BEYOND STATIC MASKS: ATTACKING AND DEFENDING TEE-ASSISTED LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

TEE–GPU collaborative inference enables efficient deployment of proprietary large language models (LLMs) by combining trusted CPU execution with untrusted GPU acceleration. Recent approaches, such as SLIM, reduce TEE–GPU interaction through transformed-domain propagation across decoder blocks. However, we reveal that this propagation mechanism introduces a transformation identifiability vulnerability under multi-query inference. By exploiting reusable linear relationships exposed during authorization, an adversary can recover equivalent secret transformations and compromise the protection without accessing the original secrets. To address this issue, we propose GateFlow, a secure authorization propagation framework that redesigns the initial decoder execution while preserving efficient transformed-domain propagation. GateFlow prevents recoverable intermediate representations during authorization and enables subsequent decoder blocks to inherit the propagation mechanism without additional TEE involvement. Extensive experiments validate the effectiveness of the proposed attack and demonstrate that GateFlow effectively mitigates transformation recovery attacks with practical inference overhead.

1 INTRODUCTION

Model providers increasingly deploy large language models (LLMs) on user devices (Apple, 2025), edge platforms (Yato et al., 2023), and third-party infrastructure (Amazon Web Services). On-device and edge deployment can reduce communication latency, support offline operation, and keep application data local. These benefits require placing model parameters in environments outside the provider's control. An adversary controlling such an environment may inspect runtime memory or exploit software vulnerabilities to extract the deployed weights (Sun et al., 2021). The stolen weights enable unauthorized use and redistribution of a model that required substantial investment in data collection, architecture design, training, and optimization. Therefore, model providers need to make their models available for inference while protecting them from unauthorized copying.

Trusted execution environments (TEEs), such as Intel SGX, use hardware-enforced isolation to protect code and data from unauthorized access or modification by host software, including the operating system (Costan & Devadas, 2016; Intel). Model weights can therefore be decrypted and used inside the enclave without exposing them to the host. However, executing an entire LLM inside a CPU TEE is costly: attention and feed-forward layers require substantial matrix computation, while model weights and intermediate states impose considerable memory demands. Therefore, existing systems combine CPU TEEs with widely available hardware accelerators, such as GPUs, to improve inference efficiency (Zhang et al., 2024; Wang et al., 2025). The TEE protects model secrets and performs security-critical operations, while the accelerator handles the bulk of computation.

A representative approach is TEE–GPU collaborative inference, which divides the execution of an LLM decoder between the trusted and untrusted domains. Specifically, the GPU performs computation-intensive matrix multiplications, such as those used for linear projections in attention and feed-forward networks. At the same time, the TEE retains security-critical operations, including secret transformation recovery and protected intermediate-state processing. However, due to the sequential dependencies among decoder blocks, each block requires the protected intermediate representation generated by the previous block to continue execution. Therefore, existing interactive schemes, such as GroupCover (Zhang et al., 2024) and ArrowCloak (Wang et al., 2025), require

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.

frequent TEE–GPU communication during inference, which becomes increasingly significant as the number of decoder blocks grows in modern LLMs.

To reduce TEE–GPU communication overhead, SLIM (Wang et al., 2026) introduces a propagation-based inference paradigm based on orthogonal transformations. By jointly transforming the model parameters and hidden representations, SLIM exploits the invariance properties of orthogonal transformations to enable normalization and nonlinear operations directly in the transformed domain. After the initial authorization performed by the TEE, the transformed representation can be propagated through subsequent decoder blocks, allowing these blocks to run on the GPU without repeated layer-wise TEE involvement. This propagation-based design removes the dependence of TEE–GPU communication on decoder depth, achieving up to a $13.8\times$ speedup in time-to-first-token (TTFT) compared with ArrowCloak (Wang et al., 2025).

In this work, we revisit the security of propagation-based TEE–GPU collaborative LLM inference and reveal a fundamental transformation identifiability vulnerability in SLIM under multi-query inference. This vulnerability lets an adversary recover equivalent authorization transformations and compromise the obfuscated model's protection without accessing the original secrets. To address this issue, we propose GateFlow[1], a secure authorization propagation framework that redesigns the first-decoder authorization procedure to eliminate recoverable intermediate relationships while preserving the transformed-domain propagation mechanism. The main contributions are summarized as follows:

- **Transformation Recovery Attack.** By collecting diverse observations of inference inputs, intermediate representations, and authorization outputs, an adversary can formulate the exposed relationships as an identifiable linear recovery problem and reconstruct equivalent hidden transformations.
- **Secure Authorization Propagation Framework.** We propose GateFlow, which introduces secure attention authorization and encrypted interaction to break the observable correlations exploited by transformation recovery attacks, while preserving transformed-domain propagation.
- **Security Analysis and Evaluation.** We provide a security analysis to establish sysname's confidentiality guarantees. Experiments demonstrate that GateFlow effectively resists model recovery attacks while introducing only moderate computational and communication overhead.

2 Related Works

Model Stealing Attacks. Model stealing attacks aim to obtain a functionally equivalent surrogate of a private model by exploiting the information available to an adversary. Existing model-stealing attacks can be broadly categorised as black-box or white-box. Black-box attacks assume that the adversary can only access the input-output behavior of the target model through query interfaces. By collecting sufficient query-response pairs, the adversary trains a surrogate model to approximate the target model's functionality (Tramèr et al., 2016; Orekondy et al., 2019; Jagielski et al., 2020). In contrast, white-box attacks assume stronger access to internal model information, such as model parameters, intermediate representations, or execution states, and directly exploit these privileged observations to recover model components or construct equivalent models (Sun et al., 2021; Rakin et al., 2022). In TEE-assisted LLM inference, the objective of model protection is to prevent an untrusted accelerator from obtaining sufficient internal information to perform white-box extraction, while ideally reducing the adversary's capability to a black-box setting.

TEE-based Obfuscated LLM Inference. Recent studies have explored weight obfuscation techniques for efficient TEE-assisted inference on untrusted accelerators. Early approaches, such as Magnitude (Hou et al., 2021) and NNSplitter (Zhou et al., 2023), reduce trusted computing base (TCB) by selectively modifying model parameters, but may leave detectable patterns. Later schemes, including SOTER (Shen et al., 2022), ShadowNet (Sun et al., 2023), TSQP (Sun et al., 2025), TransLinkGuard (Li et al., 2024), and CoreGuard (Li et al., 2025), adopt lightweight transformations such as scaling, permutation, and channel morphing. However, these vector-level transformations may preserve exploitable statistical information (Zhang et al., 2024; Wang et al., 2025). To enhance obfuscation strength, GroupCover (Zhang et al., 2024) introduces inter-vector mixing,

[1]The code is available at: https://anonymous.4open.science/r/GateFlow-5C3B/

while ArrowCloak (Wang et al., 2025) employs shared directional perturbations with lightweight trusted correction. Nevertheless, recent studies reveal that such designs may still expose structural dependencies in accelerator-visible parameters, enabling weight recovery attacks against shared-direction masking (Zhang et al., 2026; Liu et al., 2026). Recently, SLIM (Wang et al., 2026) introduces a propagation-based obfuscation mechanism that employs factorized orthogonal transformations to maintain the transformed representations across decoder blocks, thereby significantly reducing TEE–GPU interaction overhead. While SLIM substantially improves inference efficiency through propagation, its security under repeated inference executions remains an open question. In this work, we revisit the security of propagation-based obfuscation and explore how to achieve a better balance between inference efficiency and authorization security. A detailed comparison of existing schemes is provided in Appendix A.

3 PRELIMINARIES

3.1 TEE–GPU INFERENCE MODEL

We consider a TEE–GPU collaborative inference architecture for protecting a proprietary LLM M_{vic}. The TEE obfuscates sensitive model information and performs security-critical operations, while the untrusted GPU executes efficient inference on the protected model. During inference, the GPU processes protected intermediate representations, and the TEE performs necessary recovery operations according to the adopted authorization mechanism.

3.2 THREAT MODEL

Adversary's Goal. The adversary aims to steal the deployed proprietary LLM by recovering a functionally equivalent substitute model M_{sur} from the protected inference system of M_{vic}. Specifically, the adversary seeks model functionality that achieves inference performance comparable to the victim model, rather than necessarily recovering every secret transformation parameter. This goal follows the threat model commonly adopted in TEE-based model obfuscation schemes (Zhang et al., 2024; Wang et al., 2025).

Adversary's Capability. We consider an untrusted GPU adversary that follows the prescribed inference protocol but attempts to infer protected model information from exposed observations. The adversary can observe GPU-visible data, including obfuscated model parameters and inference-time intermediate representations (Tramèr et al., 2016; Orekondy et al., 2019; Jagielski et al., 2020), and can issue chosen queries to collect corresponding observations. However, the adversary cannot access any code, data, or secret transformations protected inside the TEE. We do not consider TEE implementation vulnerabilities or hardware side channels (Xu et al., 2015), which are orthogonal to the transformation-based protection studied in this work.

3.3 REVISITING SLIM'S SCHEME

SLIM leverages an orthogonal transformation to preserve the norm of a hidden representation, enabling nonlinear normalization operations such as RMSNorm to run directly in the transformed domain. Let $\Pi \in \mathbb{R}^{d\times d}$ be an orthogonal transformation matrix satisfying $\Pi\Pi^\top = I$. Let $\gamma_1 \in \mathbb{R}^d$ denote the learnable scaling vector of the first RMSNorm layer, and define $D_1 = \mathrm{diag}(\gamma_1)$. For a transformed input $x' = x\Pi$, the orthogonality of Π gives $\mathrm{RMS}(x\Pi) = \mathrm{RMS}(x)$. Therefore, the normalized representation becomes

$$\begin{aligned} y' &= \gamma_1 \odot \frac{x\Pi}{\mathrm{RMS}(x\Pi)} \\ &= yD_1^{-1}\Pi D_1. \end{aligned} \tag{1}$$

To cancel the transformation before the attention computation, SLIM obfuscates the query, key, and value projection weights as $W_q' = D_1^{-1}\Pi^\top D_1 W_q$, $W_k' = D_1^{-1}\Pi^\top D_1 W_k$, $W_v' = D_1^{-1}\Pi^\top D_1 W_v$. Hence, $Q' = Q$, $K' = K$, $V' = V$, allowing the attention operation, including the nonlinear softmax computation, to run directly on the GPU. The output projection is further transformed as $W_o' = W_o\Pi$, $b_o' = b_o\Pi$, such that the resulting attention output satisfies $o' = o\Pi$. The same principle is applied to the MLP. Similarly, let $D_2 = \mathrm{diag}(\gamma_2)$. Since the residual representation satisfies

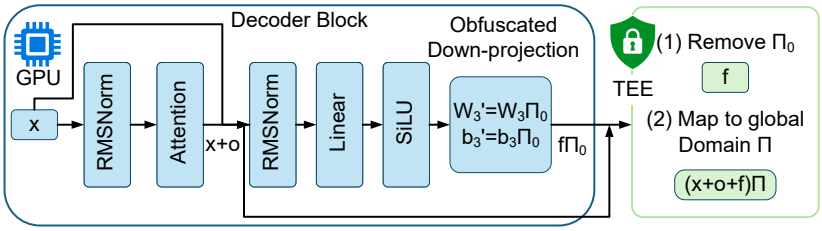

Figure 1: SLIM's TEE authorization mechanism that hides the intermediate state $(x + o + f)$ from the GPU and preserves the secrecy of the orthogonal transformation Π.

