



Temple Enclave AI: Tenant-Encrypted, Mutually Private LM Engine

Two Solutions for Confidential Multi-Tenant LLM Serving — TEMPLE-KEY-TRUST-HOST (isolated from other tenants, zero overhead) and TEMPLE-MASK-TRUST-NOBODY (isolated from the host too, 25 ms/token)

Milind K. Patil

Syncaissa Systems Inc, USA.

To Cite this Article: Milind K. Patil, “Temple Enclave AI: Tenant-Encrypted, Mutually Private LM Engine”, *International Journal of Scientific Research in Engineering & Technology*, Volume 06, Issue 05, September-October 2026, PP:46-58.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the [Creative Commons Attribution License](#); Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: Two solutions for serving one open-weight language model to many tenants. **TEMPLE-KEY-TRUST-HOST.** Every tenant speaks to the model in its own secret private language: a per-tenant key $\kappa = (\pi, Q)$ —a homophonic permutation π of the vocabulary and a secret orthogonal rotation Q of the residual stream. A pretrained transformer is exactly equivariant to Q once LayerNorm is folded into RMSNorm, so the host runs a rotated copy, computes bit-for-bit the base model’s function, and sees only permuted symbols and rotated vectors. No training, no runtime cost; tenants cannot read one another (0% cross-tenant recovery against an eight-attack suite). Its limit is a theorem: rotated copies of one body are mutually alignable, so the party that executes the model can read. **TEMPLE-MASK-TRUST-NOBODY.** The host therefore never computes on readable numbers: every linear map runs on the host over fixed-point activations one-time-padded on $\mathbb{Z}_{2^{64}}$; every nonlinear step, the KV cache and the unembedding run on the tenant; the host returns $W(x + r) + b$ and the tenant subtracts a precomputed Wr . The masks are prompt-independent preprocessing, supplied by a second cheap server at another provider (**TEMPLE-DEALER-TRUST-NO-TWO**) or by the host itself under exact homomorphic encryption when idle, with a stateless clerk parking the batches (**TEMPLE-MINT-TRUST-NONE**). Verified by eighteen self-checking tests and deployed on a rented GPU, a native C++/CUDA host and tenant answer at 24–25 ms per token (40 tokens/s), token for token the plain model’s answer; with the host minting its own masks, 23 ms per token after 13.6 minutes of idle GPU per 8,192 tokens. Twelve alternatives are recorded with the measurement or theorem that set them aside; a reference implementation reproduces every number.

Keywords: Confidential inference, multi-tenant LLM serving, keyed-equivariant inference, masked split inference, one-time pad, homomorphic encryption, Beaver preprocessing.

Table of Contents

The problem this paper addresses.

Whenever an organization sends prompts to a hosted language model, the same question stands in the way: *will our confidential material—contracts, financial figures, patient records, source code—be readable by anyone other than us?* Two parties are of concern: the *other tenants* who share the same model and infrastructure, and the *operator/LLM API host* itself, whose systems, logs and staff handle every prompt in the clear. Transport encryption addresses neither; contractual assurances address neither technically. This paper gives two constructions that settle the question by design rather than by policy: **TEMPLE-KEY-TRUST-HOST** makes a tenant’s traffic unreadable to every other tenant and to everything around the model at no runtime cost, and **TEMPLE-MASK-TRUST-NOBODY** additionally makes it unreadable to the operator/LLM API host—the machine running the model never holds a readable number—while the answers remain, token for token, those of the unprotected model.

Easy Explanation (for a non-specialist)

Easy Explanation (Solution 1) — TEMPLE-KEY-TRUST-HOST: the operator/LLM API host can still read. Every customer talks to the shared model in its own secret language: the words are relabelled and the model’s inner coordinate system is tilted, under a key only that customer holds. Other customers, the network, the logs and the storage see gibberish; the model answers exactly as before, with no training and no slowdown. But the company that *runs* the model holds the relabelled copy and, we prove, can always work the relabelling out. Solution 1 protects you from everyone *except* the operator/LLM API host.

Easy Explanation (Solution 2) — TEMPLE-MASK-TRUST-NOBODY: not even the operator/LLM API host. Your machine adds fresh random noise to every number before sending it—a one-time pad—and the powerful machine multiplies the noisy numbers by its weights and sends the result back. What the noise turns into is pre-computed elsewhere, so your machine subtracts it and is left with the true answer: either by a cheap second machine at a *different* company that holds

the same public weights but never hears a word from you (TEMPLE-DEALER-TRUST-NO-TWO), or by the powerful machine itself, on noise you hand it in a sealed envelope (exact homomorphic encryption), in the background for thousands of future words at once (TEMPLE-MINT-TRUST-NONE). Either way no machine but yours can read your prompt or your answer, and the answer is word for word what the unprotected model would say.

I. INTRODUCTION

Large language models are served as a shared resource: one model body answers prompts from many tenants, and the pipeline plaintext → tokenizer → embeddings → transformer body → unembedding → plaintext is legible at every arrow to the operator, to its logs, and to anyone who compromises them. Transport encryption protects only the wire. The genuinely private alternatives—homomorphic encryption (Hao et al. 2022; Pang et al. 2024; Lu et al. 2025; Chen et al. 2022), multi-party computation (Li et al. 2023), trusted hardware (Costan and Devadas 2016)—are orders of magnitude slower than plaintext, need special hardware, or both.

This paper gives two solutions and keeps them apart. Solution 1, TEMPLE-KEY-TRUST-HOST (Sections 3–5), starts from the observation that *the encrypted form of natural language is itself a language*: a keyed bijection of English is one more language a pretrained model can operate in natively, and each tenant can have its own. Rotation equivariance of transformers is known from the quantization literature (Ashkboos et al. 2024; Liu et al. 2024); we use it as a per-tenant secret, combine it with a homophonic vocabulary and tenant-held tables, and attack the result with a purpose-built suite. It isolates tenants from one another and from everything around the model at zero cost, and it has a proved limit: the executing party can always align rotated copies (Theorem 2). Solution 2, TEMPLE-MASK-TRUST-NOBODY (Sections 7–8), removes that party’s ability to read by never giving it a readable number: the linear maps run on the host over one-time-padded activations, the nonlinear steps on the tenant, and the masks come from a second small server or from the host itself under exact encryption. Solution 2 runs inside Solution 1. Every name in this paper ends in its trust statement (TRUST-HOST, TRUST-NOBODY, and for the mask supplies TRUST-NO-TWO and TRUST-NONE), so a reader chooses by the name alone; TEMPLE-CIPHER, the online-encryption precursor of Solution 2, is an option considered. Table 1 sets the two solutions side by side; Section 11 records, one paragraph each, the alternatives that were built or analysed and set aside.

The two solutions at a glance. “Reads the data” means the party can recover prompts or answers from what it holds or observes.

	TEMPLE-KEY-TRUST-HOST (Sec. 3)	TEMPLE-MASK-TRUST-NOBODY (Sec. 7)
Mechanism	per-tenant secret key (π, Q): vocabulary map + rotation of the residual stream; the host runs a rotated copy	the tenant one-time-pads every activation; the host does the linear maps on padded integers; the tenant does the nonlinear steps and the unembedding
Other tenants, network, logs, storage, operators read the data?	no	no
The LLM host (executor) reads the data?	yes (Theorem 2)	no: one-time pad, given a mask supply the host does not control
Mask supply	—	TEMPLE-DEALER-TRUST-NO-TWO (second small server, other provider) or TEMPLE-MINT- TRUST-NONE (the host itself, under exact encryption, offline)
Training / model change; runtime cost	none; zero	none; plaintext-speed arithmetic, 96 round trips per token
Measured	exact on GPT-2, Qwen2.5-0.5B, a Llama-architecture model	Qwen2.5-0.5B, all 24 layers: exact answers, host view uniform, 25 ms per token

1.1 Contributions

1. **Solution 1 — TEMPLE-KEY-TRUST-HOST, tenant isolation.**
2. **The construction** (Section 3): a zero-training per-tenant key (π, Q) with an exactness theorem (Theorem 1), an exact LayerNorm-to-RMSNorm folding, a homophonic vocabulary map whose $-\log k_t$ bias keeps server-side sampling exactly unbiased (Proposition 2), and tenant-side tables so that token identities never leave the tenant.
3. **The governing invariant and the alignability theorem** (Sections 2, 6): *wherever the knowledge of a tenant’s language lives, that party can read the tenant’s data*; rotated copies of one body are mutually alignable (Theorem 2), so Solution 1 cannot hide data from the executing host—the precise statement of its limit and the reason Solution 2 exists.
4. **An attacker suite and measurements** (Sections 4, 5): eight attacks; exact isolation between tenants; 0% recovery by a host without canonical weights and exact recovery of Q by a host with a public base (Proposition 3); decipherment of a deterministic cipher versus none for a homophonic one; exactness on GPT-2, Qwen2.5-0.5B-Instruct and a Llama-architecture model; zero overhead, 32-byte keys, host-side re-keying.
5. **Solution 2 — TEMPLE-MASK-TRUST-NOBODY, full isolation, including from the LLM host.**

6. **The masked split protocol** (Section 7): linear maps on the host over one-time-padded fixed-point activations on $\mathbb{Z}_{2^{64}}$, nonlinear steps on the tenant, masks as prompt-independent preprocessing; exact integer kernels on GPUs; two separately authenticated servers and a stateless host; eighteen self-checking tests; deployment across providers; and the engineering from 3.6 s to **0.025 s per token** with a native C++/CUDA host and tenant (Section 7.4).
7. **TEMPLE-MINT-TRUST-NONE** (Section 8): the mask wall (Proposition 6); exact minting by the host under BFV with CRT recombination onto the unchanged protocol; token batching at $2 \times \text{parameters}/8192$ scalar operations per token; a native minting kernel on raw ciphertexts; a stateless, weightless clerk; a whole 8,192-token batch minted on the host's GPU in 13.6 minutes—99 ms of offline time per token—and the prompt answered on those packs with no second party.
8. **The options considered** (Section 11): twelve alternatives, built or analysed, with the measurement or theorem that set each aside.

II. THREAT MODEL AND GOVERNING PRINCIPLE

2.1 Parties

A tenant A owns prompts and completions. The host operates the shared serving system and holds whatever weights it is given. Other tenants B, C, ... use the same system. A network/log adversary observes everything in transit or at rest outside the tenant. Table 2 states what each party holds.

Who holds what. W : canonical (base) weights; W_Q : weights rotated by Q ; E_A : tenant A's vocabulary tables.

Party	κ_A	E_A	W_Q	W	Goal of the design
Tenant A			—	—	reads own data only
Other tenant B	κ_B	E_B	W_{Q_B}	?	reads nothing of A
Host in the clear, private base	—	—		—	<i>breaks</i> (Thm. 2)
Host in the clear, public base	—	—			<i>breaks</i> (Prop. 3)
Host under Solution 2	—	—		?	reads nothing (one-time pad, Sec. 7)
Host with attested enclave	—	—	sealed	?	reads nothing (hardware trust)
Network / logs	—	—	—	?	reads nothing of A

2.2 Governing principle

The design rests on one sentence: *wherever the knowledge of a tenant's language lives, that party can read the tenant's data*. A model fine-tuned on tenant A's ciphertext learns A's language exactly as it learned Japanese, and can then translate it for anyone. Therefore:

Requirement 1 (Cipher-blind body). *No tenant key, and no tenant ciphertext, ever enters the shared weights. All tenant-specific language knowledge lives in tenant-held components.*

This makes tenant isolation structural rather than a policy: tenant B holds nothing of κ_A , and the body learned nothing of either.

III. SOLUTION 1: TEMPLE-KEY-TRUST-HOST — KEYED-EQUIVARIANT INFERENCE, TENANT ISOLATION AT ZERO COST

We use the row-vector convention: activations $h \in \mathbb{R}^{1 \times d}$, linear maps $h \mapsto hW + b$.

3.1 Rotation equivariance, made exact

A pre-norm transformer block (Vaswani et al. 2017) reads the residual stream through a normalization layer and writes back through a linear projection. RMSNorm (Zhang and Sennrich 2019) satisfies $\text{RMS}(hQ) = \text{RMS}(h)Q$ for any orthogonal Q . LayerNorm (Ba et al. 2016) does not, because it subtracts the mean, which is not rotation invariant. GPT-2 (Radford et al. 2019) uses LayerNorm, so we first fold it away exactly.

Proposition 1 (Exact LayerNorm folding). *In a network whose only reads from the residual stream pass through LayerNorm, (i) subtracting the row mean from every matrix that writes into the residual stream (token and position embeddings, attention and MLP output projections and their biases) leaves the network function unchanged and makes every residual vector zero-mean, after which LayerNorm's mean subtraction is the identity; and (ii) each LayerNorm's affine parameters (γ, β) fold into the following linear layer as $W' = \text{diag}(\gamma)W$, $b' = b + \beta W$. The result is an equivalent network whose norms are parameter-free RMSNorm.*

Proof sketch. LayerNorm is invariant to shifts along **1**, so mean-centring every write is invisible (i); $(\text{RMS}(h) \odot \gamma + \beta)W + b = \text{RMS}(h) \text{diag}(\gamma)W + (\beta W + b)$ (ii); the tied unembedding is untied and given the bias $\beta W_{\text{unembed}}^T$. $\square$

Theorem 1 (Exact rotation equivariance). *Let f be a transformer with RMSNorm (natively, or after Proposition 1) with token embedding E , position embedding P , per-block read matrices W^{in} (attention W_{QKV} , MLP W_1), write matrices W^{out} (attention W_O , MLP W_2) with biases b^{out} , and unembedding U . For any orthogonal $Q \in \mathbb{R}^{d \times d}$ define f_Q by $E \mapsto EQ$, $P \mapsto PQ$, $W^{\text{in}} \mapsto Q^T W^{\text{in}}$, $W^{\text{out}} \mapsto W^{\text{out}} Q$, $b^{\text{out}} \mapsto b^{\text{out}} Q$, $U \mapsto UQ$. Then f_Q computes exactly the same logits as f on every input, and every residual-stream activation of f_Q equals the corresponding activation of f right-multiplied by Q .*

Proof sketch. Induction over blocks: a read computes $\text{RMS}(hQ)Q^T W^{\text{in}} = \text{RMS}(h)W^{\text{in}}$, so scores, softmaxes and MLP activations are unchanged; a write adds (original write) Q , so the residual stays rotated; the unembedding cancels QQ^T . $\square$

This is the mechanism QuaRot and SpinQuant use to remove outliers (Ashkboos et al. 2024; Liu et al. 2024); here Q is drawn Haar-uniformly from $O(d)$ by a tenant and kept secret—a *key*—so every hidden state the host sees is expressed in a basis only the tenant knows.

3.2 The vocabulary map π and its homophonic extension

The second half of the key renames tokens. A permutation π of the vocabulary applied to the rows of E and U means the host receives token id $\pi(t)$ and emits $\pi(t')$; only the tenant maps back. A permutation is a substitution cipher and inherits all of its weaknesses (Section 5.4), so we extend it to a *homophonic* map: an expanded cipher vocabulary of $n_c > V$ ids in which plaintext token t owns $k_t \geq 1$ ids, with k_t proportional to t 's frequency in the tenant's own data, and each occurrence of t is encrypted to a uniformly random one of its k_t ids. This flattens the cipher-symbol distribution toward uniform.

Duplicated unembedding rows would split t 's probability mass k_t ways; a bias fixes it.

Proposition 2 (Unbiased cipher-space sampling). *Let the server's unembedding have, for every cipher id c of token t , the row $U[t]Q$ and the bias $b_c = b_t - \log k_t$. Then for every context, $\sum_{c \in \text{hom}(t)} p_{\text{server}}(c) = p_{\text{base}}(t)$; in particular, sampling a next cipher id from the server's distribution samples t with its base probability and then a uniform homophone.*

Proof. $\sum_{c \in \text{hom}(t)} \exp(\ell_t - \log k_t) = k_t \exp(\ell_t) / k_t = \exp(\ell_t)$, so the partition function is unchanged and each group sums to $p_{\text{base}}(t)$. $\square$

Greedy decoding log-sum-exps the cipher logits within each group on the tenant; sampling needs nothing—the host generates directly in cipher space.

3.3 Tenant-side embeddings

Nothing forces E , P and U to live on the host. In the *split* variant the tenant holds EQ , PQ and UQ , sends the vectors $(E[c] + P[p])Q$, receives the final hidden states, and unembeds locally. The host then holds only the rotated body; token identities never leave the tenant, so π and any homophony become purely local and the wire carries ordinary authenticated encryption. Section 5.3 shows why this variant is not optional.

3.4 Multi-tenant serving

Tenants $A, B, C, \dots$ each hold $[\pi, Q, E]$ and send ciphertext to a shared cipher-blind body; each costs one rotated copy of the body's weights (or, with a key-equivariant body, none). Two serving invariants must hold: attention is per-sequence, so cross-tenant *batching* is safe, but cross-tenant *prefix or KV-cache sharing* is not and must be disabled or keyed per tenant; and A 's ciphertext is routed only to A 's rotated copy. Key rotation is cheap—re-draw Q , re-sample homophones—so a leaked key exposes one epoch of traffic.

IV. THE ATTACKER SUITE

We evaluate every mechanism against the attacks in Table 3. All are implemented in the reference code.

Attacks, what the attacker holds, and what each targets.

#	Attack	Attacker holds	Targets
A1	Nearest-neighbour embedding inversion	vectors, public E	tokens from hidden states
A2	Orthogonal Procrustes on P (Schönemann 1966)	W_Q, W	Q via unpermuted parameter
A3	Linear handle on any block weight	W_Q, W	Q via $W_Q = Q^T W$
A4	Frequency-rank matching	traffic	π from unigram statistics
A5	Simulated-annealing substitution solver (Knight et al. 2006)	traffic, English n -grams	π from bigram statistics
A6	Duplicate-row clustering	server E	homophone groups from weights
A7	Homophone context clustering	traffic	homophone groups from contexts
A8	Cross-tenant “translate” prompt	κ_B , A 's ciphertext	A 's text via the model

V. EXPERIMENTS

Unless stated otherwise (Section 5.6), the Solution-1 experiments use GPT-2 small ($d = 768$, $V = 50,257$, 12 blocks) on CPU. Tenant keys are drawn from seeds. Traffic for the statistical attacks is Jane Austen's *Emma*, *Sense and Sensibility*, *Mansfield Park*, *Persuasion* and *Northanger Abbey* (876,280 GPT-2 tokens); the tenant's own frequency statistics come from *Pride and Prejudice*; the attacker's English model comes from five novels by other authors (Brontë, Eliot, Melville, Dickens; 1.56M tokens).¹

5.1 Exactness and what the host sees

Folding LayerNorm (Proposition 1) changes logits by at most 1.5×10^{-4} (float32 noise).

The keyed model, after the tenant un-permutes, differs from the base by at most 1.4×10^{-4} , and 25-token greedy generations are identical after decryption. Table 4 shows the host's view.

Tenant A's prompt as seen by each party (GPT-2 small).

Party	Observation
Tenant A	The secret project meeting is scheduled for
Host (ids)	35746 16256 44518 21682 44947 6163 31700
Host (ids via public tokenizer)	apest Indonesia patrolling frostYA selected Bie

Party	Observation
Host (layer-6 hidden states)	$\cos(h_{\text{base}}, h_{\text{host}}) \in [-0.02, 0.04]$ per position
Tenant A applying Q^T	$\cos = 1.0000$

5.2 Isolation between tenants

Tenant *B* decrypting *A*'s completion obtains params unexpl anomal nuanced Investorax...; feeding *A*'s ciphertext through *B*'s model yields fluent but unrelated text. In attack A8, *B* wraps *A*'s ciphertext of “Our merger with Acme closes on Friday; the price is 40 million dollars.” in “Translate the following text into English.”; *B*'s model reads the middle as uta Surrey Bryanandraitative... and answers “The following text is a translation of the original text.” *B* is stopped by absence of information: κ_A exists only on *A*'s side and never entered any weights.

5.3 Attacks on the weights

Without canonical weights (A1), nearest-neighbour matching of the host's input vectors against the public embedding table recovers 0/7 tokens (chance 2×10^{-5}). With canonical weights (A2, A3), the unpermuted position embedding gives a direct correspondence: orthogonal Procrustes recovers Q to 2.6×10^{-6} , after which un-rotating the host's embedding rows recovers π for 100% of a 2,000-token sample and decrypts *A*'s completion perfectly. Moving *P* to the tenant closes this handle and opens no fewer than 48 others:

Proposition 3 (Canonical handles). *If the base weights W are public and the host holds W_Q , then for every read matrix $W^{\text{in}} \in \mathbb{R}^{d \times m}$ ($m \geq d$, full row rank) $Q = W^{\text{in}} (Q^T W^{\text{in}})^+$, and for every write matrix W^{out} of full column rank $Q = (W^{\text{out}})^+ W^{\text{out}} Q$: each block exposes four exact handles on Q .*

Measured on block 0 of GPT-2 small with tenant-side tables: W_{QKV} recovers Q to 3×10^{-7} , W_O to 10^{-4} , W_1 to 7×10^{-7} , W_2 to 8×10^{-8} , and with $\hat{Q}$ the attacker recovers 100% of the prompt.

Corollary 1 (Derivatives are handles). *If any public model shares a full-rank residual-stream read or write matrix with the base—a fine-tune, a distillation, a quantized re-release—Proposition 3 applies with that model in place of the base. A private base body must therefore have no public ancestor.*

Proposition 4 (The body is an oracle). *Let a party hold a body f trained to tolerate input noise— $\text{KL}(f(x) \parallel f(x + \delta)) \leq \epsilon$ for $\delta \sim \mathcal{N}(0, \sigma^2 \parallel x \parallel^2 I/d)$ on natural inputs—and be able to run it on chosen inputs. Given two noisy occurrence vectors $x_1 = e_{t_1} + \delta_1$ and $x_2 = e_{t_2} + \delta_2$, the party decides whether $t_1 = t_2$ by comparing $f(x_1)$ and $f(x_2)$ in a fixed context: robustness makes them agree when $t_1 = t_2$, and utility requires them to differ when $t_1 \neq t_2$. Hence vector-level noise cannot hide token identity from any party able to run the body, whatever the training.*

Proposition 4 rules out the family of embedding-perturbation defences (Feyisetan et al. 2020; Zhou et al. 2023) against the executor; Section 11 records our own attempt. Duplicate rows (A6): homophone rows in a server-held table are exact copies and cluster trivially, which is why the tables live on the tenant. Conclusion: rotation, permutation, tenant-side tables and homophony all hold against parties that do not hold the body—other tenants, logs, the network, unaligned embedding inversion—and all fall to whoever holds a rotated copy of a public base. The rotation is a transport and tenant-isolation key; secrecy from the host needs more (Section 6).

5.4 Attacks on the traffic

A deterministic cipher is decipherable (A5). Tenant *B* collects *A*'s ciphertext and runs a simulated-annealing substitution solver that maximizes bigram log-likelihood under an English model built from other authors (Table 5). Function words fall first; from 10k symbols the attacker reads “...I believe said...though pass famous where, so that none for said he...” for the truth “...to keep him from such thoughts; but when tea came, it was impossible for him not to say...”—a weak lower bound (a fixed 300k-step budget; a tuned solver would recover more). A $2 \times$ homophonic map removes the frequency-rank signature (A4) and gives the solver no foothold: 0.3%.

Simulated-annealing substitution solver (A5) versus ciphertext volume.

Cipher	Ciphertext (tokens)	Symbols solved cover	Tokens recovered
Plain permutation	10,000	100%	29.6%
Plain permutation	40,000	97%	32.2%
Plain permutation	160,000	95%	14.3%
Plain permutation	876,280	93%	16.8%
Homophonic $2 \times$	160,000	19%	0.6%
Homophonic $2 \times$	876,280	13%	0.3%

Homophone context clustering (A7): homophones of one token share a context distribution, and clustering 876k tokens of $2 \times$ homophonic traffic by context merges sibling pairs for 7.4% of tokens in one round and 11.3% after four (purity 95–97%), raising the solver's recovery from 0.3% to 1.2%—a fraction that grows with volume, one more reason a fixed language must not be learnable from its traffic.

5.5 Reproducible demonstrations and the adversary scoreboard

Three self-checking programs accompany the construction. demo1_private_language.py keys GPT-2 small for three tenants and checks eleven claims (exactness for each, isolation between each pair, host sees gibberish; 11/11).

demo2_hacker_attacks.py plays fourteen scenarios of what an adversary who holds a given set of artifacts recovers: with logs and ciphertext only, or with another tenant’s key, nothing (ten scenarios safe); with the canonical weights or with any one tenant’s rotated copy plus their key, everything (three documented breaks, the content of Proposition 3 and Theorem 2); with enough traffic of a one-to-one cipher, a weakened result (the decipherment of Table 5). demo3_overhead_and_keys.py measures the overhead (none), the key size (32 bytes), re-keying and the writer/reader capability split of Section 9. Transcripts are in the repository.

5.6 Model coverage

The identical keyed test—fold, rotate, permute, compare logits, decrypt the answer—is exact on GPT-2 small (OpenAI; LayerNorm, folded), Qwen2.5-0.5B-Instruct (Alibaba; RMSNorm) and SmolLM2-135M-Instruct (Llama architecture): maximum keyed logit difference 1.4×10^{-4} , 10^{-5} and 8.6×10^{-5} , answers identical to each base model, the host’s view of a Qwen prompt a mixture of scripts (“[CJK] ndx statutory_continuous panoramaStatic ...”). Keying the 630M-parameter model takes 14 s on two CPU cores. Closed-weight models (ChatGPT, Gemini) cannot be keyed: both solutions need the weights (Section 11).

VI. THE LIMIT OF SOLUTION 1 AND THE UNLEARNABILITY REQUIREMENT: A TENANT LANGUAGE MUST NOT BE A LANGUAGE

If the model can translate Japanese, why can it not translate tenant A’s language? Because Japanese is in its weights and not one sentence of A’s language is; the “grammar” is a random key held by A. But Section 5.4 shows the sharper truth: a fixed deterministic key produces English with every word renamed—same statistics—and the renaming can be inferred from enough text. Determinism plus fixity is a language, and a language is learnable.

Definition 1 (Learnable tenant language). A tenant encoding $\mathcal{E}_\kappa$ is *learnable* if there exists an algorithm that, from a sample of $\mathcal{E}_\kappa$ -encoded traffic and public knowledge of natural language (but not κ), recovers plaintext tokens with accuracy bounded away from chance as the sample grows.

Proposition 5 (Deterministic injective encodings are learnable). *If $\mathcal{E}_\kappa$ maps each token to a fixed symbol, every n -gram statistic of the traffic equals the corresponding plaintext statistic up to relabeling of symbols; hence the encoding is learnable by statistical decipherment given sufficient traffic (Knight et al. 2006), and it is learnable by any model trained on the traffic.*

Requirement 2 (Unlearnability). *A tenant language must not be a fixed, learnable language, and the shared body must never be trained on any tenant’s traffic. Standing test: given only ciphertext and the body, can anything produce plaintext? The answer must always be no.*

6.1 Meeting the requirement

Two families of mechanism satisfy Unlearnability. *Traffic-level*: rotate the key per session or per document, use non-deterministic (homophonic, or nonce-based) encoding so that no fixed symbol language exists on the wire, and keep the vocabulary tables on the tenant so that token identities never leave it; all of these are part of Solution 1 and cost nothing. *Body-level*: whatever the traffic looks like, the party that executes the body can read, and the reason is a theorem.

Theorem 2 (Mutual alignability of rotated copies). *Let $W_{Q_A} = Q_A^T W$ and $W_{Q_C} = Q_C^T W$ be two rotated copies of the same full-row-rank matrix W , with W unknown. Then $W_{Q_C} W_{Q_A}^+ = Q_C^T Q_A$. Consequently a host that holds any two tenants’ rotated bodies can map either tenant’s residual stream into the other’s basis, and a host that owns one tenant key (for instance by registering as a tenant) can read every other tenant, whether or not the base model is public.*

Proof. $W_{Q_A}^+ = W^+ Q_A$ for full row rank W , so $W_{Q_C} W_{Q_A}^+ = Q_C^T W W^+ Q_A = Q_C^T Q_A$. $\square$

Measured on GPT-2 small (demo2, scenario H9): the host aligns tenant A’s body to the body of an account it registered itself, from one weight matrix, and decrypts A’s traffic with its own key—100% of tokens.

Proposition 3 is the special case of a public base; the theorem says that even a *private* base does not help once one tenant key is known to the host. Confidentiality against the executing party therefore requires that it never compute on the real numbers—on encrypted numbers, on secret-shared numbers, or inside attested hardware. The first of these at plaintext speed is Solution 2.

VII. SOLUTION 2: TEMPLE-MASK-TRUST-NOBODY — FULL ISOLATION, INCLUDING FROM THE LLM HOST

Solution 1 leaves one party able to read: the host that executes the body. Solution 2 never gives it a readable number, at plaintext speed, by splitting the model by kind of arithmetic (Figure 1): the **main host** (GPU) does every linear map on one-time-padded activations; the **tenant** does the embedding, the nonlinear steps, the KV cache and the unembedding; and a **mask supply** provides, ahead of time, the pairs (r, Wr) the tenant needs—by default a cheap second server holding the same public weights at a different provider, **TEMPLE-DEALER-TRUST-NO-TWO**. Neither server ever receives a token id, a logit, a key or a readable activation.

Tenant (your machine)	Main host (GPU, online)	Dealer (cheap CPU, offline, other provider)
tokens, embeddings, KV cache, every nonlinear step, the answer	holds the weights in fixed point; computes $W(x + r) + b$ at full speed	holds the same weights; draws random r , computes Wr in bulk, ahead of time
sends $x + r \rightarrow$	sees: uniformly random 64-bit integers only	sees: “send me n masks” only

Tenant (your machine)	Main host (GPU, online)	Dealer (cheap CPU, offline, other provider)
$\leftarrow$ receives $W(x + r) + b$, subtracts Wr	keeps no tenant state	(seed, Wr) packs $\rightarrow$ tenant, in advance

The flow of TEMPLE-MASK-TRUST-NOBODY. Per token and layer the tenant sends every activation as $x + r$ (a one-time pad on $\mathbb{Z}_{2^{64}}$), the host returns $W(x + r) + b$, the tenant subtracts the precomputed Wr and continues.

7.1 The protocol

Online. Activations and weights are fixed-point integers on the ring $\mathbb{Z}_{2^{64}}$ (2^{18} fractional bits; 64-bit arithmetic wraps, which is exactly the ring). For each of the four linear groups of a layer ($q|k|v$, o , gate|up, down) the tenant draws a fresh mask r , uniform on the ring, and sends $x + r$ in the clear: to the host it is a uniformly random vector—a one-time pad, perfect secrecy per element, no key, no encryption, no assumption about the host’s power. The host computes $W(x + r) + b$ with its fixed-point weights and returns it; the tenant subtracts the precomputed Wr (the wrap-around cancels), obtains $Wx + b$ exactly, and does RMSNorm, RoPE, attention, SiLU and the KV cache itself. Ninety-six such calls per token; nothing in the model changes. *Offline.* The dealer draws each mask from a 32-byte seed, computes (r, Wr) for many future tokens in bulk (a matrix–matrix product, cheap on a CPU) and ships them as packs (≈ 3.9 MB per token for Qwen2.5-0.5B); the tenant expands r from the seed with SHAKE-256 and consumes each mask once. The only assumption is that host and dealer do not collude, so they are run by different providers. This is Beaver-style preprocessing (Beaver 1991; Mishra et al. 2020) applied to a language model.

7.1.1 Adversary model, and what the host does learn.

Confidentiality of every value needs no honesty from the host: each call carries a fresh uniform $x + r$, so a host that deviates arbitrarily from the protocol, logs everything and computes without limit still receives only uniform integers, and the dealer receives no activation at all. Correctness of the answer, by contrast, assumes an honest host and dealer: a cheating host can corrupt the answer but cannot read it. What the host does learn is metadata—the tenant’s bearer identity, the number of prompt tokens (the prefill batch), the number of tokens generated and the timing of the calls. Padding prompts and answers to fixed lengths would hide the counts at proportional cost; this paper does not do so.

7.2 Production-shape implementation and verification: two LLM servers, eighteen checks

temple-mask-trust-nobody/ holds three self-contained programs: main_host.py (stateless; bearer-token authentication; one binary request per layer group; exact $\bmod 2^{64}$ products in float64 limbs for GPUs without 64-bit integer GEMM, verified bit for bit in a self test), dealer.py (a background producer keeps packs ready and hands each out once) and tenant.py. demo_two_llms.py spawns both servers with fresh random tokens, or targets real deployments, and verifies the three claims of this section—prompts work, the host cannot eavesdrop, tenants are isolated—with the checks of Table 6.

Solution 2 verified: demo_two_llms.py, Qwen2.5-0.5B-Instruct, all 24 layers masked, main host, dealer and two tenants on one two-core CPU machine. All 18 checks pass.

Checks	Result
S0, A1–A2	int64, two-limb float64, chunked float32 and float64 kernels agree bit for bit; a wrong bearer token is rejected (401) by both servers
C1–C4 prompts work	alice: “The capital city of Canada is Ottawa.”; bob (a clinical prompt): “The most likely diagnosis in this case is angina pectoris, ...”—token for token the plain model’s greedy answers; logits agree at every one of 9 and 17 decoding steps to 3.3×10^{-3} and 1.9×10^{-3} (fixed-point rounding; smallest top-2 margin on the paths 1.99 and 0.14)
P1	decode 1.8 s per token on two CPU cores, host compute 0.5 s of it, 96 host calls per token; $1,678 \times$ faster than online homomorphic encryption on the same machine
E1–E5 the host cannot eavesdrop	the strongest curious-host attack (nearest-embedding token recovery from the layer-0 input) recovers 37/37 tokens from plaintext and 0/37 from what the host receives; the 8.2 M int64 values sent have 1-bit fraction 0.5000, a byte histogram within 0.7% of flat and correlation -0.0005 with the true activations; what the host returns is masked too; the same prompt sent again shares 0 values with the first (bit Hamming distance 0.500), answer unchanged; neither server’s log holds a token id, prompt or answer text
I1–I5 tenants are isolated	alice and bob generate concurrently against the same servers, both exact; the host reports tenant_state = 0; 159 and 162 mask seeds received, none shared, none reused; bob subtracting his masks from alice’s traffic recovers 0/37; the dealer served 87 requests and saw only “ n masks”

7.2.1 Deployed across providers.

Run for real—the host on a rented GPU pod (RunPod, RTX 3090, created by runpod_host.py), the dealer and the tenant at another provider—the tenant decoded “The capital of Canada is Ottawa.”, every check passed, and the host’s log held 768 lines of “masked in, masked out” and no text. Decode was 3.6 s per token: 96 sequential calls at 37 ms of round trip, the GPU idle 98% of the time. The wire, not the arithmetic, is the cost; the next subsection removes it.

Easy Explanation (Solution 2) — try it yourself.

Everything is in `temple-mask-trust-nobody/` of the repository. **One machine, nothing to set up:** `bash run_demo.sh` starts the host and the dealer with fresh random tokens, runs two tenants and prints PASS/FAIL for every claim (answers identical to the unprotected model; the strongest eavesdropping attack recovers 100% of the tokens from plaintext and 0% from what the host sees; repeating a prompt produces an unrelated wire; the logs hold no text; tenants are isolated). **Two providers:** on a GPU pod run `main_host.py` (or the native `temple_host_native`); on a cheap CPU box at another provider run `dealer.py`; on your own machine run `tenant.py --host tcp://<gpu-host>:8010 --dealer http://<dealer>:8009` "What is the capital of Canada?" and watch the answer appear while the host's log shows only masked in, masked out (run the tenant on a small VM next to the host for the 25 ms-per-token figure). **No second provider:** mint the masks on the host with `temple-mint-trust-none/mint_local.py`, serve them with `mint_client.py` serve, and point the same tenant at that. The GPU commands, one per experiment, are in `REPRODUCE.md`.

7.3 The configurations, on one prompt

Table 7 gives the numbers of the configurations this paper recommends, all on "What is the capital of Canada?" with Qwen2.5-0.5B-Instruct (36 prompt tokens, an 8-token answer, identical in every row). Every blind row makes the same 96 sequential host calls per token, so the online time is set by the wire and the per-call overhead; the rows below are the ones that pushed both down.

The recommended configurations on the same prompt. *host and dealer do not collude; the last row removes even that.

Configuration	Who reads	Decode / token	Whole answer
Plain model	host	≈ 0.05 s (GPU)	< 1 s
TEMPLE-KEY-TRUST-HOST (Sec. 3)	the host	zero overhead	< 1 s
TEMPLE-MASK-TRUST-NOBODY, Python host and tenant, all on one two-core CPU (Sec. 7.2)	nobody*	1.8 s	≈ 15 s
same, GPU host at one provider, tenant across the internet	nobody*	3.6 s	≈ 29 s
same, tenant's process co-located with the host, Python	nobody*	0.14 s	2.7 s
same, co-located, native C++/CUDA host and tenant (Sec. 7.4)	nobody*	0.024–0.025 s	0.6 s
TEMPLE-MINT-TRUST-NONE: masks minted by the host, native host and tenant (Sec. 8.6)	nobody	0.023 s	0.6 s (+13.6 min of idle GPU per 8,192 tokens, once)

7.4 Below one second: the tenant's process next to the host, host and tenant compiled

The remedy that changes nothing in the protocol or the trust model is to place the *tenant's* process—the one that holds its keys, KV cache and masks—in the same datacenter as the host, and to have the tenant's device talk only to that process. Measured on the pod itself (loopback round trip 0.2 ms; the dealer is off the latency path, since packs are prefetched), decode with the Python host and tenant fell from 3.6 s to 0.32–0.46 s per token; four engineering steps, each verified exact—pre-stocking the dealer, a raw binary transport instead of HTTP, the tenant's tensors on the GPU, one weight matrix per linear group on the host—brought it to 0.14 s. The host was then rewritten in about 350 lines of C++ and CUDA (exact mod 2^{64} arithmetic in native 64-bit integers, one captured CUDA graph per (layer, group) for decode): 0.10–0.15 ms per call, 0.065 s per token. Finally the tenant was compiled too: `temple_tenant_native.cu` runs embedding, RMSNorm, RoPE, causal attention over a KV cache, SiLU, mask and unmask, unembedding and argmax, the SHAKE-256 seed expansion (verified byte for byte against Python's hashlib) and both wire protocols, with only tokenization left in a thin Python wrapper. Table 8 gives the result.

Three tenants against the native host, same pod (RTX 3090), co-located, raw TCP, CPU dealer pre-stocked with 512 tokens. All answers identical to the plain model, token for token.

	Python tenant, CPU	Python tenant, GPU	Native tenant (C++/CUDA)
Decode per token, "capital of Canada"	155–180 ms	65–75 ms	24–25 ms
Prefill, 36 tokens	1.8 s	0.9–1.0 s	0.43 s
Whole answer (36 + 8 tokens), generation only	3.2 s	1.5 s	0.6 s
64-token answer, per token	—	78–89 ms	27.7 ms
Clinical prompt (top-2 margin 0.136), per token	—	—	28.2 ms
Host per call ($q k v$, o , gate up, down)	0.10, 0.10, 0.15, 0.14 ms (native host)		

The per-token budget at 25 ms is now 96 host calls ≈ 5 –6 ms, the tenant's own kernels ≈ 3 –5 ms, mask upload and bookkeeping ≈ 6 ms, and the socket; the floor is 96 times the loopback round trip, about 20 ms. That floor, not the arithmetic, is the overhead: an optimized plain server decodes a 0.5B model in a few milliseconds per token, so the relative cost here is several-

fold; but the floor is fixed while the host’s own memory-bound GEMVs grow with the model (of the order of 15 ms per token for a 7B model on the same GPU), so at 7B the same protocol lands near 40 ms per token—25 tokens/s, still conversational—and the ratio to plaintext improves with model size. Prefill of long prompts (thousands of tokens; here 36) is batched in chunks of 256 rows per call but is not measured in this paper. The Python host and tenant remain the reference implementation (any machine, CPU or GPU, HTTP or TCP) and pass the same checks; the native pair is the production path. Two supply lessons from the runs: at 25 ms per token the tenant outruns a CPU dealer that mints while it decodes (pre-stock it, or give the dealer its own GPU), and a dealer minting on the *same* GPU as the host halves everyone’s speed—it belongs on another machine, which is where the trust model puts it anyway.

VIII.SOLUTION 2 WITHOUT THE SECOND SERVER: TEMPLE-MINT-TRUST-NONE

Solution 2 as built in Section 7 needs two LLM servers because someone other than the host must compute Wr . This section removes the second server. The result is a system with *one* LLM host, a stateless, weightless clerk (a serverless function such as a Cloudflare Worker) and a thin client, in which the host mints the tenant’s masks itself, blind, in bulk, off the latency path, under exact homomorphic encryption. We first state precisely why nothing simpler can work, then give the construction, then the measurement.

8.1 The mask wall

Proposition 6 (The mask wall). *Let a party S output fresh pairs (r, Wr) for uniformly random $r \in \mathbb{Z}_{2^{64}}^d$. Then (i) after d linearly independent pairs the outputs of S determine W , so S holds the information of W ; and (ii) if the host learns r for a pair used to pad x , it learns x .*

Both parts are immediate ($W = (Wr_1 \cdots Wr_d)(r_1 \cdots r_d)^{-1}$; $x = (x + r) - r$). Part (i) rules out a “weightless dealer”; part (ii) rules out letting the host compute Wr in the clear on vectors it cannot *link* to the tenant: a host that logs everything pairs every logged r_j with every observed $x + r$ and keeps the one whose difference is a plausible activation (the attack of check E1). Hence the mask source either holds W and is not the host, or is the host computing on masks it cannot read. The second is TEMPLE-MINT-TRUST-NONE.

8.2 The construction

Exact HE. Minting cannot use an approximate scheme: r is uniform on $\mathbb{Z}_{2^{64}}$, so a 10^{-6} relative error in Wr is 2^{44} times Wx . We use BFV (Brakerski 2012; Fan and Vercauteren 2012), as implemented in Microsoft SEAL (Microsoft Research 2023) through TenSEAL (Benaissa et al. 2021), whose plaintext space is $\mathbb{Z}_t$ for a prime t . *CRT keeps the online ring.* Wr as an integer is bounded ($|Wr| < 2^{96}$ for 19-bit weights, 64-bit r , width ≤ 4864), so the host computes $Wr \bmod t_1$ and $Wr \bmod t_2$ under two BFV contexts and the tenant recombines by the Chinese remainder theorem and reduces modulo 2^{64} : the minted masks are bit for bit what a dealer would produce, and the host’s online kernel, the client and the pack format are unchanged. *Token batching.* A BFV ciphertext at degree $N = 8192$ carries N slots; we put *tokens* in the slots, so ciphertext C_k holds coordinate k of the masks of 8192 future tokens and

$$\text{Enc}(Wr)_i = \sum_k W_{ik} C_k$$

is a chain of scalar-times-ciphertext products and additions—no rotations, one multiplicative level—each advancing 8192 tokens at once. The cost is $2 \times (\text{linear parameters})/N$ scalar operations per token, 87,000 for Qwen2.5-0.5B, all 24 layers: masks are prompt-independent and can be minted for thousands of future tokens in one batch, which is what online homomorphic encryption can never have.

8.2.1.Three parties, none with a session.

Table 9 gives the roles. The clerk is a stateless, weightless web service—a Cloudflare Worker (worker.js) or a plain Python server (clerk_server.py); the same six checks pass against both. The tenant derives its id and every mask seed from its 32-byte secret locally, encrypts r under its own BFV key and uploads the ciphertexts with its *public* context; the clerk parks the $\text{Enc}(r)$ chunks, lists work for the host, parks the $\text{Enc}(Wr)$ results and serves them. It holds no key and no secret, never decrypts, and never sees r , Wr , a prompt or an activation; the host pulls from it. The host’s only state is its weights; the tenant’s is the secret, a counter and the conversation; the stock of Wr is a cache.

TEMPLE-MINT-TRUST-NONE: *one LLM host, a stateless clerk, a thin client.*

Party	Holds	Does	Sees
Host (GPU)	weights	online: $W(x + r) + b$ at plaintext speed; offline queue: $\text{Enc}(r) \mapsto \text{Enc}(Wr)$ under BFV in batches of 8,192 tokens, when idle	uniform noise online; ciphertexts offline; never r
Clerk (serverless)	nothing at rest; no key, no secret	park $\text{Enc}(r)$ and the public context, drive the host queue, store $\text{Enc}(Wr)$, serve packs	ciphertexts and public contexts only
Client	32-byte secret, transcript	derive seeds, encrypt r , tokenize, $x + r$, the nonlinear steps, KV cache, argmax	everything (it is the tenant)

8.3 Measured

demo_mint_kernel.py runs the construction on a real linear map of Qwen2.5-0.5B-Instruct (the 128×896 key projection of layer 0), minting masks for 8,192 tokens on the two-core CPU machine; the host operates on a *public* context only (an attempt to decrypt on its side raises “the context doesn’t hold a secret key”), and the minted masks are then used, without any change to the host’s kernel, on a real prompt’s activations (Table 10). Against the deployed Cloudflare clerk, test_worker.py passes six checks: seeds derived deterministically from the secret and distinct across secrets; a real batch travels tenant → clerk → host → clerk → tenant with the host on the public context and 1,048,576 minted values exact; the minted masks drive the unchanged online kernel; the batch is deleted on request; 124 ms per call.

TEMPLE-MINT-TRUST-NONE measured: one map (prototype, two-core CPU) and the whole model (RTX 3090).

Quantity	Value
Minted Wr vs exact integer product mod 2^{64} , 8192 $\times$ 128 values	identical, bit for bit ; host held no secret key
Online path with minted masks, 47-token prompt, unchanged host kernel	host saw $x + r$ (1-bit fraction 0.5001); tenant recovered $Wx + b$ to 2.3×10^{-5}
Operations per token, whole model (2 CRT primes $\times$ 358M params / 8192)	87,000 scalar multiply-adds
Library loop from Python (527 μ s/op), two-core CPU	46 s of offline time per token
Native GEMM kernel on raw ciphertexts (Sec. 8.4), two-core CPU	10–13 μ s/op: 0.94 s per token
Whole model on an RTX 3090, 8,192 tokens (Sec. 8.6)	13.6 min per batch = 99 ms of offline time per token (GPU 42 ms), all 96 units exact
Traffic per token (three-prime modulus, 248 KB ciphertexts)	11 MB Enc(r) up, 19 MB Enc(Wr) down (plaintext packs: 2.4 MB)

8.4 The native kernel: minting as a matrix product on raw ciphertexts

Calling the HE library 87,000 times per token from Python costs 527 μ s per call, almost all overhead. A BFV ciphertext in SEAL is stored in coefficient form as residues over the primes q_j of the coefficient modulus, and multiplying it by a constant w is coefficient-wise multiplication by $w \bmod q_j$; a whole linear map, $\text{Acc}_i = \sum_k W_{ik} D_k \pmod{q_j}$, is therefore one integer matrix product per prime. mintkernel.py parses the serialized ciphertexts, splits the coefficients into 22-bit limbs so that every float64 GEMM over a 2^{11} -wide slice is exact, reduces modulo q_j , recombines and writes the result back into the container; the output polynomials are bit-identical to the library’s (self test), and the same torch program runs on a GPU. With three 56-bit coefficient primes instead of the library’s five (ciphertexts 433 to 248 KB; a chain 32 times longer than the model needs still decrypts exactly) the two-core CPU mints at 10–13 μ s per operation, 0.94 s per token.

8.5 Masks as a commodity

The two supplies sit on one dial: the dealer mints at $\approx 1 \times$ the plaintext cost under a non-collusion assumption; the host mints under exact encryption at 99 ms of GPU time per token under none. What is general is that *the blinding material for a language model is independent of every prompt*: Wr depends only on the public weights and on randomness, so it can be minted anywhere, by anyone who holds the weights or by the host under encryption, stockpiled, moved through a CDN and consumed statelessly from a 32-byte secret.

8.6 End to end, with no dealer, on a GPU

mint_local.py mints masks for **8,192 tokens for all 24 layers** in one process on the GPU pod (RTX 3090): for every (layer, kind) the tenant side draws r from the secret and encrypts every coordinate for the two CRT primes in a process pool, the host side computes Enc(Wr) with the GEMM kernel on the GPU from the public context, the tenant side decrypts, recombines by CRT and checks the result bit for bit against the exact integer product; the packs are written in the dealer’s format and consumed by the tenants of Section 7.4 with no dealer process anywhere. The batch took **13.6 minutes—99 ms of offline time per token**, of which 42 ms is GPU compute, 18 ms encryption and 6 ms decryption; 362,496 ciphertexts in, 608,256 out; all 96 units exact. The prompt then ran on the minted packs: “*The capital of Canada is Ottawa.*” at **23.1 ms per token**, prefill 0.46 s. A batch of 8,192 tokens is a long conversation, minted once in idle time; on a data-centre GPU with $30 \times$ the fp64 throughput it is under a minute. What remains is engineering: a native integer CUDA minting kernel in place of the float64 limbs, seeded ciphertexts to halve the upload, a single-prime online ring to halve everything again, and a browser tenant with the nonlinear steps in WebAssembly, so that a URL is the whole installation.

IX.OVERHEAD, KEY MANAGEMENT, RE-KEYING AND SCALE

A keyed body has the same matrices at the same shapes as the base, so Solution 1 costs no FLOPs: tokens per second are identical to the base model within measurement noise. Memory is one rotated weight copy per tenant (498 MB → 652 MB for GPT-2 small, the difference being the untied unembedding), which is the cost that binds at frontier scale: a rotated copy of DeepSeek-V3 is 671 GB per tenant, so at millions of tenants a single key-equivariant body, not per-tenant copies, is the route (the construction goes through DeepSeek-V3/R1 (DeepSeek-AI 2024, 2025)—RMSNorm gains folded, every residual-stream

interface rotated including the latent-attention projections, experts and router, vocabulary tables permuted—with the FP8 checkpoint the one wrinkle). Rotating only some layers is not an option: the residual stream must stay rotated from embedding to unembedding, or canonical hidden states appear in the middle of the network; the way out at that scale is a single key-equivariant body trained over random keys with the key as input—a few thousand steps of continued pre-training on public data against the $\approx 2.8\text{M}$ GPU-hours of the base—which is the open problem of Section 11. The tenant stores a 32-byte master secret; (π, Q) is derived on demand. A full re-key is one multiplication per matrix (2.8 s for GPT-2 small); an incremental re-key from Q_1 to Q_2 is the rotation $Q_1^T Q_2$ applied by the host to its rotated copy without seeing either key. (π, Q) is a symmetric key; public-key cryptography is useful *around* it: the secret sealed to a device’s X25519 key, attestation of the rotated checkpoint the host loaded, and a capability split between an input map π_{in} and an output map π_{out} so that a writer can submit documents in the tenant’s language without reading the model’s answers.

9.0 Deployment.

casestudy/ is Solution 1 as a service: the owner folds and publishes the base once; a tenant draws a 32-byte secret, rotates and permutes a private copy locally, uploads it and chats; the host (RunPod-ready, bearer-token authentication) holds no keys and logs cipher ids only—alice’s prompt leaves her machine as “Ano tribute sil...”, the reply returns as “Enh252Thus voltatson...” and decrypts on her side to the base model’s completion, and bob, with his own key, obtains nothing of alice’s.

9.1 Deployment: who downloads what

The model is public and open-weight, so downloading it reveals nothing; the secret is the tenant’s prompt, never the weights. Each party needs only the part of the model its role uses (Table 11). Under Solution 1 the tenant holds a 32-byte secret and a thin client; the host downloads the base once, as any serving stack does. Under Solution 2 the split is by function: the host holds the fixed-point weights of the 96 linear maps; the dealer the same public weights, to mint (r, Wr) offline; the clerk none; and the tenant the model’s *non-linear* pieces—token embeddings, RMSNorm gains, RoPE tables and the unembedding—because someone the tenant trusts must turn tokens into vectors and back, run the non-linear steps and keep the KV cache, and the only such party is the tenant. For Qwen2.5-0.5B-Instruct (tied embedding and unembedding) that is one $151,936 \times 896$ table, 272 MB in bf16: a one-time download of public data that does not grow with the conversation; no tenant ever needs the full model. In production the tenant ships with, or fetches on first run, the exported tables (`native/export_tenant_tables.py`; `tenant.py`’s `from_pretrained` of the whole checkpoint is a prototype convenience), and runs on the user’s laptop or, for the 25 ms configuration of Section 7.4, on a small VM the user controls in the host’s datacenter: the trust boundary is the process, not the building. *The tenant downloads, but it does not compute much.* The 96 linear maps are essentially all of the model’s parameters and FLOPs, and they stay on the host; the tenant’s work per token—one embedding lookup, $24 \times$ (RMSNorm, RoPE, attention over its own KV cache, SiLU) and one unembedding—is about 0.3 GFLOP, a laptop’s work, and every CPU-tenant measurement in this paper is that: 155–180 ms per token next to the native host, 1.8 s per token with host, dealer and two tenants all on one two-core machine (Section 7.2). The GPU in the 25 ms rows is the host’s; a tenant-side GPU (65 ms) and the compiled tenant (25 ms) are optional. No high-power GPU is needed where the data is in the clear.

Who downloads what. Sizes for Qwen2.5-0.5B-Instruct.

Party	Solution 1 (KEY)	Solution 2 (MASK / MINT)	Size
Host (provider, GPU)	the base model, once; a rotated copy per tenant	fixed-point weights of the 96 linear maps (<code>main_host.py</code> ; <code>native/export_weights.py</code> once)	1 GB / 1.4 GB
Dealer (second provider, CPU)	–	the same public weights; mints (r, Wr) offline, never sees an activation	1 GB
Clerk (MINT only)	–	nothing: no weights, keys or sessions	0
Tenant (the user)	a 32-byte secret; thin client	embeddings, RMSNorm gains, RoPE tables, unembedding (<code>tenant.py</code> ; <code>native/export_tenant_tables.py</code> once)	272 MB bf16
Tenant’s compute per token	none (the host runs the model)	embedding, $24 \times$ (RMSNorm, RoPE, attention, SiLU), unembedding: ≈ 0.3 GFLOP; CPU 155–180 ms, GPU 65 ms, compiled 25 ms	no GPU needed

X.RELATIONSHIP TO EXISTING WORK

QuaRot and SpinQuant (Ashkboos et al. 2024; Liu et al. 2024) rotate the residual stream to remove outliers before quantization; the rotation is public and shared by all users, and its exactness is what we use as a per-tenant *secret*. Embedding inversion (Morris et al. 2023) shows that text embeddings invert to text given the embedding model, which is attack A1 in our setting and the reason the host must not hold the canonical weights in Solution 1. CipherChat (Yuan et al. 2024) shows that large models can *speak* simple ciphers when instructed; we relabel the machine instead of teaching it, so nothing is learned. Unsupervised decipherment (Knight et al. 2006) is the formal basis of attack A5 and of the Unlearnability requirement. Split learning (Vepakomma et al. 2018) cuts a network between client and server; our split is by kind of arithmetic, linear on the host, nonlinear on the tenant, with the host blinded by a one-time pad. Homomorphic and MPC inference (Hao et al. 2022; Pang et al. 2024; Lu et al. 2025; Chen et al. 2022; Li et al. 2023) and enclaves (Costan and Devadas 2016) protect against the executor at a cost we measured (49 minutes per token for online CKKS on our machine) or with a hardware trust we do not need; Beaver-style preprocessing (Mishra et al. 2020) is the ancestor of the dealer.

XI. OPTIONS CONSIDERED

Everything below was built or analysed on the way to the two solutions and set aside; one paragraph each, what it was and why not.

11.1 TEMPLE-CIPHER: the host computes on homomorphic ciphertext online

The same split as Solution 2 with the host operating on CKKS-encrypted activations. A one-layer prototype on Qwen2.5-0.5B was exact to 1.6×10^{-6} with the host holding only a public context, and the full generation loop ran end to end with the base model's answer. Why not: 123 s per layer on two CPU cores—49 minutes per token. Its mathematics survives in TEMPLE-MINT-TRUST-NONE, offline and batched, at 99 ms per token.

11.2 A noise-robust body

Train the shared body to tolerate Gaussian noise on its inputs so that tenants can randomize each occurrence of a token in embedding space. Two hundred steps of continued pre-training taught GPT-2's body to tolerate noise of three to six times the embedding norm (perplexity $1557 \rightarrow 138$ and $2544 \rightarrow 557$) and the nearest-neighbour attacker fell to chance. Why not: at the noise level that defeats clustering the utility gap to the control was still $15 \times$, and Proposition 4 says no training closes it—vector noise protects the stream from passive observers, never from the executor.

11.3 Rotation keys as confidentiality against the host

The hope that Solution 1 already hides data from the host. Why not: Theorem 2 and Proposition 3—with a public base from any single weight matrix, with a private base from one tenant key, the executor recovers $\mathbf{Q}$ to 10^{-7} and reads every tenant. Solution 1 is what its name says.

11.4 The tenant far from the host

The two-server design with the tenant across the internet works and passed every eavesdropping and isolation check on a real deployment. Why not as the recommended configuration: 96 sequential host calls per token at 37 ms of round trip is 3.6–4.7 s per token with the GPU idle 98% of the time; the tenant's own process in the host's datacenter brings the same protocol to 25 ms.

11.5 The Python host and tenant as the fast path

They pass every check and reached 0.14 s per token; at that point Python framing and synchronization over 96 calls were the whole budget, which the native pair removes. They remain the code to read and change.

11.6 The tenant's own machine as the dealer

Trust nobody: mint one's own masks from the public weights while idle. Why not as a headline: Beaver-style preprocessing is linear in the model size per token (2.4–3.9 MB for 0.5B, ≈ 12 MB for 7B, ≈ 54 MB for 70B), so storage binds, and for any model the laptop can run interactively it would simply run the model. A legitimate supply for the 7–14B class with an external disk; no new code.

11.7 Weightless or anonymized minting

A serverless function with no weights cannot mint masks (Proposition 6), and letting the host mint in the clear on vectors routed through a proxy fails against a host that logs everything. Why not: impossible against the adversary assumed. The serverless function is instead the clerk, which needs no weights.

11.8 Two-party MPC across two full hosts

Two powerful online hosts and a communication round per linear map, for the same trust assumption as the dealer, whose second party is small, offline and touches no data.

11.9 Trusted execution environments

Trade cryptography or non-collusion for a vendor's attestation and side-channel record; orthogonal to and composable with both solutions; not measured.

11.10 The dealer on the host's GPU

Halved everyone's speed (44–47 ms per token); a CPU dealer minting while the tenant decodes falls behind at 25 ms. The dealer belongs on another machine, pre-stocked—where the trust model puts it anyway.

11.11 Two kernel details set aside

Fusing the limbs of the exact GPU kernel into one GEMM halved the launches and doubled the per-call time (a memory-bound GEMV became a padded fp64 GEMM tile at $1/64$ rate); the native host removed the limbs altogether. The HE library's default five-prime coefficient modulus is for deep circuits; minting is one level, and three 56-bit primes halve the ciphertexts with a noise budget 32 times what is needed.

11.12 Closed-weight models

ChatGPT and Gemini cannot be keyed or split: both solutions need the weights. A boundary, not a choice.

11.13 The research roadmap that was

Key-equivariant bodies serving millions of tenants without per-tenant copies, MPC-native bodies, stronger decipherment attackers: none is needed for the two solutions as deployed; the first is the open problem that matters for scale.

All code exists at <https://github.com/syncaissa/TempleEnclaveAI>: the reference implementation of both solutions, the native C++/CUDA host and tenant, the clerk, the attacker suite, every experiment behind every table, and the verbatim transcripts they produced (results/). REPRODUCE.md maps each table of this paper to the command that reproduces it, the transcript it produced and the machine it ran on.

XII. CONCLUSION

Temple-Key-Trust-Host: each tenant of a shared LLM holds a secret key (π, Q) that renames the vocabulary and rotates the residual stream; exact by the equivariance theorem, token identities never leaving the tenant; tenant isolation is exact against an eight-attack suite, while a host that executes the body can align it and read—the theorem that makes the second solution necessary. **TEMPLE-MASK-TRUST-NOBODY:** the host evaluates every linear map on one-time-padded activations and the tenant the nonlinear steps, so the host is information-theoretically blind with the model unchanged; verified by eighteen checks, deployed across providers, and answering at 25 ms per token with a native C++/CUDA host and tenant, token for token the base model’s answer. Its masks come from a second small server (TEMPLE-DEALER-TRUST-NO-TWO) or from the host itself minting under exact encryption—8,192 tokens in 13.6 minutes of idle GPU (TEMPLE-MINT-TRUST-NONE)—so that the whole system can be one host, a stateless clerk and a 32-byte secret. The design goal—one body, many tenants, 0% sharing with other tenants and with the host—is met.

REFERENCES

1. Ashkboos, Saleh, Amirkeivan Mohtashami, Maximilian L. Croci, et al. 2024. “QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs.” *Advances in Neural Information Processing Systems (NeurIPS)*.
2. Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. “Layer Normalization.” *arXiv Preprint arXiv:1607.06450*.
3. Beaver, Donald. 1991. “Efficient Multiparty Protocols Using Circuit Randomization.” *Advances in Cryptology – CRYPTO ’91*, 420–32.
4. Benaissa, Ayoub, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. 2021. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. *arXiv preprint arXiv:2104.03152*.
5. Brakerski, Zvika. 2012. “Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP.” *Advances in Cryptology – CRYPTO 2012*, 868–86.
6. Chen, Tianyu, Hangbo Bao, Shaohan Huang, et al. 2022. “THE-X: Privacy-Preserving Transformer Inference with Homomorphic Encryption.” *Findings of ACL*.
7. Costan, Victor, and Srinivas Devadas. 2016. “Intel SGX Explained.” *IACR Cryptology ePrint Archive 2016* (86).
8. DeepSeek-AI. 2024. “DeepSeek-V3 Technical Report.” *arXiv Preprint arXiv:2412.19437*.
9. DeepSeek-AI. 2025. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.” *arXiv Preprint arXiv:2501.12948*.
10. Fan, Junfeng, and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive, Report 2012/144*.
11. Feyisetan, Oluwaseyi, Borja Balle, Thomas Drake, and Tom Diethe. 2020. “Privacy- and Utility-Preserving Textual Analysis via Calibrated Multivariate Perturbations.” *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*.
12. Hao, Meng, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. “Iron: Private Inference on Transformers.” *Advances in Neural Information Processing Systems (NeurIPS)*.
13. Knight, Kevin, Anish Nair, Nishit Rathod, and Kenji Yamada. 2006. “Unsupervised Analysis for Decipherment Problems.” *Proceedings of COLING/ACL*.
14. Li, Dacheng, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. 2023. “MPCFormer: Fast, Performant and Private Transformer Inference with MPC.” *International Conference on Learning Representations (ICLR)*.
15. Liu, Zechun, Changsheng Zhao, Igor Fedorov, et al. 2024. “SpinQuant: LLM Quantization with Learned Rotations.” *arXiv Preprint arXiv:2405.16406*.
16. Lu, Wen-jie, Zhicong Huang, Zhen Gu, et al. 2025. “BumbleBee: Secure Two-Party Inference Framework for Large Transformers.” *Network and Distributed System Security Symposium (NDSS)*.
17. Microsoft Research. 2023. Microsoft SEAL (Release 4.1). <https://github.com/Microsoft/SEAL>.
18. Mishra, Pratyush, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. “Delphi: A Cryptographic Inference Service for Neural Networks.” *USENIX Security Symposium*.
19. Morris, John X., Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M. Rush. 2023. “Text Embeddings Reveal (Almost) as Much as Text.” *Proceedings of EMNLP*.
20. Pang, Qi, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. “BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers.” *IEEE Symposium on Security and Privacy (s&p)*.
21. Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. “Language Models Are Unsupervised Multitask Learners.” *OpenAI Technical Report*.
22. Schönemann, Peter H. 1966. “A Generalized Solution of the Orthogonal Procrustes Problem.” *Psychometrika* 31 (1): 1–10.
23. Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. “Attention Is All You Need.” *Advances in Neural Information Processing Systems (NeurIPS)*.
24. Vepakomma, Praneeth, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. “Split Learning for Health: Distributed Deep Learning Without Sharing Raw Patient Data.” *arXiv Preprint arXiv:1812.00564*.
25. Yuan, Youliang, Wenxiang Jiao, Wenxuan Wang, et al. 2024. “GPT-4 Is Too Smart to Be Safe: Stealthy Chat with LLMs via Cipher.” *International Conference on Learning Representations (ICLR)*.
26. Zhang, Biao, and Rico Sennrich. 2019. “Root Mean Square Layer Normalization.” *Advances in Neural Information Processing Systems (NeurIPS)*.
27. Zhou, Xin, Yi Lu, Ruotian Ma, et al. 2023. “TextObfuscator: Making Pre-Trained Language Model a Privacy Protector via Obfuscating Word Representations.” *Findings of ACL*.