $j \in [g]$ and $i \in [b]$. No operation mixes batch lanes, so the output retains interleaved row packing.

Replacing exact slot operations with their CKKS counterparts yields Eq. equation 15. For blockwise multiplication, correctness follows by applying the same identity to each padded block and accumulating the products over input blocks.

Cross-block rotation reuse. Homomorphic rotations incur key-switching overhead. For blockwise projections, we therefore choose the BSGS parameters according to the numbers of input and output ciphertexts, rather than optimizing each weight block independently.

Let c_{in} and c_{out} denote the input and output ciphertext counts of one jointly evaluated blockwise projection over a batch of B examples. For $\alpha \in [c_{\text{in}}]$ and $\beta \in [c_{\text{out}}]$, let $W_{\alpha,\beta} \in \mathbb{R}^{n\times n}$ be the corresponding padded weight block. The padded input block in lane r is $\mathbf{x}_\alpha^{(r)} \in \mathbb{R}^n$, and the β-th output block satisfies

$$\mathbf{y}_\beta^{(r)} = \sum_{\alpha\in[c_{\text{in}}]} \mathbf{x}_\alpha^{(r)} W_{\alpha,\beta}. \tag{20}$$

All blocks use the same (n, B) layout and a common BSGS split $bg = n$.

Let $ct_{\mathbf{x}_{\alpha,\mathrm{I}}}$ encrypt the interleaving of $(\mathbf{x}_\alpha^{(r)})_{r\in[B]}$. For each input ciphertext, we compute

$$ct_{\mathbf{u}_{\alpha,\mathbf{i}}} \leftarrow \mathrm{Rot}^{\mathrm{HE}}_{iB}(ct_{\mathbf{x}_{\alpha,\mathrm{I}}}), \qquad i \in [b], \tag{21}$$

where the zero-offset rotation is the identity. These baby-step rotations are reused across all output blocks.

Define $\mathbf{p}_{\alpha,\beta;j,i}$ by the preceding plaintext construction with $W = W_{\alpha,\beta}$. For each output block β and giant-step index $j \in [g]$, we accumulate contributions from all input blocks before applying the giant-step rotation:

$$\begin{aligned} ct_{\mathbf{t}_{\beta,\mathbf{j}}} &\leftarrow \sum_{\alpha\in[c_{\text{in}}]} \sum_{i\in[b]} \mathrm{PlainMult}\big(ct_{\mathbf{u}_{\alpha,\mathbf{i}}}, \mathbf{p}_{\alpha,\beta;j,i}\big), \\ ct_{\mathbf{y}_{\beta,\mathrm{I}}} &\leftarrow ct_{\mathbf{t}_{\beta,\mathbf{0}}} + \sum_{j=1}^{g-1} \mathrm{Rot}^{\mathrm{HE}}_{jbB}(ct_{\mathbf{t}_{\beta,\mathbf{j}}}). \end{aligned} \tag{22}$$

Ciphertext sums denote repeated Add. This reordering preserves correctness because rotations distribute over addition.

With rotation keys for the required offsets, the baby steps require $c_{\text{in}}(b-1)$ rotations, while the giant steps require $c_{\text{out}}(g-1)$ rotations. The total number of nonzero ciphertext rotations is

$$R_{\text{rot}}(b, g) = c_{\text{in}}(b-1) + c_{\text{out}}(g-1), \qquad bg = n. \tag{23}$$

This count applies to the entire batch and does not incur an additional factor of B. Offline plaintext rotations, packing conversion, and scale management are excluded.

Ignoring the integer and boundary constraints, the continuous balancing point is

$$b_0 = \sqrt{\frac{nc_{\text{out}}}{c_{\text{in}}}}, \qquad g_0 = \sqrt{\frac{nc_{\text{in}}}{c_{\text{out}}}}. \tag{24}$$

We select a feasible factorization by minimizing the rotation count over positive divisors of n:

$$\begin{aligned} b^\star &\in \operatorname*{arg\,min}_{\substack{b\in\mathbb{Z}_{>0}\\ b\mid n}} \left[c_{\text{in}}(b-1) + c_{\text{out}}\left(\frac{n}{b}-1\right)\right], \\ g^\star &= n/b^\star. \end{aligned} \tag{25}$$

Thus, more input ciphertexts favor a smaller baby-step width, whereas more output ciphertexts favor fewer giant-step groups. When the input and output ciphertext counts are equal, the rule reduces to a near-balanced factorization of n.

