
Automating Privacy-Preserving Model Deployment

Ivan Lin

Jacky Li

With
Apart Research

Abstract

The deployment of large language models (LLMs) in high-stakes defensive domains (e.g. biosecurity, critical infrastructure, etc) is constrained by the risk of prompt and model-weight exposure on untrusted hardware. Homomorphic Encryption (HE) offers a cryptographic shield by enabling computation on encrypted data, but standard transformer architectures rely on non-polynomial operations (e.g., softmax, GELU, LayerNorm) that are poorly matched to HE’s addition, multiplication arithmetic. In this work, we present an automated detection transformation pipeline that converts off-the-shelf pretrained transformers into HE-ready polynomial circuits without retraining. We apply this end-to-end transformation to GPT-2 and Qwen2.5-Coder-3B-Instruct and evaluate both baseline and HE-ready models on HellaSwag and LAMBADA. Qwen2.5 retains essentially identical performance after full polynomialization, while GPT-2 degrades sharply, suggesting that modern over-parameterized models exhibit substantially greater approximation resilience than older, smaller architectures. Our prototype integrates with a TenSEAL-style HE backend conceptually, focusing on the architectural side rather than full latency optimization. Overall, this pipeline lowers the barrier to deploying privacy-preserving, encrypted LLM inference in zero-trust environments, complementing ongoing advances in GPU-accelerated HE and secure model-serving infrastructure.

<https://github.com/ivanlin9/Def-acc.git>

1. Introduction

As Large Language Models (LLMs) become integral to critical decision-making processes, the privacy of input queries and context windows has become paramount. Homomorphic Encryption (HE) enables inference directly on encrypted data so that a server can run the model without ever seeing the plaintext, modern deep networks are built from operations (e.g. softmax, LayerNorms, etc) that rely on exp, division and square roots. All of which are poorly matched to the addition and multiplication world of HE.

At the same time, the underlying threat model is getting sharper as models get more capable. In high-stakes deployments (healthcare, cyber defense, biosecurity, finance), prompts and context windows routinely contain sensitive or security-relevant information, and models are often served on infrastructure that cannot be fully trusted. If weights sit in plaintext on a cloud GPU, a compromised server or malicious insider can exfiltrate both user data and the model itself. HE offers a clean cryptographic story for this setting: keep user inputs encrypted end-to-end, optionally keep model weights encrypted on untrusted hardware, and perform the entire forward pass without exposing either to the server. The catch is that practical HE is still very slow. A large fraction of the overhead (often 50–90%), comes from bootstrapping (Leo de Castro. Et al), and even with recent GPU-accelerated HE libraries this remains orders of magnitude slower than standard inference. This is promising but not yet something you would casually drop into every production LLM service.

Our contribution in 48 hours is to attack a different bottleneck: not the raw cryptographic speed, but the friction of making real pretrained models HE-compatible in the first place. Rather than designing new architectures and retraining from scratch, we introduce a two-stage, model-agnostic pipeline:

Scanner. The Scanner is a static analyzer over the model’s computation graph. Given a pretrained transformer, it walks the PyTorch module hierarchy and automatically detects HE-unfriendly components: softmax layers, normalization layers (LayerNorms and related variants), non-polynomial activations (e.g. GELU, SiLU, tanh), and any modules that internally rely on exponentials, division, square roots, or max operations. It produces a structured “HE pain report” describing where these components appear and how frequently. Conceptually, the Scanner encodes our threat model and hardware constraints into a concrete diagnostic: it highlights exactly which parts of a given LLM would either break HE correctness or explode HE runtime and noise growth if left untouched.

Wrapper. The Wrapper takes the Scanner’s report and monkeypatches on the model to make it HE-friendly while preserving as much of the original computation as possible. Concretely, it systematically replaces the flagged components with low-degree polynomial surrogates and HE-compatible alternatives, while leaving the underlying pretrained weights intact wherever the operation semantics allow. The result is an HE-ready variant of the original model whose forward pass consists only of additions and multiplications. This transformed model can then be plugged into any approximate HE backend; in our reference implementation we use a TenSEAL-style scheme to encrypt parameters and activations and run encrypted

inference on untrusted infrastructure. By automating this pipeline and demonstrating that it introduces minimal degradation on standard benchmarks for realistic models, we aim to make it far easier for defenders to experiment with and eventually adopt encrypted-input, encrypted-weight deployments as HE runtimes improve, strengthening both privacy and resistance to weight exfiltration from untrusted servers.