$x' + o' = (x+o)\Pi$, the second RMSNorm produces $z' = zD_2^{-1}\Pi D_2$. SLIM accordingly transforms the MLP weights as $W_1' = D_2^{-1}\Pi^\top D_2 W_1$, $W_2' = D_2^{-1}\Pi^\top D_2 W_2$, $W_3' = W_3\Pi, b_3' = b_3\Pi$, which yields $f' = f\Pi$. Consequently, each obfuscated decoder block preserves the transformed domain:

$$y'^{(i)} = \text{Decoder}_{\text{obf}}^{(i)}\left(y^{(i-1)}\Pi\right) = \text{Decoder}^{(i)}\left(y^{(i-1)}\right)\Pi = y^{(i)}\Pi. \tag{2}$$

By recursively applying this property, the output after n decoder blocks satisfies $y'^{(n)} = y^{(n)}\Pi$, allowing consecutive decoder blocks to be executed on the GPU without per-layer TEE recovery.

Transformation Identifiability. Unlike previous schemes that primarily obfuscate model parameters, SLIM couples model obfuscation with input transformation by applying the same secret matrix Π to hidden representations. This input-model coupling introduces a fundamental identifiability challenge: linear transformation applied to inputs is inherently reversible when correlated input-output observations are available. Specifically, if an adversary obtains a hidden representation and its transformed counterpart, $x' = x\Pi$, the recovery of Π becomes a linear inverse problem. Given multiple observations $\{(x_i, x_i')\}_{i=1}^N$, the adversary can formulate $X' = X\Pi$. When X has full column rank, the hidden transformation can be uniquely recovered as $\Pi = X^\dagger X'$.

TEE authorization. To avoid directly exposing the input-transformation pair, SLIM introduces an auxiliary authorization mechanism in the first decoder block. As illustrated in Fig. 1, instead of applying the global transformation Π to the input representation, SLIM first passes the input through the decoder block with an auxiliary orthogonal transformation Π_0. Specifically, the GPU uses obfuscated down-projection parameters $W_3' = W_3\Pi_0$ and $b_3' = b_3\Pi_0$, producing the intermediate representation in the auxiliary domain $f' = f\Pi_0$. The attention residual component $x + o$ is kept in the original domain during this process. The TEE then performs the authorization operation to remove the auxiliary transformation and convert the intermediate representation into the global propagation domain. Specifically, it first recovers the original MLP output $f'\Pi_0^\top = f\Pi_0\Pi_0^\top = f$, and subsequently applies the global transformation $(x + o + f'\Pi_0^\top)\Pi = (x + o + f)\Pi$. Therefore, the first decoder block outputs the transformed representation used in the original decoder execution, while preventing the GPU from directly obtaining the pair $(x + o + f, (x + o + f)\Pi)$.

Insight: Authorization Does Not Remove Identifiability

The introduced auxiliary transformation Π_0 remains linear and does not eliminate transformation identifiability, but only converts the single-transformation recovery problem into a multi-variable identification problem.

4 Transformation Recovery Attack

Problem Formulation. The SLIM authorization mechanism leads to a joint identification problem for two hidden transformations. Given GPU-visible observations $\{(y_i, x_i, o_i, f_i')\}_{i=1}^N$, the authorization relationship is given by $y_i = (x_i + o_i + f_i'\Pi_0^\top)\Pi$, where $f_i' = f_i\Pi_0$ denotes the auxiliary transformed representation. Let $a_i = x_i + o_i$, $P = \Pi$, and $Q = \Pi_0^\top\Pi$. The above relationship can be reformulated as $y_i = a_iP + f_i'Q$. Therefore, transformation recovery becomes identifying the unknown transformation pair (P, Q) from accumulated observations.

Attack. A direct approach is to construct:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} a_1 & f'_1 \\ a_2 & f'_2 \\ \vdots & \vdots \\ a_N & f'_N \end{bmatrix} \begin{bmatrix} P \\ Q \end{bmatrix}. \tag{3}$$

Let $A = [a_1; \ldots; a_N]$,$F' = [f'_1; \ldots; f'_N]$, $Y = [y_1; \ldots; y_N]$, where the stacking is performed along the token dimension. The above system can be compactly written as:

$$Y = [A \quad F'] \begin{bmatrix} P \\ Q \end{bmatrix} = Z \begin{bmatrix} P \\ Q \end{bmatrix}. \tag{4}$$

When the collected observations are sufficiently diverse such that $\mathrm{rank}(Z) = 2d$, the hidden transformations become uniquely identifiable. The adversary can recover them by:

$$\begin{bmatrix} P \\ Q \end{bmatrix} = Z^{\dagger} Y. \tag{5}$$

Finally, the adversary simultaneously obtains $\Pi = P$ and $\Pi_0^\top = QP^{-1}$. An equivalent recovery strategy can further exploit the orthogonality property of Π. Instead of jointly recovering (Π_0, Π), the adversary first eliminates Π using $\Pi\Pi^\top = I$ and recovers Π_0, followed by recovering Π from the original authorization relationship. Let $a_i = x_i + o_i$ and $R = \Pi_0^\top$. Then,

$$y_i y_i^T = (a_i + f'_i R)\Pi\Pi^T (a_i + f'_i R)^T \tag{6}$$

$$= (a_i + f'_i R)(a_i + f'_i R)^T. \tag{7}$$

Expanding the above equation gives

$$y_i y_i^T = a_i a_i^T + f'_i f_i'^T + f'_i R a_i^T + a_i R^T f_i'^T. \tag{8}$$

Therefore, the adversary can construct

$$C_i = y_i y_i^T - a_i a_i^T - f'_i f_i'^T, \tag{9}$$

which satisfies

$$C_i = f'_i R a_i^T + a_i R^T f_i'^T. \tag{10}$$

By vectorizing the above equation, the recovery of R can be formulated as a linear estimation problem:

$$\mathrm{vec}(C_i) = [(a_i \otimes f'_i) + (f'_i \otimes a_i)K]\,\mathrm{vec}(R), \tag{11}$$

where K denotes the commutation matrix. Since $R = \Pi_0^\top \in \mathbb{R}^{d\times d}$, the unknown vector $\mathrm{vec}(R)$ contains d^2 unknown elements. Collecting sufficient queries yields a linear system $M\,\mathrm{vec}(R) = c$. Recovering R requires the accumulated coefficient matrix to provide sufficient independent constraints. Specifically, when $\mathrm{rank}(M) = d^2$, the transformation $R = \Pi_0^\top$ can be uniquely recovered. After obtaining Π_0, the adversary computes $s_i = a_i + f'_i R$, and recovers the global transformation from $y_i = s_i \Pi$.

4.1 Attack Validation

Experimental Setup. We evaluate the proposed transformation recovery attack. We consider two models, including GPT2-medium (Radford et al., 2018) and Llama2-7B (Touvron et al., 2023), with hidden dimensions of 1024 and 4096, respectively. Three representative datasets are adopted to generate inference queries: Alpaca for instruction-following conversations (Taori et al., 2023), CodeAlpaca for code-related queries (Chaudhary, 2023), and SciQ for scientific question answering (Welbl et al., 2017). For each dataset, we sequentially feed sampled prompts into the model and collect the GPU-visible intermediate representations and authorization outputs required by the attack. We evaluate the recovery quality using two metrics. The relative error measures the normalized difference between the recovered and ground-truth transformations $\mathrm{Rel.Error} = \|\hat{\Pi} - \Pi\|_F / \|\Pi\|_F$. The cosine similarity measures the structural consistency between the recovered and original transformations, with values closer to 1 indicating more accurate recovery.

Table 1: Transformation recovery results under different models and datasets.

Model	Dataset	Prompts	Rank	Π Rel. Error	Π_0 Rel. Error	Π / Π_0 Cos.
GPT2-medium	Alpaca	24	2048/2048	5.14×10^{-4}	5.16×10^{-4}	0.99999 / 0.99999
GPT2-medium	CodeAlpaca	19	2048/2048	1.55×10^{-2}	1.56×10^{-2}	0.99988 / 0.99988
GPT2-medium	SciQ	16	2048/2048	2.81×10^{-4}	2.82×10^{-4}	1.00000 / 1.00000
Llama2-7B	Alpaca	87	8192/8192	3.31×10^{-4}	3.45×10^{-4}	0.99999 / 0.99999
Llama2-7B	CodeAlpaca	88	8192/8192	5.56×10^{-3}	5.58×10^{-3}	0.99998 / 0.99998
Llama2-7B	SciQ	53	8192/8192	2.21×10^{-3}	2.22×10^{-3}	0.99999 / 0.99999

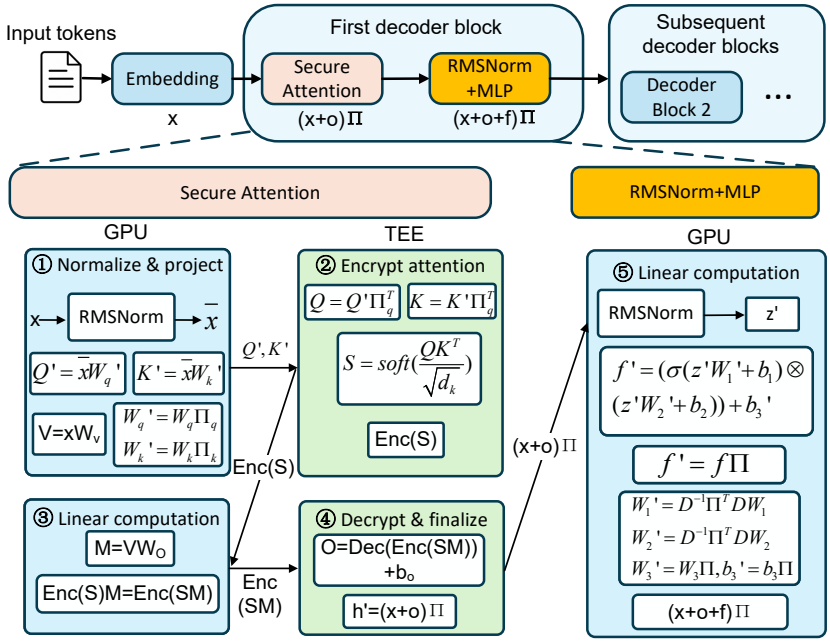

Figure 2: Overview of GateFlow's secure inference process.