E Layout-Compatible KV Cache via Online Softmax

Algorithm 3 implements both paths and all ciphertexts use the interleaved layouts; the subscript I is omitted for brevity. For a single prefill-sequence block, we use a shared power-of-two capacity $n \geq \max(m, d_h)$ with $nB = S$. Each prefill-cache column is zero-padded along the sequence axis from m to n entries per batch lane, while each row-packed query, key, and value is zero-padded along the feature axis from d_h to n entries per batch lane, using the same batch ordering

Algorithm 3 Layout-compatible online softmax attention

Require: Interleaved row-packed query $ct_{\mathbf{q}}$ and decode pairs $(ct_{\mathbf{k}_\tau}, ct_{\mathbf{v}_\tau})_{\tau\in\mathcal{D}}$
Require: Interleaved column-packed prefill cache $(ct_{\mathbf{k}_\mathbf{j}^{\mathrm{pre}}}, ct_{\mathbf{v}_\mathbf{j}^{\mathrm{pre}}})_{j\in[d_h]}$
Ensure: Attention output $ct_{\mathbf{o}}$ in interleaved row packing

Column-packed prefill-cache path:
$(ct_{\mathbf{r_j}})_{j\in[n]} \leftarrow \mathrm{Replicate}_{n,B}(ct_{\mathbf{q}})$
$ct_{\mathbf{s}_{\mathrm{pre}}} \leftarrow \sum_{j\in[d_h]} \mathrm{Mult}(ct_{\mathbf{r_j}}, ct_{\mathbf{k}_\mathbf{j}^{\mathrm{pre}}})$
$ct_{\mathbf{s}_{\mathrm{pre}}} \leftarrow \mathrm{PlainMult}(ct_{\mathbf{s}_{\mathrm{pre}}}, \mathbf{c})$
$ct_{\mathbf{e}_{\mathrm{pre}}} \leftarrow \mathrm{Exp}(ct_{\mathbf{s}_{\mathrm{pre}}})$
$ct_{\mathbf{e}_{\mathrm{pre}}} \leftarrow \mathrm{PlainMult}(ct_{\mathbf{e}_{\mathrm{pre}}}, \mathbf{m}_{\mathrm{pre}})$
$ct_{\mathbf{w}} \leftarrow \mathrm{Reduce}_{n,B}(ct_{\mathbf{e}_{\mathrm{pre}}})$
for $j = 0, \ldots, d_h - 1$ **do**
    $ct_{\mathbf{f_j}} \leftarrow \mathrm{Reduce}_{n,B}\Big(\mathrm{Mult}(ct_{\mathbf{e}_{\mathrm{pre}}}, ct_{\mathbf{v}_\mathbf{j}^{\mathrm{pre}}})\Big)$
end for
$ct_{\mathbf{u}} \leftarrow \sum_{j\in[d_h]} \mathrm{PlainMult}(ct_{\mathbf{f_j}}, \mathbf{m}_j)$

Row-packed decode-KV path:
for $\tau \in \mathcal{D}$ **do**
    $ct_{\mathbf{s}_\tau} \leftarrow \mathrm{Reduce}_{n,B}(\mathrm{Mult}(ct_{\mathbf{q}}, ct_{\mathbf{k}_\tau}))$
    $ct_{\mathbf{s}_\tau} \leftarrow \mathrm{PlainMult}(ct_{\mathbf{s}_\tau}, \mathbf{c})$
    $ct_{\mathbf{e}_\tau} \leftarrow \mathrm{Exp}(ct_{\mathbf{s}_\tau})$
    $ct_{\mathbf{u}} \leftarrow \mathrm{Add}(ct_{\mathbf{u}}, \mathrm{Mult}(ct_{\mathbf{e}_\tau}, ct_{\mathbf{v}_\tau}))$
    $ct_{\mathbf{w}} \leftarrow \mathrm{Add}(ct_{\mathbf{w}}, ct_{\mathbf{e}_\tau})$
end for