2. Methods

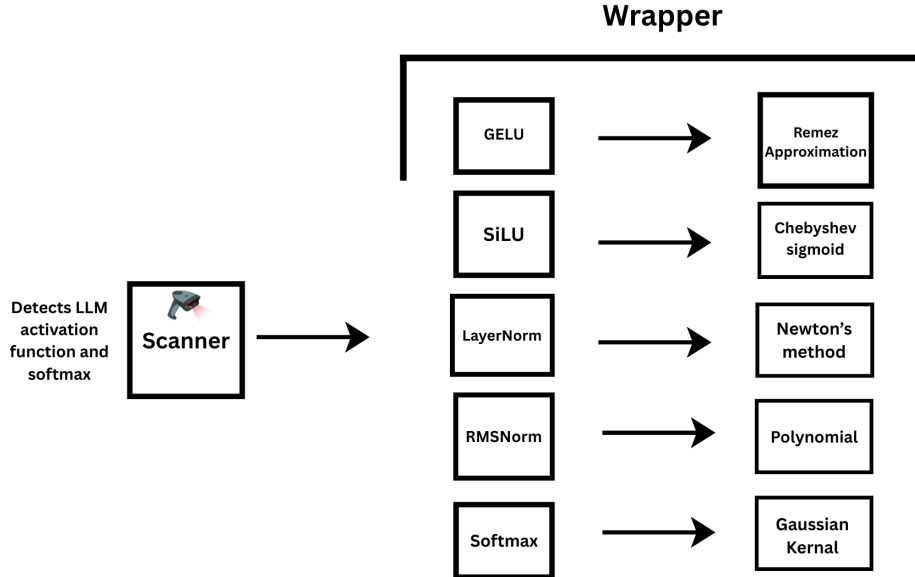


Figure 1: The Scanner traces the model architecture including functional calls via monkeypatching to identify specific layers that break FHE constraints. The Wrapper then transmutes these layers into polynomial equivalents using approximation theory. Our approach provides configurable precision levels, from low degree polynomial fallbacks for shallow encryption depth to high-degree Remez algorithms for maximum semantic preservation. This tool effectively democratizes Private AI by making standard open-source models compatible with encrypted inference engines.

Our pipeline operates in two distinct phases: detection (Scanning) and transformation (Wrapping). The system targets three primary architectural components that are hostile to homomorphic encryption: activation functions, normalization layers, and the attention mechanism. All code is implemented in PyTorch and built to work directly with Hugging Face transformer models.

2.1 The Scanner & Wrapper Architecture

The Scanner uses PyTorch’s module hooks and lightweight functional tracing to map out the model’s computation graph. It detects both Class-based modules and Functional calls.

This produces a structured report that tells us, for any given pretrained model, exactly where HE-unfriendly operations live and how frequently they appear.

The Wrapper consumes this report and performs the actual graph surgery. It walks the module hierarchy and replaces flagged components with custom subclasses whose forward passes use only addition and multiplication. These replacements are configured through presets such as:

- `he_friendly_low`: lower-degree polynomials, shallower multiplicative depth, better for HE but slightly looser approximation;
- `he_friendly_high`: higher-degree polynomials, slightly more depth, but closer to the original floating-point behaviour.

The result is a drop-in HE-ready model with the same architecture and pretrained weights, but a modified computation graph that is compatible with approximate HE backends.

2.2 Polynomial Approximation of Activations

Full derivations are located in the appendix under Approximations!!!

Standard activations (GELU, SiLU) depend on $\tanh$ or sigmoid, which in turn require exponentials and divisions, both expensive and noisy under HE. Our design goal is to keep the “shape” of these activations while reducing them to low-degree polynomials.

We work over a clipped domain (e.g. $x \in [-4, 4]$), which matches the typical range of activations in trained transformers and keeps polynomial errors bounded.

- **GELU Replacement:** The original GELU uses a $\tanh$ -based approximation

$$\text{GELU}(x) = 0.5x \left(1 + \tanh \left(\sqrt{2/\pi}(x + 0.044715x^3) \right) \right)$$

In our HE ready variant, we approximate it using a pure polynomial.

$$\text{GELU}_{\text{poly}}(\tilde{x}) = 0.5 \tilde{x} \left(1 + a_1 \tilde{x} + a_3 \tilde{x}^3 \right),$$

The coefficients come from offline minimax / Remez-style fitting on the clipped domain, so at runtime we only evaluate a small polynomial.