Transformation Recovery Results. Table 1 summarizes the transformation recovery results under different models and datasets. The proposed attack achieves full-rank recovery with only 16–24 queries for GPT2-medium and 53–88 queries for Llama2-7B. After satisfying the rank condition, the recovered transformations closely match the original secret matrices, with relative errors below 1.6×10^{-2} for GPT2-medium and 5.6×10^{-3} for Llama2-7B, while the cosine similarities remain above 0.998. These results demonstrate that SLIM's hidden transformations can be accurately recovered from limited inference observations.

4.2 GateFlow Design

Overview. GateFlow introduces a secure interactive procedure in the first decoder block to protect sensitive intermediate representations and prevent transformation recovery, as shown in Fig. 2. Specifically, GateFlow combines an auxiliary representation transformation with a homomorphic mechanism to securely perform attention computation while preserving GPU acceleration for linear operations. The resulting hidden representation satisfies SLIM's propagation requirement after applying the global transformation Π, allowing subsequent decoder blocks to execute non-interactively on the GPU. The compatibility analysis and security analysis are provided in Appendices B and C, respectively.

Secure Inference Process. The secure execution of the first decoder block in GateFlow consists of five steps, as illustrated in Fig. 2.

Step 1. Given the input hidden representation x, the GPU first computes the normalized representation $\bar{x} = \mathrm{RMSNorm}(x)$. To protect the query and key representations, GateFlow applies auxiliary orthogonal transformations: $W'_q = W_q\Pi_q$ and $W'_k = W_k\Pi_k$, while keeping the value projection unchanged, i.e., $W'_v = W_v$. The GPU then obtains: $Q' = \bar{x}W'_q$, $K' = \bar{x}W'_k$, $V = \bar{x}W_v$.

Step 2. The transformed query and key representations are transferred to the TEE, which removes the auxiliary transformations $Q = Q'\Pi_q^T$, $K = K'\Pi_k^T$. The TEE then computes the attention weights: $S = \mathrm{Softmax}\left(QK^T/\sqrt{d_k}\right)$.

Step 3. To avoid exposing the attention weights, GateFlow encrypts S and outsources the following linear computation to the GPU. Let $M = VW_o$. Using ciphertext–plaintext linear evaluation, the GPU computes: $\mathrm{Enc}(S)M = \mathrm{Enc}(SM) = \mathrm{Enc}(SVW_o)$.

Step 4. The encrypted attention output is returned to the TEE, which decrypts the result and adds the output bias: $o = \mathrm{Dec}(\mathrm{Enc}(SVW_o)) + b_o$. The TEE then generates the transformed residual representation: $h' = (x + o)\Pi$.

Step 5. After the first decoder block establishes the transformed representation, the remaining decoder blocks follow SLIM's propagation mechanism. The hidden representation and corresponding model parameters remain in the transformed domain, allowing subsequent blocks to execute with the same propagation property.

Coefficient-wise RLWE Encryption (Coef-RLWE). To securely outsource the attention-weight multiplication, GateFlow introduces a lightweight RLWE-based encryption instantiation, termed Coef-RLWE. Unlike general-purpose homomorphic encryption schemes, Coef-RLWE only supports the required ciphertext–plaintext linear evaluation $\mathrm{Eval}(\mathrm{Enc}(S), M) = \mathrm{Enc}(SM)$, which is sufficient for securely computing the attention output projection while avoiding unnecessary homomorphic operations. Coef-RLWE encodes each column of the attention matrix as a polynomial and exploits RLWE encryption's linearity. Let $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ denote the polynomial ring, where q is the ciphertext modulus and N is the polynomial degree. Coef-RLWE consists of four algorithms: key generation KeyGen, encryption Enc, ciphertext–plaintext evaluation Eval, and decryption Dec.

Key Generation. The TEE executes $\mathrm{KeyGen}(1^\lambda)$ to generate a secret key $s \in R_q$, where λ denotes the security parameter. The TEE keeps the secret key inside and never exposes it to the GPU.

Encryption. Given an attention matrix $S \in \mathbb{R}^{L\times L}$, the TEE encodes each column of S as a polynomial over the ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$. Specifically, the k-th column is encoded as

$$m_k(X) = \sum_{i=1}^{L} \hat{S}_{ik} X^{i-1}, \tag{12}$$

where $\hat{S}_{ik}$ denotes the fixed-point representation of S_{ik} with scale factor Δ. The encoded polynomial is encrypted using the Coef-RLWE encryption algorithm $c_k = \mathrm{Enc}(m_k(X)) = (c_{0,k}, c_{1,k})$. The resulting ciphertexts $\{c_1, c_2, \ldots, c_L\}$ are transmitted to the GPU.

Ciphertext–Plaintext Evaluation. Given the plaintext matrix $M \in \mathbb{R}^{L\times d}$, the GPU encodes its elements into fixed-point representations and performs ciphertext–plaintext linear evaluation:

$$c_j^{out} = \sum_{k=1}^{L} p_{kj} c_k. \tag{13}$$

Due to the linearity of RLWE encryption, the evaluated ciphertext corresponds to the encrypted product:

$$c_j^{out} = \mathrm{Enc}\left(\sum_{k=1}^{L} p_{kj} m_k(X)\right). \tag{14}$$

Decryption. The TEE decrypts the returned ciphertext using the secret key $m_j(X) = \mathrm{Dec}(c_j^{out})$. The recovered polynomial is decoded to obtain the corresponding column of the attention output. Appendix D provides the detailed security definition, correctness analysis, noise bound, and parameter selection.

5 Experiments

Experimental Setup. We further evaluate GateFlow on three additional LLMs, including Gemma-3-1B (Team et al., 2025), Llama-3.2-3B (Grattafiori et al., 2024), and Qwen3-4B (Yang et al.,

Table 2: Defense effectiveness against model recovery attacks on different LLMs.

Scheme	Eval.	Gemma3-1B			LLaMA3.2-3B				Qwen3-4B			
		Alpaca	CodeAlpaca	SciQ	WiC	GoEmotions	FinQA	PubMedQA	WiC	GoEmotions	FinQA	PubMedQA
SOTER	Protected	60.12	80.73	96.96	54.37	8.31	0.00	0.00	48.42	8.41	0.00	0.00
	GAME	62.60	83.75	98.06	73.84	55.47	53.08	79.51	73.26	53.54	67.41	87.86
ShadowNet	Protected	0.00	0.00	0.00	48.76	7.61	0.00	0.00	49.67	7.68	0.00	0.00
	GAME	62.62	83.75	98.04	74.12	55.03	53.37	78.52	72.91	53.62	66.91	88.12
TransLinkGuard	Protected	0.00	0.00	0.00	49.82	6.02	0.00	0.00	51.84	8.85	0.00	0.00
	GAME	62.60	83.77	98.06	73.88	55.41	53.02	78.91	73.25	53.44	67.46	87.92
CoreGuard	Protected	0.27	0.13	0.00	50.44	4.96	0.00	0.00	50.18	8.79	0.00	0.00
	GAME	62.62	83.74	98.04	74.51	54.69	60.24	85.18	73.41	55.08	64.72	84.51
TSQP	Protected	39.54	52.74	61.50	50.17	7.82	0.00	0.00	49.71	7.63	0.00	0.00
	GAME	62.61	83.74	98.06	74.36	52.22	55.47	84.21	71.12	54.31	69.74	89.12
SLIM	Protected	0.01	0.00	0.00	49.72	5.46	0.00	0.00	51.23	10.84	0.00	0.00
	GAME	5.12	4.38	0.05	50.44	7.21	0.00	0.00	50.29	7.06	0.00	0.00
GateFlow (Ours)	Protected	0.00	0.00	0.00	49.31	5.12	0.00	0.00	50.87	10.42	0.00	0.00
	GAME	**0.30**	**0.32**	**0.00**	**50.02**	**6.93**	**0.00**	**0.00**	**49.86**	**6.81**	**0.00**	**0.00**
GateFlow w/o Authorization	–	48.92	66.84	82.57	68.45	43.26	38.91	71.28	69.73	41.56	35.82	73.41
No-shield	–	62.74	83.68	98.20	73.62	57.12	59.84	87.51	73.27	56.57	68.33	89.12
Random	–	0.02	0.00	0.00	50.57	2.46	0.00	0.00	51.28	7.82	0.00	0.00

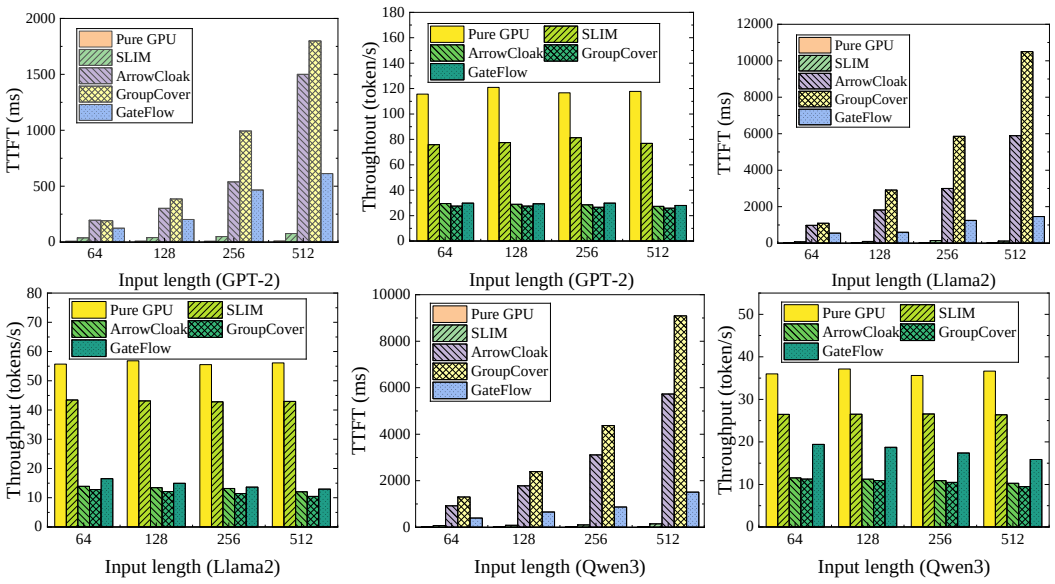

Figure 3: Comparison of TTFT and throughput between GateFlow and representative TEE-based LLM inference schemes under different prompt lengths.