Shared normalization:
$ct_{\mathbf{o}} \leftarrow \mathrm{Mult}(ct_{\mathbf{u}}, \mathrm{Inv}(ct_{\mathbf{w}}))$
return $ct_{\mathbf{o}}$

and stride B in both layouts. The post-exponentiation mask $\mathbf{m}_{\mathrm{pre}}$ retains only the first m logical positions within each batch lane, ensuring that padded sequence positions contribute to neither the numerator nor the denominator. Here, $\mathrm{Reduce}_{n,B}$ broadcasts each batch lane's sum across all n logical positions within that lane, using rotation offsets $B, 2B, \ldots, nB/2$. The plaintext mask $\mathbf{m}_j$ retains logical position j within each batch lane, while $\mathbf{m}_{\mathrm{pre}}$ retains valid prefill positions. Here, $\mathbf{c} = d_h^{-1/2}\mathbf{1}_S$, Exp and Inv denote homomorphic approximations of the exponential and reciprocal functions. Ciphertext sums denote repeated Add.

After both paths, the numerator accumulator $ct_{\mathbf{u}}$ represents the zero-padded vector $\mathbf{u}_{\mathcal{P}}^{(r)} + \mathbf{u}_{\mathcal{D}}^{(r)}$ in lane r. The denominator accumulator $ct_{\mathbf{w}}$ represents $\omega_{\mathcal{P}}^{(r)} + \omega_{\mathcal{D}}^{(r)}$, broadcast across all n logical positions within that lane. The final multiplication therefore implements the shared normalization in Section 4.2.

F Additional Experimental Details and Analysis

This appendix supplements Section 5 with the full implementation configuration, numerical error profiles, and runtime analyses. The error profiles support the accuracy recovery results, while the runtime analyses connect the measured speedups to multi-step behavior and layer execution.

F.1 Implementation Details

CKKS configuration. We use a polynomial modulus degree of $N = 2^{16}$, providing 2^{15} complex-valued slots per ciphertext, and a 1743-bit coefficient modulus to target 128-bit security, following the Homomorphic Encryption Standard (Albrecht et al., 2022). The initial scale is 2^{46}. We adopt scale propagation (Bossuat et al., 2021) to manage scale and noise growth throughout homomorphic evaluation.

CPU implementation. The CPU implementation is written in C++ using Microsoft SEAL (Microsoft Research, 2023) for CKKS operations and FHE-MP-CNN (Lee et al., 2022) for bootstrapping. OpenMP (Dagum & Menon, 1998) enables multi-threaded execution on an AMD EPYC 9755 128-Core Processor.

GPU implementation We implement CKKS operations and bootstrapping using PhantomFHE (Yang et al., 2024). All reported online computation is performed on a single NVIDIA H200. Offline preprocessing and auxiliary storage are excluded from the reported online latency.

Training and selection. GSM8K-Aug (Deng et al., 2024) augments the original GSM8K (Cobbe et al., 2021) training set to approximately 385K examples. The 3B server produces 6 latent states per question, which condition the client's Hidden-Prefix (HP) and Latent-to-CoT (L2C) branches. The selector is fitted and calibrated on GSM8K-Aug splitdev and applied unchanged to GSM-Hard, MultiArith, and SVAMP. Oracle selection in Table 1 counts an example as correct whenever either branch succeeds.

F.2 Cross-Model Generality

We extend TELEX to two latent reasoning servers beyond the main Llama experiments: **SIM-CoT (CODI) on Qwen3-4B** and **SIM-CoT (Coconut) on GPT-2**. The Qwen server transfers 6 reasoning latents to Qwen3-0.6B, while the GPT-2 server transfers 10 reasoning latents to Pythia-14M. These experiments cover different latent reasoning methods, base models, and server–client scales.

Table 5: **Generality across latent reasoning models.** GSM8K-Aug accuracy (%) and recovery (%), defined as $100\,\mathrm{Acc}_{\mathrm{client}}/\mathrm{Acc}_{\mathrm{server}}$.