- **SiLU Replacement:** The original SiLU uses a tanh-based approximation

$$\text{SiLU}(x) = x \sigma(x), \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

We approximate this to

$$\text{SiLU}_{\text{poly}}(x) = \tilde{x} \sigma_{\text{poly}}(\tilde{x}) = \tilde{x} (s_0 + s_1 \tilde{x} + s_3 \tilde{x}^3).$$

2.3 Division-Free Normalization

Standard LayerNorm and RMSNorm rely on reciprocal square roots ($\frac{1}{\sqrt{x}}$), which are expensive in FHE.

- **LayerNorm:** Standard LayerNorm over a hidden vector $x \in \mathbb{R}^d$ computes

$$\text{LN}(x) = \frac{x - \mu}{\sqrt{v + \epsilon}} \odot \gamma + \beta,$$

We approximate the inverse square root using Newton's method, achieving the following as our final HE friendly LayerNorm.

$$\text{LN}_{\text{HE}}(x) = (x - \mu) \widehat{(v + \epsilon)^{-1/2}} \odot \gamma + \beta.$$

- **RMSNorm:** For RMSNorm, we normalize by the root mean square:

$$\text{rms} = \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \epsilon}.$$

Thus, our HE-friendly RMSNorm is

$$\text{RMSNorm}_{\text{HE}}(x) = x \cdot \left(1 - \frac{1}{2}u + \frac{3}{8}u^2\right) \odot \gamma,$$

where γ is the learned scaling parameter.

2.4 HE-Friendly Attention Mechanisms

The Softmax function is the most computationally expensive operation in Transformers due to the exponentiation and the division by the sum. We provide two replacement strategies:

1. Chebyshev Softmax (Semantically Preserving):
2. Gaussian Kernel Attention (HE-Optimized):

In the current hackathon codebase, Gaussian kernel attention is the default implemented path, while Chebyshev Softmax is more of a performance-optimized variant that slots cleanly into the same Wrapper framework.

To validate the efficacy of the Scanner & Wrapper pipeline, we conducted comparative inference tests between the original PyTorch models (“Baseline”) and their polynomial-approximated counterparts (“HE-ready”). We evaluated two distinct architectures: GPT-2 and Qwen2.5-Coder-3B-Instruct. Performance was measured on HellaSwag and LAMBADA (OpenAI).

3. Results

Model	Variant	Hella Swag	LAMBDA
Qwen2.5 Coder 3B-Instruct	Baseline	0.514	0.678
Qwen2.5 Coder 3B-Instruct	HE-ready	0.5133	0.680
GPT-2	Baseline	0.289	0.326
GPT-2	HE-ready	0.250	0.00

Qwen2.5-Coder-3B-Instruct:

- HellaSwag drops by 0.0003 ($0.5136 \rightarrow 0.5133$).
- LAMBADA slightly increases by 0.0016 ($0.6784 \rightarrow 0.6800$)

for a reasonably large, well-trained model, the residual stream is robust to small, smooth perturbations of the activation nonlinearity, so polynomial approximations over the typical activation range don’t hurt much.

GPT-2:

- HellaSwag drops by about 0.039 ($0.2892 \rightarrow 0.2504$).
- LAMBADA collapses to 0.0.

GPT-2 is small and heavily reliant on precise normalization. Approximations of LayerNorm destabilize the scale of the residual stream, so the model effectively fails on long-range close tasks. This is evidence that

“just polynomialize everything” is unsafe, and that norms need more careful design or partial plaintext fallback.

4. Discussion and Conclusion

The result differences are well within expected evaluation noise. Qualitatively, Qwen’s behaviour on few-shot prompts and coding-style tasks also appeared stable: completions remained coherent and on-topic, and we did not observe obvious degeneration. This suggests that, for a reasonably large and over-parameterized model, the residual stream is robust to small, smooth perturbations in the activation nonlinearity. Intuitively, Qwen’s internal feature representations appear to live in a regime where the polynomial error of our GELU/SiLU approximations does not push activations across important decision boundaries.