.

2025), in addition to GPT-2-medium and Llama-2-7B used previously. We also incorporate four additional datasets from diverse domains, including WiC (Pilehvar & Camacho-Collados, 2019), GoEmotions (Demszky et al., 2020), FinQA (Chen et al., 2021), and PubMedQA (Jin et al., 2019).

Security Evaluation. Following the state-of-the-art model recovery attack methodology (Wang et al., 2025), we evaluate whether GateFlow can prevent practical model recovery. The attacker first exploits the collected inference observations to remove the protection layer, then reconstructs a functionally equivalent model. We compare three settings: (1) *No-Shield*, where the attacker directly accesses the original model; (2) *Random*, where the recovery starts from a randomly initialized model; and (3) *GateFlow w/o Authorization*, where the secure authorization mechanism is removed from GateFlow. Defence effectiveness is evaluated by the attacker's recovery capability under the same setting. As shown in Table 2, GateFlow significantly limits model recovery, with results close to the random baseline across different LLMs and datasets. Removing the authorization mechanism substantially improves an attacker's capabilities, underscoring the importance of secure authorization in preventing transformation-recovery and subsequent model-recovery attacks.

Table 3: TTFT overhead breakdown of GateFlow under different LLMs. The values represent the percentage contribution of GPU computation, TEE computation, and TEE–GPU communication.

Input Length	GPT-2-medium			Llama2-7B			Qwen3-4B		
	GPU	TEE	Transfer	GPU	TEE	Transfer	GPU	TEE	Transfer
64	7.21%	91.40%	1.39%	8.38%	90.17%	1.45%	10.22%	88.26%	1.52%
128	6.98%	91.12%	1.90%	8.22%	90.03%	1.75%	9.02%	89.16%	1.82%
256	5.60%	92.32%	2.09%	8.27%	89.61%	2.12%	8.53%	89.27%	2.20%
512	5.44%	92.11%	2.45%	10.24%	87.50%	2.26%	7.25%	90.13%	2.62%

Table 4: Model fidelity evaluation under different inference schemes.

Model	Scheme	Logit Cosine Similarity ↑		Relative Error ↓	
		FP32	FP16	FP32	FP16
GPT-2-medium	Original	1.0000	1.0000	0	0
	GateFlow w/o Enc.	0.9999	0.9998	1.2e-5	3.5e-5
	GateFlow	0.9998	0.9996	2.4e-5	6.8e-5
	SLIM	0.9999	0.9997	1.8e-5	5.2e-5
Llama2-7B	Original	1.0000	1.0000	0	0
	GateFlow w/o Enc.	0.9999	0.9997	1.5e-5	4.2e-5
	GateFlow	0.9998	0.9995	3.1e-5	8.5e-5
	SLIM	0.9999	0.9996	2.0e-5	6.1e-5
Qwen3-4B	Original	1.0000	1.0000	0	0
	GateFlow w/o Enc.	0.9999	0.9998	1.1e-5	3.8e-5
	GateFlow	0.9998	0.9996	2.8e-5	7.4e-5
	SLIM	0.9999	0.9997	1.9e-5	5.6e-5

Efficiency Evaluation. We evaluate GateFlow's inference overhead on an SEV-SNP-enabled server with an H100 GPU. We compare GateFlow with the GPU-only scheme (Pure GPU) and representative TEE-based LLM inference designs, including SLIM, ArrowCloak, and GroupCover. During evaluation, we set input sequence lengths to 64, 128, 256, and 512 tokens, while fixing the generated output length at 512 tokens. We measure end-to-end inference throughput and further analyse TTFT latency, which consists of TEE computation, TEE–GPU communication, and GPU computation. As shown in Fig. 3, GateFlow introduces additional latency compared with SLIM due to the secure authorization procedure and encryption/decryption operations. However, it preserves the propagation-based execution paradigm and avoids repeated TEE–GPU interactions required by layer-wise schemes. Consequently, GateFlow achieves lower inference latency than ArrowCloak and GroupCover while maintaining practical throughput. Further, Table 3 presents the TTFT breakdown of GateFlow. The results show that TEE computation dominates the overhead, while GPU computation and TEE–GPU communication contribute only a small portion of the total latency.

Model Fidelity Evaluation. Table 4 evaluates the impact of secure computation on model fidelity. GateFlow maintains high output consistency across different models, achieving cosine similarities above 0.9995 and relative errors below 8.5×10^{-5} under both FP32 and FP16 settings. These results demonstrate that the encrypted attention computation introduces negligible numerical degradation.

6 Conclusion

In this work, we revisit the security of propagation-based obfuscation for efficient LLM inference and reveal a transformation identifiability vulnerability in SLIM under multi-query inference. To address this issue, we propose GateFlow, a secure authorization propagation framework that redesigns the initial authorization process while preserving execution in the transformed domain. Extensive security analysis and experiments demonstrate that GateFlow effectively resists transformation recovery and model recovery attacks with practical inference overhead.

REFERENCES

Amazon Web Services. Use custom model import to import a customized open-source model into Amazon Bedrock. Amazon Bedrock Documentation. URL https://docs.aws.amazon.com/bedrock/latest/userguide/model-customization-import-model.html. Accessed: 2026-09-16.

Apple. Apple Intelligence gets even more powerful with new capabilities across Apple devices, June 2025. URL https://www.apple.com/newsroom/2025/06/apple-intelligence-gets-even-more-powerful-with-new-capabilities-across-apple-devices/. Accessed: 2026-09-16.

Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation, 2023.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, 2021.

Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. Goemotions: A dataset of fine-grained emotions. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4040–4054, 2020.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4270–4284, 2021.

Intel. Intel SGX SDK for Linux OS. URL https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/linux-overview.html. Accessed: 2026-09-16.

Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pp. 1345–1362, 2020.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2567–2577, 2019.

Qinfeng Li, Zhiqiang Shen, Zhenghan Qin, Yangfan Xie, Xuhong Zhang, Tianyu Du, Sheng Cheng, Xun Wang, and Jianwei Yin. Translinkguard: safeguarding transformer models against model stealing in edge deployment. In *Proceedings of the 32nd ACM international conference on multimedia*, pp. 3479–3488, 2024.

Qinfeng Li, Tianyue Luo, Xuhong Zhang, Yangfan Xie, Zhiqiang Shen, Lijun Zhang, Yier Jin, Hao Peng, Xinkui Zhao, XianWei Zhu, et al. Coreguard: Safeguarding foundational capabilities of llms against model stealing in edge deployment. *Advances in Neural Information Processing Systems*, 38:69221–69243, 2025.

Beijie Liu, Junyi Ouyang, Haoxuan Xu, Vincent Quentin Ulitzsch, Potung Yu, Yajie Zhao, and Mengyuan Li. Hiding directions, leaking structure: Breaking arrowcloak through low-rank structure. *arXiv preprint arXiv:2608.21615*, 2026.

Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 1–23. Springer, 2010.

Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4949–4958. IEEE, 2019.

Mohammad Taher Pilehvar and Jose Camacho-Collados. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (Long and short papers)*, pp. 1267–1273, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE symposium on security and privacy (SP)*, pp. 1157–1174. IEEE, 2022.

Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. {SOTER}: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX annual technical conference (USENIX ATC 22)*, pp. 723–738, 2022.

Yu Sun, Gaojian Xiong, Jianhua Liu, Zheng Liu, and Jian Cui. Tsqp: Safeguarding real-time inference for quantization neural networks on edge devices. In *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 2114–2132. IEEE, 2025.

Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX security symposium (USENIX security 21)*, pp. 1955–1972, 2021.

Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1596–1612. IEEE, 2023.

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

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pp. 601–618, 2016.

Pengli Wang, Bingyou Dong, Yifeng Cai, Zheng Zhang, Junlin Liu, Huanran Xue, Ye Wu, Yao Zhang, and Ziqi Zhang. Game of arrows: On the ({In-) Security} of weight obfuscation for {On-Device}{TEE-Shielded}{LLM} partition algorithms. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 279–298, 2025.

Wei Wang, Zihao Guan, Xing Zhou, Yan Ding, Yusong Tan, Jie Yu, and Bao Li. Slim: Secure and efficient inference for large language models on untrusted devices via tees. In *Forty-third International Conference on Machine Learning*, 2026.

Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE symposium on security and privacy*, pp. 640–656. IEEE, 2015.

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*, 2025.

Chitoku Yato, Dustin Franklin, and John Welsh. Bringing generative AI to life with NVIDIA Jetson. NVIDIA Technical Blog, October 2023. URL https://developer.nvidia.com/blog/bringing-generative-ai-to-life-with-jetson/. Accessed: 2026-09-16.

Menghui Zhang, Aoying Zheng, Guoxiao Liu, Zizhuang Deng, Jiejing Wen, Jincheng Zhuang, and Ran Tao. Beyond vector hiding: Breaking and mitigating shared-direction weight obfuscation in tee-offloaded large language models. *arXiv preprint arXiv:2608.26651*, 2026.

Zheng Zhang, Na Wang, Ziqi Zhang, Yao Zhang, Tianyi Zhang, Jianwei Liu, and Ye Wu. Groupcover: a secure, efficient and scalable inference framework for on-device model protection based on tees. In *Forty-first international conference on machine learning*, 2024.

Tong Zhou, Yukui Luo, Shaolei Ren, and Xiaolin Xu. Nnsplitter: an active defense solution for dnn model via automated weight obfuscation. In *International Conference on Machine Learning*, pp. 42614–42624. PMLR, 2023.

A LLM USAGE

LLM tools were used only for proofreading, grammar correction, and language polishing during manuscript preparation. The authors are fully responsible for the originality, correctness, and compliance of the paper, including all technical contributions, experimental results, and analyses.

B EFFICIENCY COMPARISON OF EXISTING SCHEMES

Table 5 summarises representative TEE-based obfuscated LLM inference schemes by TEE–GPU communication complexity and trusted-side computation overhead. Communication complexity denotes the online data exchange between the TEE and GPU during inference. In contrast, TEE computation complexity represents additional trusted execution introduced by transformation recovery, intermediate-state processing, or security enhancements. This comparison excludes offline preprocessing.

Most existing schemes require layer-wise TEE involvement, leading to communication overhead proportional to the number of decoder blocks, i.e., $O(nld)$. More recent approaches reduce this overhead through selective execution or transformed-domain propagation. In particular, SLIM maintains a consistent transformed representation across decoder blocks and eliminates repeated authorization operations, achieving $O(ld)$ communication complexity with $O(ld \log_k d)$ trusted computation.

To address the security limitations of fixed authorization transformations, GateFlow redesigns the authorization procedure of the first decoder block. Specifically, GateFlow performs secure attention authorization inside the TEE and establishes a protected transformed representation before entering the propagation domain. Unlike layer-wise authorization schemes, this additional trusted computation is independent of the number of decoder blocks and is introduced only at the first decoder block.

For prefill inference with sequence length l, the secure attention computation inside the TEE requires $O(l^2 d)$ computation to compute the attention scores and softmax. Meanwhile, the encrypted attention outsourcing introduces additional communication cost determined by the RLWE ciphertext representation. For a single attention head, the encrypted attention matrix is represented by $O(l)$ ciphertexts, and each ciphertext contains two degree-N polynomials. Therefore, the corresponding communication complexity is $O(lN \log q)$ bits, where N is the polynomial degree and q is the ciphertext modulus. When the polynomial degree scales with the sequence length, i.e., $N = \Theta(l)$, this communication can be approximately represented as $O(l^2)$ ciphertext-level elements.

After the initial transformed representation is established, subsequent decoder blocks directly inherit the propagation mechanism and introduce no additional TEE–GPU interaction, maintaining the same asymptotic propagation overhead as SLIM. During autoregressive decoding with a KV cache, attention computation involves only the current query and cached historical keys. Therefore, secure attention computation for each generated token requires $O(ld)$ computation, while encrypted attention communication scales with the KV-cache length rather than the full decoder depth.

Consequently, GateFlow replaces repeated layer-wise authorization with a secure authorization procedure in the first block. Its additional overhead depends on the attention computation and ciphertext representation of the current sequence, while the communication cost remains independent of the number of decoder blocks. This design preserves the efficiency advantage of propagation-based inference while addressing the security limitations of fixed authorization transformations.

Furthermore, Table 6 summarizes the detailed transformation mechanisms of representative TEE-based obfuscated LLM inference schemes. Specifically, it illustrates how different approaches transform model weights, perform GPU-side computation, and recover the original representation inside the TEE.

Table 5: Efficiency comparison of representative TEE-based obfuscated LLM inference schemes. Here, n denotes the number of decoder blocks, l denotes the input token length, d denotes the hidden dimension, and k denotes the decomposition block size used in structured orthogonal transformations.

Scheme	Communication	TEE Computation	Protection Mechanism	Execution
Magnitude	$O(nld)$	$O(nld)$	Selective modification	Layer-wise
NNSplitter	$O(nld)$	$O(nld)$	Selective weight obfuscation	Layer-wise
SOTER	$O(nld)$	$O(nld)$	Parameter morphing	Layer-wise
ShadowNet	$O(nld)$	$O(nld)$	Weight transformation	Layer-wise
GroupCover	$O(nld)$	$O(nldk)$	Group-wise mixing	Layer-wise
TransLinkGuard	$O(nld)$	$O(nld)$	Channel-wise transformation	Layer-wise
TSQP	$O(nld)$	$O(nld)$	Quantization-based protection	Layer-wise
CoreGuard	$O(ld)$	$O(ld)$	Capability-oriented protection	Selective execution
ArrowCloak	$O(nld)$	$O(nld)$	Shared-direction masking	Layer-wise
SLIM	$O(ld)$	$O(ld\log_k d)$	Orthogonal transformation	Propagation
GateFlow	$O(lN\log q)$ $O(l_{\mathrm{kv}}N\log q)$	$O(l^2d)$ (prefill) $O(ld)$ (decode) $+O(ld\log_k d)$	Secure authorization + orthogonal transformation	Propagation

Table 6: Transformation mechanisms of representative TEE-based obfuscated LLM inference schemes. Here, π denotes a permutation matrix, Π denotes the hidden propagation transformation, μ represents a scaling factor, D denotes a diagonal scaling matrix, A_i denotes the group-wise mixing matrix in GroupCover, and W_{dissim} denotes the learned dissimilar weight matrix in TSQP.

Scheme	Weight Transformation	GPU Computing	TEE Recovery
Magnitude	$W' = W + r$	$y' = xW'$	$y = y' - xr$
NNSplitter	$W' = W + \Delta W_s$	$y' = xW'$	$W = (1-M)\odot W' + M \odot c$
SOTER	$W' = \mu W\pi$	$y' = xW\mu\pi$	$y = y'\pi^T/\mu$
ShadowNet	$W' = WD\pi$	$y' = xWD\pi$	$y = y'\pi^T D^{-1}$
GroupCover	$W' = \{A_iW_i\}_{i=1}^k\pi$	$y' = xW'$	$y = y'\pi^T A_i^{-1}$
TransLinkGuard	$W' = W\pi$	$y' = xW\pi$	$y = y'\pi^T$
CoreGuard	$W' = W\pi$	$y' = xW\pi$	$y = y'\pi^T$
TSQP	$W' = \mu W_{\mathrm{dissim}}\pi$	$y' = \mu xW_{\mathrm{dissim}}\pi$	$y = y'\pi^T/\mu$
ArrowCloak	$W' = (WD_1 + v\mathbf{1}^TD_2)\pi$	$y' = x(WD_1 + v\mathbf{1}^TD_2)\pi$	$y = (y'\pi^T - xv\mathbf{1}^TD_2)D_1^{-1}$
SLIM	$W' = W\Pi$	$y' = xW\Pi$	$y = y'\Pi^T$

C Architecture Compatibility

C.1 Adaptation to LayerNorm

The main text describes GateFlow based on RMSNorm. This section discusses its adaptation to LayerNorm-based architectures. The first normalization operation in the first decoder block is applied directly to the original hidden representation x before the secure authorization procedure and therefore remains unchanged. After the secure authorization procedure, the hidden representation is mapped into the transformed domain. Therefore, subsequent LayerNorm operations must preserve the normalization property under orthogonal transformations.

LayerNorm. For a LayerNorm operation, the normalized representation is defined as

$$\mathrm{LayerNorm}(x) = \gamma \odot \frac{x - \mu_x \mathbf{1}_d}{\sqrt{\sigma_x^2 + \epsilon}} + \beta, \tag{15}$$

where

$$\mu_x = \frac{1}{d} x\mathbf{1}_d, \quad \sigma_x^2 = \frac{1}{d}\|x - \mu_x \mathbf{1}_d^\top\|_2^2. \tag{16}$$

Consider a transformed hidden representation:

$$x' = x\Pi, \tag{17}$$

where Π is an orthogonal transformation satisfying

$$\Pi\Pi^\top = I, \quad \Pi\mathbf{1}_d = \mathbf{1}_d. \tag{18}$$

The mean of the transformed representation is preserved:

$$\mu_{x'} = \frac{1}{d}x\Pi\mathbf{1}_d = \frac{1}{d}x\mathbf{1}_d = \mu_x. \tag{19}$$

Meanwhile, the variance remains invariant under the orthogonal transformation:

$$\begin{aligned} \sigma_{x'}^2 &= \frac{1}{d}\|x\Pi - \mu_x\mathbf{1}_d^\top\|_2^2 \\ &= \frac{1}{d}\|(x - \mu_x\mathbf{1}_d^\top)\Pi\|_2^2 \\ &= \sigma_x^2. \end{aligned} \tag{20}$$

Therefore, the normalized component of LayerNorm satisfies:

$$\frac{x\Pi - \mu_{x'}\mathbf{1}_d^\top}{\sqrt{\sigma_{x'}^2 + \epsilon}} = \frac{(x - \mu_x\mathbf{1}_d^\top)\Pi}{\sqrt{\sigma_x^2 + \epsilon}}. \tag{21}$$

Unlike RMSNorm, LayerNorm additionally contains an affine transformation with coordinate-wise scale γ and bias β. For a general orthogonal transformation Π, the affine parameters cannot be directly transformed to preserve:

$$\mathrm{LayerNorm}'(x\Pi) = \mathrm{LayerNorm}(x)\Pi, \tag{22}$$

because element-wise scaling does not generally commute with arbitrary coordinate mixing.

To support LayerNorm-based architectures, GateFlow implements the affine transformation via an equivalent linear-layer reformulation. Specifically, for the following linear projection:

$$y = \mathrm{LayerNorm}(x)W + b, \tag{23}$$

the affine parameters can be absorbed into the linear layer:

$$\widehat{W} = \mathrm{diag}(\gamma)W, \quad \widehat{b} = \beta W + b. \tag{24}$$

Thus, the remaining normalized representation can operate in the transformed domain while preserving functional equivalence.

Consequently, GateFlow supports LayerNorm-based decoder architectures by preserving the normalization invariance under orthogonal transformations and absorbing the affine parameters into adjacent linear transformations.

C.2 Adaptation to Post-RMSNorm

The main text describes GateFlow based on the Pre-RMSNorm architecture. This section presents a dedicated execution strategy for Post-RMSNorm architectures. Unlike Pre-RMSNorm architectures, where normalization is applied before the residual branch, Post-RMSNorm applies normalization after the residual connection. Therefore, the transformed representation cannot propagate directly through the normalization layer. To address this issue, GateFlow adopts a branch-and-merge transformed execution strategy for subsequent Post-RMSNorm decoder blocks.

First Decoder Block. For the first decoder block, GateFlow follows the secure authorization procedure described in the main text. After obtaining the attention output, the TEE constructs the residual representation:

$$h = x + o. \tag{25}$$

The TEE then performs the Post-RMSNorm operation in the original domain:

$$z = \mathrm{RMSNorm}(h) = \gamma \odot \frac{h}{\mathrm{RMS}(h)}. \tag{26}$$

The normalized representation is subsequently mapped into the transformed domain:

$$z' = z\Pi, \tag{27}$$

where Π denotes the global propagation transformation. The representation z' is used as the input of subsequent decoder blocks.

Subsequent Decoder Blocks. After the first decoder block establishes the transformed representation, subsequent Post-RMSNorm decoder blocks operate directly in the transformed domain. Specifically, assume that the input representation is:

$$x' = x\Pi. \tag{28}$$

For a linear projection with the original weight matrix W, GateFlow constructs the transformed weight:

$$W' = \Pi^\top W \Pi. \tag{29}$$

The GPU can then perform the transformed-domain computation:

$$u' = x'W' = x\Pi\Pi^\top W\Pi = xW\Pi. \tag{30}$$

Therefore, the linear output preserves the propagation property:

$$u' = u\Pi, \tag{31}$$

where $u = xW$ denotes the corresponding original-domain output.

Since Π is orthogonal, the RMS statistics remain unchanged:

$$\mathrm{RMS}(u') = \mathrm{RMS}(u\Pi) = \mathrm{RMS}(u). \tag{32}$$

Consequently, the normalized component satisfies:

$$\frac{u'}{\mathrm{RMS}(u')} = \frac{u}{\mathrm{RMS}(u)}\Pi. \tag{33}$$

For Post-RMSNorm blocks containing multiple linear transformations, including attention projections and feed-forward layers, the above transformation is applied to each linear layer independently. The nonlinear operations and residual connections are handled through the branch-and-merge strategy: the transformed branch maintains the propagation representation, while the merging operation combines intermediate results to preserve functional equivalence.

The scale parameters of RMSNorm are folded into the subsequent linear transformations to avoid introducing additional trusted computation. Therefore, after the initial authorization in the first decoder block, subsequent Post-RMSNorm decoder blocks can execute in the transformed domain while preserving the propagation property.

C.3 Compatibility with Rotary Position Embedding

Modern LLM architectures such as LLaMA employ Rotary Position Embedding (RoPE) to encode relative positional information in attention computation. Since RoPE introduces an additional position-dependent rotation on query and key representations, this section describes the corresponding adaptation of GateFlow for RoPE-based architectures.

The standard formulation in the main text considers generic attention computation, where the auxiliary transformations are applied to the query and key representations. For RoPE-based architectures, we apply the auxiliary transformations after the RoPE operation to preserve the original positional encoding procedure. Specifically, given the query and key projections:

$$Q = xW_q, \quad K = xW_k, \tag{34}$$

the standard RoPE operation is first performed:

$$Q_r = \mathrm{RoPE}(Q), \quad K_r = \mathrm{RoPE}(K). \tag{35}$$

After obtaining the RoPE-enhanced representations, GateFlow applies independent auxiliary orthogonal transformations:

$$Q' = Q_r \Pi_q, \quad K' = K_r \Pi_k, \tag{36}$$

where Π_q and Π_k are secret transformations maintained by the TEE. The transformed representations are then transferred to the TEE, which removes the auxiliary transformations:

$$Q_r = Q' \Pi_q^\top, \quad K_r = K' \Pi_k^\top. \tag{37}$$

The TEE then performs the attention computation using the recovered RoPE-enhanced representations:

$$S = \mathrm{Softmax}\left(\frac{Q_r K_r^T}{\sqrt{d_k}} + M_{\mathrm{causal}}\right), \tag{38}$$

where M_{causal} denotes the standard causal attention mask used in decoder-only LLMs.

For multi-head attention, we apply the above procedure independently to each attention head after reshaping the hidden representation into head dimensions. Therefore, the auxiliary transformations operate within individual head spaces and do not modify the standard multi-head attention computation.

After the first decoder block establishes the transformed propagation representation, subsequent decoder blocks follow the same propagation mechanism as described in the main protocol. The RoPE-specific adaptation only changes the position where the auxiliary query/key transformations are applied and does not affect the security design or transformed-domain propagation of GateFlow.

C.4 Compatibility with KV Cache

Modern autoregressive LLM inference relies on the KV cache mechanism to avoid repeatedly computing historical key and value representations during token generation. This section discusses GateFlow's compatibility with KV-cache-based decoding. During autoregressive decoding, the attention computation for the t-th generated token only requires the newly generated query representation and the cached key-value states:

$$S_t = \mathrm{Softmax}\left(\frac{Q_t [K_1, \ldots, K_t]^T}{\sqrt{d_k}}\right). \tag{39}$$

Without KV caching, the model must recompute all historical key and value projections for every generated token, adding unnecessary computational overhead. By maintaining the KV cache, only the current token requires additional projection computation. Unlike the key and value representations, the query representation directly determines the attention distribution when interacting with cached key states. Therefore, GateFlow protects the query projection while leaving the key and value projections unchanged, enabling efficient KV-cache reuse. Specifically, the query projection is transformed as:

$$W_q' = W_q \Pi_q, \tag{40}$$

where Π_q is a secret orthogonal transformation maintained by the TEE. The key and value projections remain unchanged:

$$W_k' = W_k, \quad W_v' = W_v. \tag{41}$$

During inference, the GPU computes the transformed query representation:

$$Q' = X W_q' = Q \Pi_q, \tag{42}$$

while directly generating and maintaining the key-value cache:

$$K = X W_k, \quad V = X W_v. \tag{43}$$

The transformed query is then transferred to the TEE, which removes the auxiliary transformation:

$$Q = Q' \Pi_q^\top. \tag{44}$$

The TEE subsequently computes the attention weights:

$$S = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right). \tag{45}$$

Since the query representation is unavailable to the GPU in its original coordinate system, the adversary cannot reconstruct the attention distribution from the cached key states alone. Meanwhile, keeping the key and value projections unchanged allows the KV cache to be stored and reused directly on the GPU without additional transformation, encryption, or TEE-side recovery. Therefore, during autoregressive decoding, GateFlow requires secure processing only of the current query representation, while the standard KV cache mechanism efficiently reuses historical key-value states. The secure authorization overhead is independent of the number of generated tokens.

With KV caching, the attention computation complexity is reduced from the prefill complexity:

$$O(L^2 d) \tag{46}$$

to the per-token decoding complexity:

$$O(Ld), \tag{47}$$

where L denotes the number of cached historical tokens. By avoiding additional KV-cache transformations and reusing the GPU-side cache, GateFlow preserves the efficiency advantages of propagation-based inference while supporting practical autoregressive generation.

C.5 Compatibility with Multi-Head Attention

Modern LLM architectures, such as LLaMA, use multi-head attention, where the hidden representation is split into multiple attention heads. In the main text, we present the secure attention procedure using a single attention head for clarity. This section extends the proposed mechanism to the standard multi-head attention setting.

Let H denote the number of attention heads and d_h denote the dimension of each head. The query, key, and value representations are first reshaped as:

$$Q = [Q_1, \ldots, Q_H], \quad K = [K_1, \ldots, K_H], \quad V = [V_1, \ldots, V_H], \tag{48}$$

where each head representation satisfies:

$$Q_h, K_h, V_h \in \mathbb{R}^{L \times d_h}. \tag{49}$$

For each attention head, GateFlow independently applies auxiliary orthogonal transformations:

$$Q'_h = Q_h \Pi_{q,h}, \quad K'_h = K_h \Pi_{k,h}, \tag{50}$$

where $\Pi_{q,h}$ and $\Pi_{k,h}$ are secret transformations maintained by the TEE. The transformed representations are sent to the TEE, which recovers the original head-wise query and key representations:

$$Q_h = Q'_h \Pi_{q,h}^{\top}, \quad K_h = K'_h \Pi_{k,h}^{\top}. \tag{51}$$

The TEE independently computes the attention matrix for each head:

$$S_h = \mathrm{Softmax}\left(\frac{Q_h K_h^T}{\sqrt{d_h}}\right), \tag{52}$$

and securely obtains the corresponding head output:

$$O_h = S_h V_h. \tag{53}$$

After all heads are processed, the head outputs are concatenated following the standard Transformer architecture:

$$O = \mathrm{Concat}(O_1, O_2, \ldots, O_H). \tag{54}$$

The concatenated attention output is then passed through the output projection layer:

$$o = OW_o + b_o. \tag{55}$$

Since each attention head independently preserves the same confidentiality property as the single-head case, extending GateFlow from single-head to multi-head attention does not introduce additional security assumptions. The auxiliary transformations only operate within individual head dimensions and are compatible with standard Transformer head partitioning.

D SECURITY ANALYSIS OF GATEFLOW

We analyze the confidentiality of GateFlow against an adversary that fully controls the untrusted GPU and host software. The adversary can observe all GPU-visible intermediate representations and encrypted messages during inference, but cannot access any secret information maintained inside the TEE, including the auxiliary transformations, the global propagation transformation, and the RLWE secret key.

Theorem 1 (Non-recoverability of Query and Key Representations). The GPU adversary cannot uniquely recover the original query and key representations or the corresponding secret transformations from the available GPU-visible observations.

Proof. In the first decoder block, GateFlow applies two independent secret orthogonal transformations to the query and key representations:

$$Q' = Q\Pi_q, \quad K' = K\Pi_k, \tag{56}$$

where Π_q and Π_k are independently generated orthogonal transformations and are only available to the TEE.

The adversary may observe GPU-visible information, including transformed representations, transformed model parameters, and inference-time inputs. However, recovering the original query and key representations requires identifying the hidden coordinate transformations:

$$Q = Q'\Pi_q^\top, \quad K = K'\Pi_k^\top. \tag{57}$$

Since Π_q and Π_k are unknown independent transformations, the observed representations do not provide a unique correspondence between the transformed coordinate system and the original coordinate system. Specifically, multiple pairs of plaintext representations and orthogonal transformations can produce the same GPU-visible observations. Therefore, the original coordinate representations cannot be uniquely identified.

It is worth noting that orthogonal transformations preserve rotation-invariant statistics:

$$Q'Q'^T = QQ^T, \quad K'K'^T = KK^T. \tag{58}$$

This invariant information reveals only geometric properties independent of the hidden coordinate system and is insufficient to reconstruct the original query and key representations or recover the secret transformations.

Therefore, although the GPU may obtain transformation-invariant information from the observed representations, it cannot recover the plaintext query and key coordinates or the secret transformations required for reconstruction. □

Theorem 2 (Confidentiality of Attention Weights). The attention weights computed in GateFlow are not revealed to the GPU.

Proof. The TEE reconstructs the plaintext query and key representations and computes:

$$S = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right). \tag{59}$$

By Theorem 1, the GPU cannot obtain the plaintext query and key representations needed to reproduce the attention computation.

Moreover, GateFlow never releases the plaintext attention matrix S. Instead, the TEE encrypts S using Coef-RLWE before outsourcing the subsequent linear operation. The security of Coef-RLWE relies on the hardness of the RLWE problem. Without the secret key s, the GPU cannot recover the encrypted attention weights from the ciphertexts. Therefore, the attention matrix remains confidential during outsourced computation. □

Theorem 3 (Confidentiality of Attention Output). The GPU cannot recover the plaintext attention output during ciphertext evaluation.

Proof. Let:

$$M = VW_o. \tag{60}$$

The GPU performs ciphertext–plaintext evaluation:

$$\mathrm{Eval}(\mathrm{Enc}(S), M) = \mathrm{Enc}(SM). \tag{61}$$

Due to the semantic security of RLWE-based encryption, the evaluated ciphertext does not reveal the plaintext product SM to the GPU.

The TEE is the only entity that can decrypt the returned ciphertext and obtain:

$$o = \mathrm{Dec}(\mathrm{Enc}(SM)) + b_o. \tag{62}$$

Therefore, the attention output o is never exposed to the GPU. □

Theorem 4 (Confidentiality of the Propagated Representation). The transformed hidden representation generated by GateFlow does not reveal the plaintext residual representation.

Proof. After obtaining the attention output, the TEE computes:

$$h' = (x + o)\Pi, \tag{63}$$

where Π is the secret propagation transformation.

The GPU may observe the transformed representation h' and the input representation x. However, according to Theorem 3, the attention output o is not available to the GPU. Therefore, the GPU cannot construct the plaintext residual representation:

$$x + o. \tag{64}$$

Even with the observation of h' and x, recovering o requires knowledge of the secret transformation Π and the hidden residual state simultaneously:

$$x + o = h'\Pi^T. \tag{65}$$

Since neither o nor Π is available to the GPU, the observed transformed representation does not reveal the original residual state. Consequently, GateFlow prevents exposure of the recoverable authorization relationships used in previous transformation recovery attacks. □

E Security and Correctness Analysis of Coef-RLWE

E.1 Instantiation Details of Coef-RLWE

This section provides additional implementation details of Coef-RLWE omitted from the main text. The proposed Coef-RLWE follows the standard RLWE encryption framework and only utilizes the ciphertext–plaintext linear evaluation property required by attention outsourcing. Let

$$R_q = \mathbb{Z}_q[X]/(X^N + 1) \tag{66}$$

be the polynomial ring used by Coef-RLWE. The secret key is sampled from a small secret distribution:

$$s \leftarrow \chi_s, \tag{67}$$

where χ_s denotes the secret-key distribution over R_q. The encryption noise is independently sampled from a small error distribution:

$$e \leftarrow \chi_e. \tag{68}$$

In practice, discrete Gaussian or centered binomial distributions can be used for χ_s and χ_e, following conventional RLWE-based encryption instantiations.

For a plaintext polynomial $m(X)$, the encryption algorithm introduces fresh randomness for every encryption operation. Specifically, random polynomials are sampled as:

$$r \leftarrow R_q, \quad e_1, e_2 \leftarrow \chi_e. \tag{69}$$

Using the public RLWE parameters, the ciphertext is generated as:

$$c = (c_0, c_1), \tag{70}$$

where the ciphertext components satisfy:

$$c_0 + c_1 s = \Delta m(X) + e_{\text{enc}}. \tag{71}$$

Here, Δ denotes the fixed-point encoding scale and e_{enc} represents the accumulated encryption noise. This relation underpins the correctness analysis in Appendix E.3.

During outsourced attention computation, Coef-RLWE does not require ciphertext–ciphertext multiplication. The GPU only performs linear combinations of ciphertexts:

$$c^{out} = \sum_k p_k c_k, \tag{72}$$

where p_k are plaintext coefficients derived from the encoded matrix M. Because RLWE encryption is linear under plaintext scaling and addition, the evaluated ciphertext remains an RLWE encryption of the corresponding linear combination:

$$c^{out} = \text{Enc}\left(\sum_k p_k m_k(X)\right). \tag{73}$$

Therefore, the proposed Coef-RLWE only requires the additive and ciphertext–plaintext linear properties of RLWE encryption, while avoiding the additional complexity introduced by general-purpose homomorphic operations.

E.2 Security Definition of Coef-RLWE

Coef-RLWE relies on the hardness of the Ring Learning With Errors (RLWE) problem (Lyubashevsky et al., 2010), which underpins the security of RLWE-based encryption schemes. This section defines the security assumption and security guarantee of Coef-RLWE for outsourced attention computation.

RLWE Assumption. Let

$$R_q = \mathbb{Z}_q[X]/(X^N + 1) \tag{74}$$

denote the polynomial ring, where q is the ciphertext modulus and N is the polynomial degree. Given a secret polynomial

$$s \leftarrow \chi_s, \tag{75}$$

an RLWE sample is defined as:

$$(a, b = a \cdot s + e) \in R_q^2, \tag{76}$$

where a is uniformly sampled from R_q and $e \leftarrow \chi_e$ is sampled from a small error distribution.

The RLWE assumption states that, for properly selected parameters, the following two distributions are computationally indistinguishable:

$$(a, a \cdot s + e) \tag{77}$$

and

$$(a, u), \tag{78}$$

where u is uniformly sampled from R_q. Therefore, a probabilistic polynomial-time adversary without knowledge of the secret key cannot distinguish an RLWE sample from a random sample with non-negligible probability.

IND-CPA Security Definition. Based on the RLWE assumption, Coef-RLWE provides semantic security under the standard indistinguishability under chosen-plaintext attack (IND-CPA) model (Brakerski et al., 2014). In this security model, the adversary obtains the public encryption parameters and ciphertexts generated by the encryption algorithm, while the TEE-held secret key remains unavailable.

The IND-CPA experiment is defined as follows. The challenger first generates the public parameters and secret key through $\text{KeyGen}(1^\lambda)$. The adversary receives the public parameters and outputs two equal-length plaintext messages m_0 and m_1. The challenger randomly selects $b \in \{0, 1\}$ and returns:

$$c^* = \text{Enc}(m_b). \tag{79}$$

The adversary outputs a guess b'. The advantage of the adversary is defined as:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{IND-CPA}} = \left| \Pr[b' = b] - \frac{1}{2} \right|. \tag{80}$$

Coef-RLWE is IND-CPA secure if, for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{IND-CPA}} \leq \mathrm{negl}(\lambda), \tag{81}$$

where λ denotes the security parameter and $\mathrm{negl}(\cdot)$ denotes a negligible function.

Security of Outsourced Attention Computation. In GateFlow, the attention matrix S is encoded into RLWE plaintext polynomials and encrypted inside the TEE:

$$c_k = \mathrm{Enc}(m_k(X)). \tag{82}$$

The GPU only performs ciphertext–plaintext linear evaluation:

$$c^{out} = \mathrm{Eval}(\mathrm{Enc}(S), M) = \mathrm{Enc}(SM), \tag{83}$$

without accessing the plaintext attention matrix S or the intermediate product SM.

Since the evaluation operation only performs linear combinations of RLWE ciphertexts, the resulting ciphertext remains a valid RLWE encryption of the corresponding plaintext result. Therefore, the confidentiality of outsourced attention computation follows from Coef-RLWE's IND-CPA security. Without the secret key maintained inside the TEE, the GPU adversary cannot recover the plaintext attention matrix S or the attention output intermediate SM from the observed ciphertexts.

E.3 Correctness Analysis of Coef-RLWE

The correctness of Coef-RLWE follows from the linearity of RLWE encryption under ciphertext–plaintext evaluation. Specifically, the evaluation operation preserves the underlying plaintext linear computation while introducing only the accumulated encryption noise.

For an encrypted polynomial $m_k(X)$, the RLWE ciphertext satisfies:

$$c_{0,k} + c_{1,k}s = \Delta m_k(X) + e_k. \tag{84}$$

During ciphertext–plaintext evaluation, the GPU computes:

$$c_j^{out} = \sum_{k=1}^{L} p_{kj} c_k. \tag{85}$$

Substituting the RLWE ciphertext relation gives:

$$\begin{aligned} c_{0,j}^{out} + c_{1,j}^{out} s &= \sum_{k=1}^{L} p_{kj}(c_{0,k} + c_{1,k}s) \\ &= \Delta \sum_{k=1}^{L} p_{kj} m_k(X) + \sum_{k=1}^{L} p_{kj} e_k. \end{aligned} \tag{86}$$

Let the evaluated plaintext polynomial be defined as:

$$m_j^{out}(X) = \sum_{k=1}^{L} p_{kj} m_k(X). \tag{87}$$

Therefore, the evaluated ciphertext remains a valid RLWE ciphertext:

$$c_{0,j}^{out} + c_{1,j}^{out} s = \Delta m_j^{out}(X) + e_j^{out}, \tag{88}$$

where e_j^{out} denotes the accumulated encryption noise.

The TEE decrypts the returned ciphertext:

$$\tilde{m}_j^{out}(X) = \mathrm{Dec}(c_j^{out}; s), \tag{89}$$

and removes the encoding scale through fixed-point decoding. Under the correctness condition analyzed in Appendix E.4, the decoded polynomial coefficients recover the fixed-point representation of the desired matrix multiplication:

$$\mathrm{Coeff}(\tilde{m}_j^{out}(X)) = [(\hat{S}\hat{M})_{1j}, \ldots, (\hat{S}\hat{M})_{Lj}]^T. \tag{90}$$

Since both the attention matrix and the plaintext matrix are encoded using the scale factor Δ, the recovered real-valued output is obtained by removing the accumulated fixed-point scale:

$$\mathrm{Dec}(\mathrm{Eval}(\mathrm{Enc}(S), M)) = SM + E, \tag{91}$$

where

$$E = E_{\mathrm{quant}} + E_{\mathrm{enc}} \tag{92}$$

contains the fixed-point quantization error and the residual encryption noise. We bound these error terms in Appendix E.4.

E.4 Noise Bound and Parameter Selection

This section analyzes the error accumulation and parameter selection of Coef-RLWE. Decryption correctness depends on controlling both the RLWE encryption noise and the fixed-point quantization error introduced during encoding.

Noise Propagation. For the k-th encrypted polynomial, the RLWE ciphertext satisfies:

$$c_{0,k} + c_{1,k}s = \Delta m_k(X) + e_k, \tag{93}$$

where e_k denotes the encryption noise.

During ciphertext–plaintext evaluation, the GPU computes:

$$c_j^{out} = \sum_{k=1}^{L} p_{kj} c_k, \tag{94}$$

where p_{kj} denotes the fixed-point encoded coefficient of the plaintext matrix M. Specifically,

$$p_{kj} = \lfloor \Delta M_{kj} \rceil. \tag{95}$$

The resulting noise term is accumulated linearly:

$$e_j^{out} = \sum_{k=1}^{L} p_{kj} e_k. \tag{96}$$

Therefore, the infinity norm of the accumulated noise is bounded by:

$$\|e_j^{out}\|_\infty \le \sum_{k=1}^{L} |p_{kj}| \|e_k\|_\infty. \tag{97}$$

Let

$$B_M = \max_{k,j} |M_{kj}|, \tag{98}$$

then the encoded coefficient satisfies:

$$|p_{kj}| \le \Delta B_M + \frac{1}{2}. \tag{99}$$

Therefore, assuming

$$\|e_k\|_\infty \le B_e, \tag{100}$$

the accumulated noise can be bounded as:

$$\|e_j^{out}\|_\infty \le L\left(\Delta B_M + \frac{1}{2}\right) B_e. \tag{101}$$

Since Coef-RLWE only performs ciphertext–plaintext linear evaluation and does not involve ciphertext–ciphertext multiplication, the noise increases linearly and does not require bootstrapping.

Decryption Correctness Condition. After decryption, the evaluated ciphertext satisfies:

$$\frac{c_{0,j}^{out} + c_{1,j}^{out} s}{\Delta} = m_j^{out}(X) + \frac{e_j^{out}}{\Delta}. \tag{102}$$

To correctly recover the encoded integer coefficients through rounding, the following condition should hold:

$$\|e_j^{out}\|_\infty < \frac{\Delta}{2}. \tag{103}$$

Under this condition, the RLWE noise does not affect the decoded integer coefficients. The recovered coefficients correspond to the fixed-point representation of the encrypted matrix multiplication.

Fixed-point Quantization Error. In addition to encryption noise, Coef-RLWE introduces numerical error due to fixed-point encoding. For a real-valued element a, the encoding operation is:

$$\hat{a} = \lfloor \Delta a \rceil. \tag{104}$$

The corresponding encoding error satisfies:

$$\left| a - \frac{\hat{a}}{\Delta} \right| \le \frac{1}{2\Delta}. \tag{105}$$

Since both the attention matrix and plaintext evaluation matrix are encoded using fixed-point representation, the recovered output contains the combined error:

$$E = E_{\text{quant}} + E_{\text{enc}}, \tag{106}$$

where E_{quant} denotes fixed-point quantization error and E_{enc} denotes the residual RLWE decryption error.

Increasing the scale factor Δ reduces quantization error but also increases the magnitude of encoded coefficients and, consequently, the noise requirement.

Parameter Selection. We select the parameters of Coef-RLWE based on the trade-off among security, correctness, and computational efficiency. The polynomial degree N determines the ring dimension and affects the security level, while the ciphertext modulus q determines the available noise budget.

The encoding scale Δ should satisfy the correctness requirement:

$$L\left(\Delta B_M + \frac{1}{2}\right) B_e < \frac{\Delta}{2}. \tag{107}$$

Meanwhile, the modulus q should be sufficiently large to prevent coefficient wrap-around during encryption and evaluation:

$$q > 2\left(\Delta \|m\|_\infty + \|e^{out}\|_\infty\right). \tag{108}$$

In practice, N, q, and Δ are selected jointly according to the target security level, sequence length, numerical precision, and inference efficiency requirements.

Block Encoding for Large Sequence Lengths. Polynomial encoding requires the polynomial degree to support the encoded coefficients. For an attention column of length L, the default setting requires:

$$N \ge L. \tag{109}$$

When $L > N$, the attention matrix is partitioned into multiple coefficient blocks. Each block contains at most N coefficients and is encoded and encrypted independently. The GPU performs ciphertext- plaintext evaluation for each block, and the TEE reconstructs the complete output by concatenating the decoded coefficients.

This block-based encoding preserves the same correctness condition because each ciphertext independently satisfies the RLWE decryption requirement. It only increases the number of ciphertexts proportionally to the number of blocks, introducing an implementation-level communication and computation trade-off.

F Efficient Orthogonal Transformation

The secure authorization procedure in GateFlow involves several orthogonal transformations, including the global propagation transformation Π and the auxiliary transformations Π_q and Π_k. Directly applying dense orthogonal matrices requires matrix–matrix multiplication with complexity $O(ld^2)$, which introduces considerable overhead inside the TEE. To reduce this cost, GateFlow adopts the T-Way Mixing implementation, which factorizes orthogonal transformations into structured Householder transformations and permutation operations.

A Householder transformation is defined as:

$$H_v = I - \frac{2vv^T}{v^T v}, \tag{110}$$

where v is the Householder vector. To preserve the mean component required by LayerNorm-compatible propagation, the Householder vector is constrained as:

$$v^T \mathbf{1}_d = 0. \tag{111}$$

Under this constraint, the Householder transformation preserves the all-ones direction:

$$H_v \mathbf{1}_d = \mathbf{1}_d - \frac{2v(v^T \mathbf{1}_d)}{v^T v} = \mathbf{1}_d. \tag{112}$$

For an input matrix X, multiplication with a Householder matrix can be rewritten as:

$$XH_v = X - 2\frac{(Xv)v^T}{v^T v}. \tag{113}$$

Thus, the matrix–matrix multiplication becomes matrix–vector multiplications, reducing the computational complexity from $O(ld^2)$ to $O(ld)$.

To enhance mixing capability in high-dimensional spaces, T-Way Mixing adopts a block-diagonal Householder structure:

$$\Lambda = \mathrm{diag}(H_1, H_2, \ldots, H_{d/k}), \tag{114}$$

where each block has size k. Each block follows the mean-preserving Householder construction above.

To further enhance mixing across dimensions, T-Way Mixing introduces permutation operations between transformation rounds. Each round applies the structured Householder transformation and then shuffles vector dimensions using a permutation matrix. Since permutation matrices satisfy:

$$P_i \mathbf{1}_d = \mathbf{1}_d, \tag{115}$$

the permutation operation also preserves the mean direction.

Therefore, the orthogonal transformations used in GateFlow can be represented as:

$$\Pi = \Lambda_1 P_1 \Lambda_2 P_2 \cdots \Lambda_r P_r. \tag{116}$$

Because each Householder transformation and permutation operation satisfies:

$$H_i \mathbf{1}_d = \mathbf{1}_d, \quad P_i \mathbf{1}_d = \mathbf{1}_d, \tag{117}$$

the resulting transformation maintains:

$$\Pi \mathbf{1}_d = \mathbf{1}_d. \tag{118}$$

Thus, the structured transformations preserve both orthogonality and the mean component required for propagation under normalization operations. During inference, the TEE performs these structured transformations instead of dense matrix multiplication. The preprocessing phase generates and stores only the transformation factors, adding no overhead during GPU-side decoder execution.

G DISCUSSION

Initial Single-Token Case and Security Enhancement. In autoregressive inference, the initial single-token case without cached historical key-value states is a degenerate attention scenario in which only one key is available. In this case, the attention matrix is deterministically given by

$$S = \mathrm{Softmax}(qk^T) = [1]. \tag{119}$$

Therefore, the attention output degenerates to the value projection:

$$o = VW_o + b_o. \tag{120}$$

In this special case, the attention score computation and the softmax operation provide no additional selectivity.

In normal autoregressive decoding, this case occurs only at the beginning of a new inference session. Once historical key-value states are introduced through the KV cache, the attention computation becomes non-degenerate:

$$S_t = \mathrm{Softmax}\left(\frac{Q_t[K_1, \ldots, K_t]^T}{\sqrt{d_k}}\right), \tag{121}$$

and the standard secure execution procedure of GateFlow is applied.

However, an adversary may intentionally create multiple independent inference sessions and issue only one-token queries in each session. In this case, the attention matrix remains fixed as $S = [1]$, making Coef-RLWE-based attention weight protection unnecessary. By repeatedly collecting the resulting observations, the adversary may attempt to accumulate information about the value projection output. Although a single observation provides only limited constraints, repeated independent single-token queries can gradually increase the available information. This attack relies on abnormal query patterns rather than the standard autoregressive inference process. Practical system-level mechanisms such as session management and query-rate limitation can control it.

To provide cryptographic protection against this potential attack without introducing additional assumptions, GateFlow further introduces an optional output-projection obfuscation mechanism. Specifically, the output projection is transformed as:

$$W'_o = W_o\Pi_o, \tag{122}$$

where Π_o is an independent secret orthogonal transformation maintained by the TEE.

For the initial single-token case, since the attention matrix is fixed as $S = [1]$, GateFlow adopts a lightweight execution branch. The GPU computes the obfuscated output projection:

$$M' = VW'_o = VW_o\Pi_o, \tag{123}$$

and transfers the result to the TEE. The TEE removes the auxiliary output transformation:

$$M = M'\Pi_o^\top = VW_o. \tag{124}$$

The attention output is then securely generated inside the TEE:

$$o = M + b_o, \tag{125}$$

and the transformed propagation representation is obtained as:

$$h' = (x + o)\Pi. \tag{126}$$

The security of this enhancement follows from the secrecy of Π_o. Although the GPU observes $M' = VW_o\Pi_o$, it cannot recover the plaintext output $M = VW_o$ without knowledge of Π_o. Consequently, the adversary cannot obtain the attention output o or construct the plaintext residual representation $x + o$. Therefore, even under repeated single-token queries, the adversary cannot establish the input-output transformation relationship required to recover the global propagation transformation Π.

Attack Identifiability. The transformation recovery attack relies on the identifiability of the underlying linear system. Specifically, given the observation model

$$Y = ZB, \tag{127}$$

the hidden transformation parameters can be uniquely recovered when

$$rank(Z) = 2d. \tag{128}$$

When the observation matrix is rank-deficient, the recovery problem becomes underdetermined, and multiple transformation matrices may satisfy the observed constraints. However, multiple solutions do not necessarily imply attack failure. Different solutions may induce equivalent transformations over the observed representation subspace.

Therefore, attack effectiveness depends on three factors: (i) the rank and diversity of collected observations, (ii) numerical stability under noisy measurements, and (iii) whether the recovered transformation generalizes to unobserved input spaces. In practice, adaptive query selection can increase observation diversity and improve identifiability of the hidden transformation.

Model Recovery After Transformation Recovery. The proposed attack primarily targets the confidentiality of the hidden authorization transformations. Once the adversary recovers the transformation matrices, it can remove the protection layer introduced by the propagation mechanism. Specifically, for a transformed weight matrix following the SLIM construction:

$$W' = W\Pi, \tag{129}$$

the recovered transformation enables the adversary to reconstruct the original weight matrix as:

$$W = W'\Pi^{\top}. \tag{130}$$

For decoder blocks that generate multiple transformed parameters using the same propagation transformation, the adversary can apply the recovered inverse mapping to restore the corresponding parameters to the original model coordinate system. This process applies to linear projections, including query, key, value, output projections, and feed-forward network layers, depending on the specific transformation strategy adopted by the protected inference framework.

After removing the transformation layer, the remaining model components can be reconstructed following the standard model extraction pipeline. Specifically, the adversary can recover the original parameter representation, restore the normal inference execution path, and deploy the recovered model independently without relying on the original TEE authorization procedure.