(a) SIM-CoT (CODI)
Qwen3-4B† $\rightarrow$ Qwen3-0.6B

Model	GSM8K-Aug	Recovery
Server (4B)	59.59	100.00
Client (0.6B)	**60.88**	**102.16**

(b) SIM-CoT (Coconut)
GPT-2 $\rightarrow$ Pythia-14M

Model	GSM8K-Aug	Recovery
Server (124M)	44.30	100.00
Client (14M)	**43.59**	**98.40**

†We reproduced SIM-CoT (CODI) on Qwen3-4B to obtain this server.

Table 5 shows that both clients recover essentially all of their corresponding server models' accuracy. The 0.6B Qwen client achieves 60.88% accuracy, corresponding to 102.16% recovery, while the 14M Pythia client achieves 43.59% accuracy with 98.40% recovery. The Qwen result demonstrates that TELEX generalizes to a different base model, while the GPT-2–Pythia result further shows cross-family transfer through Coconut's reasoning latents. Together, these results support the generality of TELEX across model architectures and families.

F.3 Scaling to larger servers

Scaling to Llama-3-8B was constrained by reproducibility issues with the publicly released SIM-CoT checkpoint, which did not achieve the reported performance under our evaluation and reproduction. Moreover, SIM-CoT results on larger Llama models still exhibit a GSM8K-Aug accuracy gap between latent reasoning and SFT-CoT. We therefore conduct the main TELEX evaluation with 3B–4B server models, while using GPT-2 as a complementary smaller-scale study.

F.4 Error Analysis

Figure 3 characterizes FHE error across transformer layers during the first decoding step and across successive latent steps. Across layers, errors remain approximately centered around zero, while their spread increases in deeper layers. Across latent steps, the range peaks at step 3 and then narrows rather than growing monotonically. The absolute mean remains below 7.4×10^{-4} across all plotted latent steps, indicating little systematic drift. Together with the 99.4% accuracy recovery under FHE in Table 2, these profiles support the numerical stability of multi-step latent transfer.

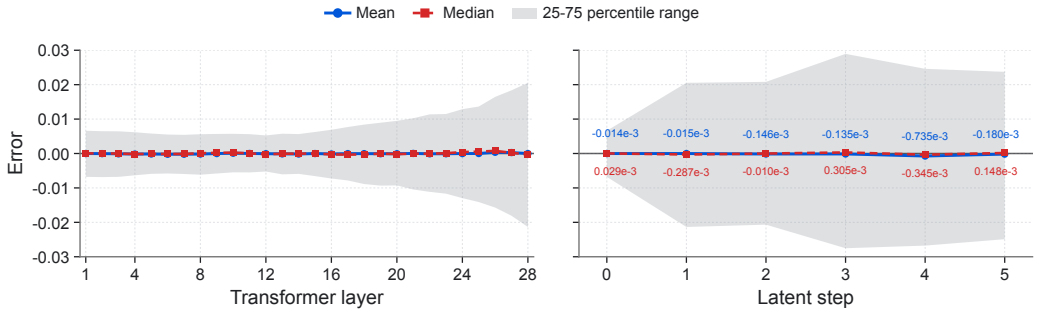


Figure 3: FHE error across transformer layers during the first decoding step (left) and across latent steps (right).

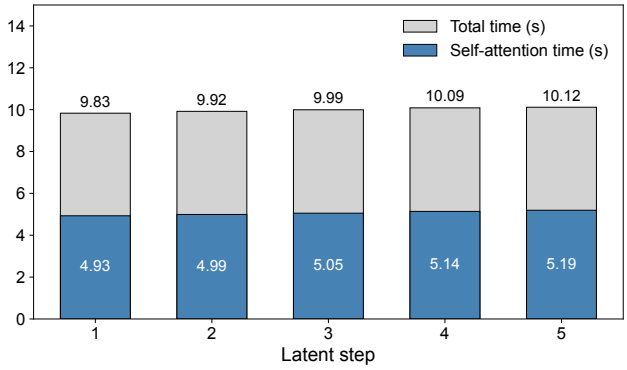


Figure 4: Per-step decoding time of TELEX. Total time remains nearly constant over the five steps, increasing from 9.83 to 10.12 s.

F.5 Runtime Analysis

F.5.1 Multi-step Decoding

The six-state trajectory consists of one latent produced by prefill and five subsequent decoding steps.Figure 4 extends the first-step comparison in Table 4 to five decoding steps. Per-step time increases only from 9.83 to 10.12 s, showing that the low decoding cost is maintained as the latent trajectory progresses. attention accounts for approximately 50.7% of the total runtime across these steps and remains a major optimization target.

F.5.2 Runtime Breakdown of a Transformer Layer

Table 6 reports operation-level GPU times and ciphertext-level transitions for the first decoding step of one Llama-3.2-3B transformer layer at batch size 256. The operation groups sum to 89.83 s, or approximately 0.35 s per input. This trace complements the stack-level operator comparison in Table 4 by exposing query-layout expansion, arithmetic evaluation, and level scheduling.

Runtime distribution. The self-attention component takes 59.08 s (65.8%), the FFN 29.43 s (32.8%), and the two RMSNorm blocks 1.32 s (1.5%). The self-attention and FFN totals include their projections and boundary bootstraps; Table 4 instead separates these costs into operator groups. Query replication for prefix-cache attention takes 24.85 s (27.7% of the layer total). The two boundary vector bootstraps each take 14.01 s, totaling 28.02 s (31.2%), while the four ct–pt projection groups—Q/K/V, attention output, gate/up, and down—take 23.70 s (26.4%). Together, these three categories account for 85.2% of the layer runtime and identify the main costs of the packed decoding pipeline.

Table 6: Runtime breakdown and multiplicative depth change of TELEX for the first decoding step of a Llama-3.2-3B transformer layer with a batch size of 256. Ct–pt and ct–ct denote ciphertext–plaintext and ciphertext–ciphertext operations, respectively; VMM denotes vector–matrix multiplication. The branch columns represent logical dataflow paths within each component and do not imply concurrent execution. A transition $\ell_{\text{in}} \to \ell_{\text{out}}$ indicates the ciphertext level before and after an operation. A timing entry spanning multiple rows gives the aggregate time of that operation group and is counted only once. The amortized time is obtained by dividing the total by 256 inputs.

Component	Operation		Depth		GPU time (s)
	Branch 1	Branch 2	Branch 1	Branch 2	
RMSNorm	-	Square	-	$18 \to 17$	0.03
	-	Reduce & Mean	-	$17 \to 16$	
	-	InvSqrt	-	$16 \to 1$	0.59
	-	Bootstrapping	-	$1 \to 21$	
	Ct-ct Hadamard Product	-	$21 \to 20$	-	0.04
Self Attention	Ct-pt VMM	-	$20 \to 19$	-	7.30
	-	Replicate	-	$19 \to 18$	24.85
	Ct-ct Hadamard Product	Ct-ct VMM	$19 \to 18$	$18 \to 17$	11.40
	Reduce	Exp	$18 \to 18$	$17 \to 9$	
	Exp	Reduce	$18 \to 10$	$9 \to 9$	
	Ct-ct Hadamard Product	Ct-ct Hadamard Product	$10 \to 9$	$9 \to 8$	
	-	Reduce	-	$8 \to 8$	
	Merge & Inv	Mask & Merge	$8 \to 4$	$8 \to 7$	0.15
	Ct-ct Hadamard Product	-	$4 \to 3$	-	
	Ct-pt VMM	-	$3 \to 2$	-	1.37
	Bootstrapping	-	$2 \to 1 \to 21$	-	14.01
RMSNorm	Same procedure as previous RMSNorm		$21 \to 20$		0.66
FFN	Ct-pt VMM	-	$20 \to 10 \to 9$	-	11.51
	-	SiLU	-	$9 \to 3$	0.39
	Ct-ct Hadamard Product	-	$3 \to 2$	-	
	Ct-pt VMM	-	$2 \to 1$	-	3.52
	Bootstrapping	-	$1 \to 21$	-	14.01
Total					89.83
Total (Amortized to 256 inputs)					0.35

Level scheduling. The trace records both arithmetic level consumption and explicit level alignment. We use modulus switching to lower the ciphertext level where the remaining computation permits, accelerating subsequent evaluation. For example, the FFN transition $20 \to 10 \to 9$ first switches to level 10 for faster evaluation; the gate/up projections then consume one level. In each RMSNorm block, the inverse-RMS factor is bootstrapped before multiplication with the preserved input. The attention and FFN updates are refreshed separately at their respective boundaries. These schedules connect the reported operation times to the level management used during decoding.