5. References

- Rho, D., Kim, T., Park, M., Kim, J. W., Chae, H., Ryu, E. K., & Cheon, J. H. (2025). Encryption-Friendly LLM Architecture. *International Conference on Learning Representations*.
<https://openreview.net/forum?id=Is15K3J8Wq>
- de Castro, L., Escudero, D., & Polychroniadou, A. (2023). *Privacy-preserving large language model inference via GPU-accelerated fully homomorphic encryption* (arXiv:2311.01893). [4] arXiv. <https://doi.org/10.48550/arXiv.2311.01893>
-] Y. Dong, W.-j. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W. Chen, “PUMA: Secure Inference of LLaMA-7B in Five Minutes,” arXiv preprint arXiv:2307.12533, 2023.
- [2] D. Rho, T. Kim, M. Park, J. W. Kim, H. Chae, J. H. Cheon, and E. K. Ryu, “Encryption-Friendly LLM Architecture,” arXiv preprint arXiv:2410.02486, 2024.
- [3] L. de Castro, A. Polychroniadou, and D. Escudero, “Privacy-Preserving Large Language Model

Inference via GPU-Accelerated Fully Homomorphic Encryption,” in Proc. NeurIPS 2024 Safe-

GenAI Workshop, 2024.

[4] N. Carlini, D. Paleka, K. D. Dvijotham, T. Steinke, J. Hayase, A. F. Cooper, K. Lee, M. Jagiel-

ski, M. Nasr, A. Conmy, I. Yona, E. Wallace, D. Rolnick, and F. Tram`er, “Stealing Part of a

Production Language Model,” arXiv preprint arXiv:2403.06634, 2024.

[5] V. Barbaza, A. R. D´ıaz-Rizo, H. Aboushady, S. Raptis, and H.-G. Stratigopoulos, “Stealing AI

Model Weights Through Covert Communication Channels,” arXiv preprint arXiv:2510.00151,

2025.

6. Appendix

Security Considerations:

This work focuses on closing the architectural gap between standard transformers and HE, not on fully optimizing the cryptographic runtime. Several limitations are important for interpreting our results and point directly to future work:

- **No end-to-end GPU-accelerated HE benchmarks.**
We designed the pipeline with modern GPU-accelerated CKKS implementations in mind, where bootstrapping accounts for roughly 50–90% of the runtime overhead. However, integrating a full GPU-accelerated bootstrapping backend into our demo pipeline proved out of scope for the hackathon timeframe. We ran initial encrypted inference tests, but due to a small logging/serialization bug in the timing code, we were unable to reliably record and report latency numbers. As a result, our current evaluation is architectural (accuracy under polynomial approximation) rather than systems-level (true encrypted throughput and latency).
- **Limited model and task coverage.**
We evaluated only two models (GPT-2 and Qwen2.5-Coder-3B-Instruct) on two benchmarks (HellaSwag, LAMBADA). This is sufficient to demonstrate that the pipeline *can* be nearly lossless in a realistic setting and can also fail badly in another, but not enough to claim generality. Future work would:

- Apply the Scanner & Wrapper to a wider range of architectures (e.g., larger Qwen/LLama variants, encoder-only models, instruction-tuned safety models)
- Evaluate on richer task suites, including reasoning-heavy and safety-relevant benchmarks.
- **End-to-end HE latency and trade-off curves.**
A natural next step is to pair our architectural pipeline with a mature GPU HE backend and report full inference curves: latency vs. multiplicative depth vs. approximation degree. This would let us map out where encrypted LLMs become practically usable (e.g. low-frequency, high-value queries in security-sensitive domains) and how much additional cryptographic acceleration is needed to reach broader deployment.

From an AI safety and Defensive Acceleration perspective, these limitations are not purely technical annoyances; they directly shape whether encrypted LLM deployment becomes a real tool for defenders or remains a theoretical ideal. Our results show that:

- For modern medium-scale models, the architectural barrier is surmountable: we can make them HE-ready with minimal accuracy loss in seconds.
- The remaining bottlenecks are systems-level (fast bootstrapping, GPU HE) and design-level (robust normalization), both of which are amenable to further research and engineering.

If those gaps are closed, the Scanner & Wrapper pipeline becomes a reusable bridge: it lets future HE advances immediately apply to a wide range of pretrained LLMs, enabling privacy-preserving and weight-exfiltration-resistant deployments in exactly the high-stakes settings (health, cyber, bio, finance) where we most need stronger cryptographic defenses.

Approximations: