# HEAT: FASTER FULLY HOMOMORPHIC INFERENCE VIA APPROXIMATIONS-WEIGHTS CO-ADAPTATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Fully homomorphic encryption (FHE) allows a server to run a language model directly on encrypted user prompts, but current approaches remain prohibitively slow. Ciphertexts natively support only addition, multiplication, and rotation, and multiplications may be composed only to a bounded depth before a costly bootstrapping operation is required to continue. Every nonlinearity must therefore be approximated by an iterative method; each iteration increasing the number of multiplications. A higher iteration count buys precision but exhausts the available depth more frequently and thus triggers more bootstraps, which dominate latency. We introduce **Homomorphic Encryption-Aware Training (HEAT)**, a *fine-tuning* method that makes the per-nonlinearity iteration counts *learnable*, enabling them and the model weights to co-adapt during training. HEAT optimizes iterations with respect to the task objective, allowing the model to adapt to approximation errors encountered during inference without architectural changes or retraining from scratch. We further relate iteration count to quantization bit width and bound, at fixed weights, the gap between our objective and quantization-aware training. On encrypted GPT-2 decoding, HEAT reduces iterations by 3.1×, bootstraps by 1.6×, and end-to-end latency by 1.4×, while improving decode agreement over the calibrated encrypted baseline.

## 1 INTRODUCTION

As language models grow in scale, running them locally becomes impractical for most users. Access therefore increasingly takes the form of *inference-as-a-service*, with models hosted on provider-controlled servers and queried remotely. This convenience comes at a privacy cost: prompts and outputs are processed in plaintext and are visible to the provider. Such exposure of users' data is difficult to reconcile with privacy-sensitive domains, where data protection is not only ethically preferable but legally mandated. Fully homomorphic encryption (FHE) protects outsourced language-model inference by enabling encrypted computation on ciphertexts. A client encrypts its prompt, the server evaluates the model without decrypting it, and only the client can recover the output. This is well suited to *inference-as-a-service*, because it preserves a non-interactive workflow (an encrypted request followed by an encrypted response) without requiring mutual trust (Hao et al., 2022; Li et al., 2022; Wu et al., 2024; Lu et al., 2025).

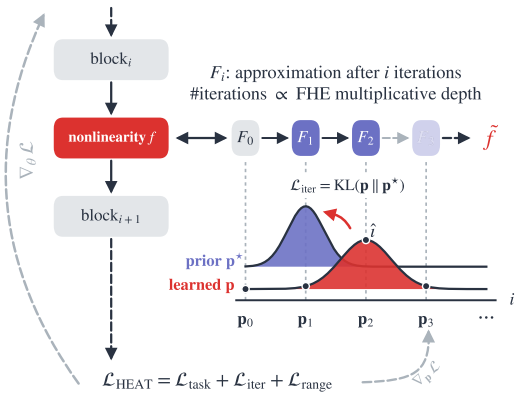


Figure 1: **HEAT overview.** Each nonlinearity $f$ is approximated iteratively $F_0, \ldots, F_n$. During fine-tuning, HEAT learns a distribution $\mathbf{p}=(p_0, \ldots, p_n)$ over iteration counts and evaluates $\tilde{f}(x)=\sum_{i=0}^{n} p_i F_i(x)$, regularized toward a prior $\mathbf{p}^\star$ favoring fewer iterations. At deployment, the count is fixed to $\hat{\imath}=\arg\max_i p_i$, and only $F_{\hat{\imath}}(x)$ is evaluated.

Running a model under FHE, however, is computationally expensive and suffers from extreme latency. Encrypted computation natively supports only additions, rotations, and multiplications oper-



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.

ations, and multiplications can be nested up to a finite maximum *multiplicative depth*, after which expensive *bootstrapping* (Gentry, 2009) is required to restore the multiplicative budget. Moreover, this limited choice of native operations forces nonlinearities to be *approximated by iterative algorithms* that recursively approximate such functions with compositions of addition and multiplication alone, with a precision that increases exponentially with the iteration count. This creates a trade-off between $(i)$ approximation precision and $(ii)$ inference latency. More iterations produce more accurate estimates, but require more multiplications and thus trigger more bootstrappings, which is the main source of latency during homomorphic inference.

For each type of nonlinearity, existing methods (de Castro et al., 2025; Moon et al., 2025; Zhang et al., 2025; Yu et al., 2026; Zhang et al., 2026) select a single iteration parameterization for the approximation and apply it indiscriminately at every occurrence in the model. Because the chosen approximation must remain accurate for the hardest occurrence, its iteration count is *chosen for the worst case*, which pays for precision it does not need. The result is far more bootstraps than the model actually requires. To remedy this, one solution is to tailor the number of iterations to each site. Yet even a per-site adjustment remains suboptimal when each site is held to the same approximation-quality target (Nam et al., 2025), since what ultimately matters is the *overall task loss* rather than the per-site accuracy (Lee et al., 2026). Searching the per-layer parameters directly against a task loss deviation proxy (Xie et al., 2026) moves the target from the site to the network, but with the weights frozen, the search can only harvest the tolerance the pretrained model already has. Alternatively, instead of adjusting approximations, some works redesign or retrain the model around `FHE` rather than encrypting (and thus approximating) existing architectures unchanged. Some methods train models with polynomial activations (Baruch et al., 2024; Ling et al., 2025) or introduce layer architectures that require fewer operations to be approximated (Ghodsi et al., 2020; Jha et al., 2021; Rho et al., 2025), while others distill pretrained models into encryption-friendly variants (Park et al., 2025). However, these approaches alter the model's structure or trigger expensive retraining.

**Contribution.** In this work, we introduce Homomorphic Encryption-Aware Training (`HEAT`), a pre-deployment technique that combines per-site iteration selection with fine-tuning (Figure 1). Building on Banino et al. (2021), `HEAT` learns the iteration count at each approximation site jointly with the model's weights. A depth regularization term pushes every site towards fewer iterations, while the task loss retains precision where the model needs it. Because the weights train under the same approximations, they absorb part of the error that shallower solvers introduce, so that each site can be cut further than a fixed model would tolerate. The resulting model is architecturally identical to its plaintext parent but demands substantially less multiplicative depth under `FHE`.

Nothing guarantees a priori that a network can co-adapt to the error of a shallow iterative solver in place of the exact nonlinearity. We motivate it by relating the error `HEAT` trains with to the error quantization-aware training (`QAT`) learns to tolerate. We show that the worst-case error of an $n$-step approximation matches that of a quantizer with bit width $\Theta(2^n)$, *i.e.* each iteration roughly doubles the number of correct bits (Proposition 1). Moreover, for any fixed set of weights, running the network with $n$-step approximants and running it with weights quantized to the matching precision yield losses that differ by a term vanishing doubly exponentially in $n$ (Theorem 1). At fixed weights, the objective `HEAT` optimizes, in which every site learns its own count, can thus be read as a per-layer quantization objective, and the learned counts as an allocation of numerical precision across the network layers. Our contributions are threefold:

- We introduce `HEAT`, a fine-tuning method that co-adapts model weights and iteration counts using standard gradient-based optimization. On encrypted `GPT-2` it reduces the total approximations' iterations by $3.1\times$.
- We establish a formal relation between iterative approximation and quantization, and bound, at fixed weights, the difference between the objective under iterative approximations and the `QAT` objective at the corresponding precision.
- We evaluate `HEAT` on teacher forced encrypted decoding against the same model under calibrated approximations. `HEAT` reduces end-to-end latency from $\approx 54\,\mathrm{s/tok}$ to $\approx 38\,\mathrm{s/tok}$ ($1.4\times$) while raising top-1 fidelity to the plaintext model from $70\%$ to $83\%$.

The theoretical results are machine-verified in Lean 4. Code and formal proofs published at https://anonymous.4open.science/r/he-aware-training/, model weights will be released upon publication.

## 2 BACKGROUND

**Threat Model.** We have two parties: the client $C$, who wants to prompt the model, and the server $S$, which owns the model. $C$ does not want to reveal the prompt to $S$, and $S$ does not want the model to leave its trust zone[1]. We operate in the honest-but-curious model for $S$, *i.e.* the adversary $S$ follows the protocol but tries to infer information about the privacy-sensitive prompt from the interaction and the computation. We assume the client generates a public–private cryptographic key pair and shares the public key with $S$. The prompt is tokenized and encrypted locally by $C$ before being sent to $S$, so $S$ never sees the plaintext tokens, intermediate activations, outputs, or KV cache, and returns encrypted logits. Additionally, the computation performed by $S$ must be oblivious, *i.e.* neither control flow nor memory access patterns can depend on the $C$'s plaintext prompt. Hardware side channels and metadata leakage, such as sequence length, are outside our threat model.

**Fully Homomorphic Encryption (`FHE`)** allows a server to evaluate arithmetic operations directly on encrypted values. We use `CKKS` (Cheon et al., 2017; 2018), an `FHE` scheme designed for approximate arithmetic over real and complex data. If $c_1 \leftarrow \mathrm{Enc}_{pk}(p_1)$ and $c_2 \leftarrow \mathrm{Enc}_{pk}(p_2)$, then

$$\mathrm{Dec}_{sk}(c_1 \oplus c_2) \approx p_1 + p_2, \quad \mathrm{Dec}_{sk}(c_1 \otimes c_2) \approx p_1 p_2, \quad \mathrm{Dec}_{sk}(\mathrm{Rot}_k(c_1)) \approx \pi^k(\mathbf{p}_1), \tag{1}$$

where $\oplus$, $\otimes$, and $\mathrm{Rot}_k$ denote encrypted addition, element-wise multiplication, and cyclic slot rotation by $k$ positions, respectively. Multiplications can be composed a finite number of times fixed by the scheme's cryptographic parameters, inducing a quantity known as *multiplicative depth*. Once the available multiplicative depth is exhausted, a *bootstrapping* refreshes the ciphertext and restores its computational capacity (Gentry, 2009). Bootstrapping costs two to three orders of magnitude more than any other primitive, so multiplicative depth is a primary factor of encrypted-inference latency.

**Iterative approximations.** Under `CKKS`, only additions, multiplications, and rotations can be evaluated directly, so nonlinear operations (division, inverse square root, activation functions…) must be approximated with methods exploiting their composition. Common choices include Goldschmidt (Goldschmidt, 1964) iterations for division, Newton-Raphson iterations for $1/\sqrt{x}$ (Qu & Xu, 2023), and polynomial approximations based on Chebyshev or Remez fitting. Given a nonlinear function $f(x)$ evaluated at $x$, an iterative method produces increasingly accurate estimates from a seed value $F_0(x)$ that converge to $f(x)$ in the limit

$$F_0(x), F_1(x), \ldots, F_n(x) \to f(x), \tag{2}$$

where $F_i(x)$ is the approximation after $i$ refinement steps and $n$ is the total iteration count. For quadratically convergent methods such as Goldschmidt and Newton-Raphson, the error satisfies

$$\varepsilon_n := \sup_{x \in \mathcal{D}} |F_n(x) - f(x)| \leq M r^{2^n}, \qquad M > 0,\ r \in (0, 1). \tag{3}$$

Where $\mathcal{D}$ denotes the set of inputs for which the chosen initialization $F_0(x)$ places the iteration strictly in its convergence interval. The constants $M$ and $r$ depend on the iterative method, its initialization, and the considered domain. Accuracy improves doubly exponentially with $n$, but each additional iteration requires an extra fixed number of multiplications. Therefore, the approximation error decreases rapidly, while its multiplicative depth grows linearly. Practical `FHE` implementations combine several such methods to balance precision, depth, and noise (Qu & Xu, 2023).

**Quantization-aware training (`QAT`)** provides the conceptual reference for `HEAT`. Both expose a model during training to the numerical error it will encounter at deployment, allowing its weights to adapt. Quantization represents weights and, in some settings, activations at lower precision to reduce inference cost. Applying it only after training can degrade accuracy, so `QAT` inserts low-precision proxies into the forward pass during training or fine-tuning (Jacob et al., 2018). The underlying floating-point parameters are still updated by backpropagation, commonly using a straight-through estimator for non-differentiable rounding. The model therefore learns to tolerate the bounded error introduced by its deployment-time computation.

[1]The weights may be proprietary, in which case $S$ has a commercial interest in keeping them opaque, or open, in which case $S$ may still require that they remain within its infrastructure.

# 3 METHOD

FHE natively supports only addition, rotation, and multiplication, and multiplications may be composed only a bounded number of times before a costly bootstrapping operation is needed to restore the multiplicative budget. Every other function a model relies on (LayerNorm, Softmax, ...) must be assembled from these primitives (Appendix D). Specifically, during encrypted inference, these functions are replaced by iterative methods that approximate them directly to their components. The parameterization of the iterative methods, and especially the total iterations performed, introduces a trade-off between accuracy and latency. Increasing iterations tends to preserve the model's behavior, producing more accurate approximations, but at the same time means more multiplications and thus more costly bootstrappings.

Instead of selecting a single iteration parameterization for each approximation or tuning it to a fixed precision target, we propose to model each iteration count as a *parameter* in its own right. We introduce iterative approximation in the plaintext model, during a pre-deployment stage we call HEAT. We fine-tune the iteration counts *jointly with the weights* to meet the final objective, so that the network adapts to the error it will encounter at inference, rather than merely tolerating it. In this manner, each site can be cut further still, since the weights will absorb the error that the iteration reduction introduces.

## 3.1 MAKING ITERATION SELECTION DIFFERENTIABLE

Let $f$ be a nonlinear function and $F_0, F_1, \ldots, F_n$ the successive results of an iterative scheme converging (in the limit) to $f$, so that $F_i$ is the approximation obtained after $i$ iterations. Substituting $F_i$ for $f$ in the network turns the iteration count $i$ into a *discrete* hyper-parameter that is non-optimizable directly through gradient methods. We make the parameter $i$ differentiable by following Banino et al. (2021)'s adaptive-computation methods. Specifically, once the approximations set for future FHE deployment is defined, we implement it in plaintext and, at each approximation site, we introduce a learnable distribution $\mathbf{p} \coloneqq (p_0, \ldots, p_n)$ over the iterations, and forward into the next layer the expected approximation under this distribution

$$\widetilde{f}(x) = \mathbb{E}_{i\sim p}\left[F_i(x)\right] = \sum_{i=0}^{n} p_i F_i(x)\,. \tag{4}$$

The loss thereby becomes differentiable in the distribution and in the model weights alike, making the two *co-adapt* during fine-tuning. Because the $F_i$ are successive states of the same solver, they are all produced in a single execution; during the forward pass we checkpoint only the solvers' inputs and not their intermediate values, which we recompute during the backward, keeping the additional memory costs small (Appendix C.2).

**Iteration count penalty.** To incentivize shallow approximations, for every site $s \in S$, we penalize the Kullback–Leibler (KL) divergence of the learnable distributions $\mathbf{p}^{\mathbf{s}}$ from a target negative-binomial prior $\mathbf{p}^{\mathbf{s},\star}$ (whose mode sits at a low $i$) and add it to the task loss

$$\mathcal{L}_{\text{iter}} = \frac{1}{|S|}\sum_{s\in S} \mathrm{KL}(\mathbf{p}^{\mathbf{s}} \,\|\, \mathbf{p}^{\mathbf{s},\star})\,, \quad \mathcal{L}_{\text{HEAT}} = \mathcal{L}_{\text{task}} + \lambda_{\text{iter}}\mathcal{L}_{\text{iter}} + \lambda_{\text{range}}\mathcal{L}_{\text{range}}\,. \tag{5}$$

The $\mathcal{L}_{\text{HEAT}}$ introduces a tension between task accuracy and reduction in iterations which, through gradient descent optimization, finds an equilibrium induced by the coefficients $\lambda_{\text{range}}$ and $\lambda_{\text{iter}}$, where $\mathcal{L}_{\text{range}}$ is the activations' squeeze term, from Zimerman et al. (2024b) and Baruch et al. (2024).

**Adaptive depth target.** Applying strong depth pressure from the beginning of training can force the model onto inaccurate approximations before its weights have co-adapted, or driving activations outside iterative methods' convergence domain which results in unstable and slow training. Therefore we initialize each learnable distribution (Figure 2) such that the expected approximation is heavily biased by high iteration counts' contributions, de facto simulating high precision; but leaving a non-zero tail to let gradient flow through lower iterations' contributions. Likewise, at the onset, the priors' modes are not fixed to encourage extremely coarse approximations, but instead they follow an adaptive back-off that decreases the targets' modes every time the learnable distributions reach it.

**Fine-tuning schedule.** HEAT is applied to a plaintext pretrained model in three phases. Phase one acts as a warm-up during which we train under $\mathcal{L}_{\text{task}} + \lambda_{\text{range}}\mathcal{L}_{\text{range}}$ alone with no iteration term, reducing activation ranges to prune outlier values, and thus producing a so-called *squeezed* baseline (see Table 2). Phase two is the main HEAT functionality: we attach a learnable distribution to every iterative solver's site (Appendix D) and optimize with full $\mathcal{L}_{\text{HEAT}}$, with the prior's mode adaptively backing off as the learned mode settles on it. Phase three (*cool-down*) consolidates the result, each approximator is frozen to the mode of its learned distribution $i^{s,\star} = \arg\max_i p_i^s$, and the weights are fine-tuned on these *fixed* counts with $\mathcal{L}_{\text{iter}}$ off, consolidating them at the counts to be deployed. At inference, this count is shared by all inputs, as data-dependent execution would leak information through latency, so the encrypted circuit evaluates only $F^s_{i^{s,\star}}(x)$.

### 3.2 Approximation Depth as Numerical Precision

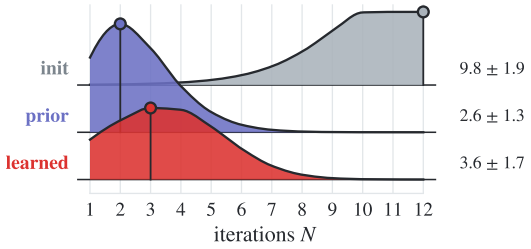


Figure 2: **Learned distribution.** Evolution of the learnable distribution $\mathbf{p}$ for one LayerNorm approximation. Training shifts probability mass from the full-depth seed toward shallower approximations; deployment selects the learned mode.

Neural networks are flexible function approximators that can tolerate a certain degree of noise introduced in the forward stream. A field that has proven this statement widely is quantization, where quantization-aware training (QAT) adapts weights jointly with the task loss and the quantization error. We show that an $n$-step iterative approximation has the same worst-case error as a uniform quantizer whose bit width is set by the iteration count, and that, at fixed weights, a network run with such approximations and the same network run with matched-precision quantized weights have task losses that differ by a term vanishing doubly exponentially in the count. More broadly, this analogy lets us study fine-tuning with iterative approximation through the lens of quantization, so that questions about approximation depth become questions about numerical precision.

Consider an $n$-step approximation $F_n$ of a nonlinearity $f$, satisfying Equation (3). A uniform $b$-bit quantizer over a range of width $R$ maps elements into representatives taken uniformly with step size $s = R/(2^b - 1)$ over $\mathbb{R}$. The representation, for elements in the interval, has a worst-case error $s/2$, and matching this error to $\varepsilon_n$ converts iteration count into an effective bit width.

**Proposition 1** (Iterations are bits)**.** *To obtain the same worst case error from an iterative approximation with a uniform $b$-bit quantizer we need to set:*

$$b = \log_2\left(1 + \frac{R}{2Mr^{2^n}}\right) \approx 2^n \log_2 \frac{1}{r} + \log_2 \frac{R}{2M}. \tag{6}$$

*Hence $b = \Theta(2^n)$,* i.e. *each additional iteration approximately doubles the number of correct bits.*

*Proof* and details in Appendix B.1. It is worth noting that the two mechanisms inject the error from two different sources. Iterative approximation acts directly on the activations, whereas weight quantization perturbs them indirectly through the linear transformation. However, under bounded activations and Lipschitz nonlinearities, the induced per-layer errors can be matched (Appendix B). Given that models' blocks are $\rho$-Lipschitz functions, and that per-layer error induced by either quantization or approximations is at most $\varepsilon_n$, by induction, the final activation is perturbed by at most $\varepsilon_n \sum_{j=0}^{L-1} \rho^j$. We can thus bound, at fixed weights, the task-loss difference between the two models:

**Theorem 1** (Matched-precision objective bound)**.** *Let $\mathcal{L}^{(n)}_{\text{approx}}$ denote the task loss of a network using $n$-step iterative approximations, and let $\mathcal{L}^{(b)}_{\text{QAT}}$ denote the task loss of the same network, with the same weights, under $b$-bit weight quantization. If the loss is $\xi_{\text{lip}}$-Lipschitz in the final activations and $b = b^\star(n) = \Theta(2^n)$ is chosen to match the per-layer error, then*

$$\left|\mathcal{L}^{(n)}_{\text{approx}} - \mathcal{L}^{(b)}_{\text{QAT}}\right| \le 2\xi_{\text{lip}} M r^{2^n} \sum_{j=0}^{L-1} \rho^j \tag{7}$$

*and the bound vanishes doubly exponentially with $n$.*

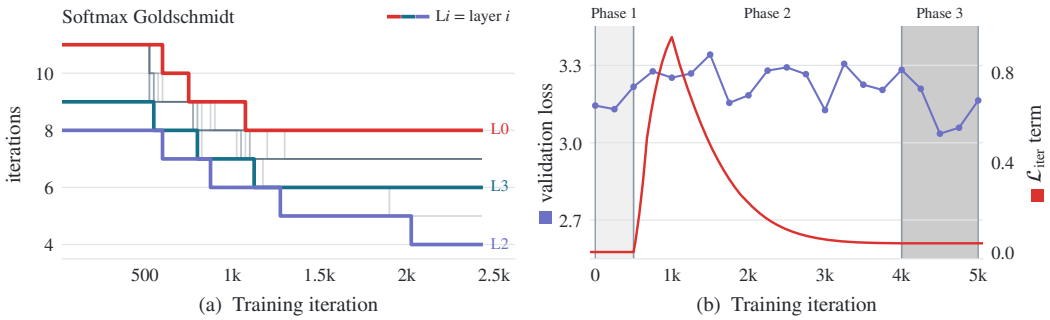


Figure 3: (a) Iteration counts of the Softmax's Goldschmidt refinement loop across layers during HEAT. The descent is not uniform, each layer settles at the depth its approximation demands. (b) HEAT fine-tuning dynamics: the validation loss stays flat while the per-token iteration budget $\mathcal{L}_{\text{iter}}$ decreases. The $\mathcal{L}_{\text{iter}}$ terms rises as the prior's success probability increases $0.6 \to 0.9$ and decays as the distributions follow it. Validation stays roughly constant (3.0–3.3 band) during fine-tuning.

*Proof* in Appendix B.2. The theorem is stated using a single iteration count (and thus bit-width) across the model for clarity. At deployment, HEAT fixes each site to its learned count $i^{s,\star}$, so the selected iteration depths can be interpreted as a learned allocation of numerical precision across the network. As in QAT, the weights adapt to the numerical error encountered at deployment; unlike conventional quantization, HEAT optimizes multiplicative depth rather than bit width.

## 4 Experiments

When compiling a network for an FHE pipeline, iterative approximations must be introduced to compensate for the limited (native) operation set supported by encrypted computation (Appendix D), and any additional compute required further slows and weighs down the system due to increased bootstraps and memory footprint. At our deployed parameters (Appendix C.1), a single bootstrap costs two to three orders of magnitude more than any other CKKS (Cheon et al., 2017; 2018) primitive (Table 3), raising two questions: *How fast does encrypted inference run? How well is the original model's performance preserved?*

These questions guide the design of our experimental setup, where we compare a per-site calibrated GPT-2 (124M) against our HEAT fine-tuned model. All latency and fidelity results of this section are measured under deployed end-to-end encrypted inference (thus accounting for any integration overhead) with CKKS at 128-bit security on a FIDESlib-based (Agulló-Domingo et al., 2025) backend. Packing follows Cachemir (Yu et al., 2026), nonlinearities use THOR-style approximations (Appendix D, Moon et al. 2025), arg max follows Avitan et al. (2025), and the bootstrap schedule is produced by a single shared algorithm for every set of weights. Every experiment runs on a single custom A100-64GB, 8 cores of an Intel Xeon Platinum 8358 and 128GB of RAM (Appendix C).

**Baselines.** The accuracy of an encrypted model depends on how well its nonlinearities are approximated, which depends on how their iteration counts are set. Existing methods in the literature choose a single parameterization per type of nonlinearity and apply it at every occurrence in the model. Because this single parameterization must accommodate the *worst-case* site, every other site pays (in computation) for precision it does not need, inflating bootstrapping cost. A per-site calibration mitigates this, and we adopt it as our calibrated baseline; each site receives the smallest iteration count that meets a global precision target $\varepsilon \in \{10^{-i}\}_{i=1,\dots,4}$ on its own input range (over a set of 128 sequences). We report results with $\varepsilon$=$10^{-4}$, as it empirically returns the best stability-quality trade-off over long sequences. Furthermore, we compare with ATLAS (Xie et al., 2026), which replaces per-site precision target with a network-level objective; we run its search from the calibrated circuit on frozen weights. By contrast, HEAT fine-tunes the model to acquire that tolerance.

**HEAT fine-tuning.** We fine-tune with HEAT a GPT-2 initially calibrated at $\varepsilon$=$10^{-4}$. The three phases take 7 GPU-hours in total, processing $\approx 3.5 \times 10^7$ OpenWebText's tokens (under 1% of GPT-2's pretraining corpus), thanks to the forward-mode depth gradients described in Appendix C.2, training with iterative approximations induces negligible memory-compute

Table 1: Encrypted decode at deployment. *it/fwd*: iterations of the iterative approximations per forward; *bts/tok* and *s/token*: bootstraps and wall time per generated token; *fidelity* over 64 paired OpenWebText sequences of 128 teacher-forced positions, each model against its own plaintext logits. *Coll.* denotes collapsed sequences, *i.e.* those in which more than half of the final states diverged during encrypted evaluation. *Top-1* is computed over all sequences, tokens that fail to decrypt counting as misses. *Perplexity* is computed over stable sequences and measured over the 128 tokens; subscript is the perplexity of the same circuit evaluated in plaintext. ⋆Uses ring dimension $N{=}2^{17}$ and 20 usable levels, not reproduced by us. †Search run from the calibrated circuit; cheapest point within 2% of its plaintext perplexity; its gap under encryption is an encrypted effect (Appendix E.5).

| | circuit | | s/token | | fidelity | | perplexity |
|---|---|---|---|---|---|---|---|
| **System** ($\lambda$=128) | it/fwd(↓) | bts/tok(↓) | Transf.(↓) | e2e(↓) | coll.(↓) | top-1(↑) | $\texttt{FHE}_{\text{(plain)}}$(↓) |
| `GPT-2` (plaintext) | – | – | – | – | – | – | 32.4 |
| de Castro et al. (2025)⋆ | – | – | 60.0 | ≈68 | – | – | – |
| `GPT-2` (calibrated $10^{-4}$) | 712 | 513 | $45.0_{(\pm 0.3)}$ | $54.3_{(\pm 0.4)}$ | 7/64 | $70.3_{(\pm 2.6)}$ | $44.6_{(31.7)}$ |
| `ATLAS` (Xie et al., 2026)† | 438 | 396 | $34.5_{(\pm 0.3)}$ | $43.8_{(\pm 0.4)}$ | 13/64 | $58.1_{(\pm 3.0)}$ | $50.5_{(33.1)}$ |
| **`HEAT` (ours)** | **228** | **326** | $\mathbf{28.2}_{(\pm 0.2)}$ | $\mathbf{37.6}_{(\pm 0.3)}$ | **0/64** | $\mathbf{82.9}_{(\pm 0.5)}$ | $\mathbf{30.7}_{(31.0)}$ |

costs. The learnable distributions are initialized as detailed in Appendix C.4 and the network is trained with AdamW (Loshchilov & Hutter, 2019); validation is the mean over 32 held-out batches every 250 updates. The full configuration is available in the released code. Figure 3a shows how iteration counts evolve during fine-tuning, taking Goldschmidt's iterations in the Softmax as an example. Depth is learned layer by layer, settling at different values. `HEAT` lowers precision only where the weights can absorb it rather than compressing every site uniformly; under the `HEAT-QAT` analogy, this is a learned mixed-precision allocation. Summed over the network, the learned circuit runs 228 solver iterations per forward against the calibrated 712, a 3.1× reduction (Figure 7). Figure 3b traces the corresponding loss. The $\mathcal{L}_{\text{iter}}$ term is held off during phase one warm-up, and it ramps in as the prior anneals to put progressively stronger drag on the counts; throughout, the validation loss stays within the 3.0–3.3 band, *i.e.* the model releases iterations without bartering perplexity for them, thanks to co-adaptation.

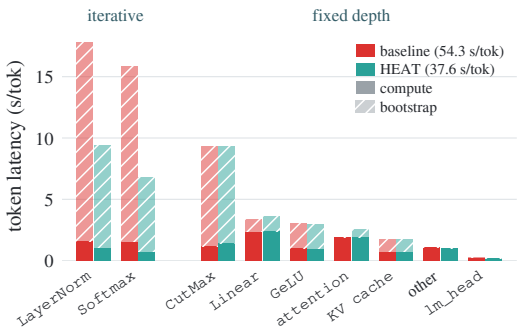


Figure 4: Per-family decode-latency composition, calibrated `GPT-2` vs. `HEAT`: paired bars in absolute seconds per token, the hatched cap of each bar being the bootstrap time spent inside that family. In the baseline, the iterative nonlinearities own 59% of the token wall and 71% of all bootstrap time; `HEAT`'s saving concentrates in exactly those bars. `CutMax` is excluded from optimization.

Fewer iterations consume fewer levels per forward pass, and fewer levels mean fewer bootstraps; Figure 4 breaks down the decode latency by operator family. In the calibrated baseline, the two iterative nonlinearities, LayerNorm and Softmax, account for 59% of the per-token wall time and 71% of all bootstrap time. `HEAT`'s saving lands exactly on those two bars, roughly halving each, while the fixed-depth families (linear layers, GELU, attention, KV cache) are left untouched. Since bootstrapping inside the iterative methods dominates the encrypted forward pass, this is the cost `HEAT` is designed to reduce. Table 1 reports the main latency-quality results. `HEAT` executes 326 bootstraps per token against the baseline's 513 (1.6× reduction), and serves a token in 37.6 s/tok against the calibrated baseline's 54.3 s/tok (1.4× reduction) and `EncryptedLLM`'s (de Castro et al., 2025) ≈68 s/tok, whose iteration counts are fixed and uniform across the model. The encrypted arg max's (Avitan et al., 2025) calibration is shared by every approach and contributes the same 9.3 s/tok to each row.

Latency is half the deployment question; the other half is whether the shallower circuit remains faithful to its plaintext analog. Table 1 gives the perplexity of the encrypted inference; here we study in detail how degradation happens along the sequence. Figure 5 answers this over 64 teacher-forced

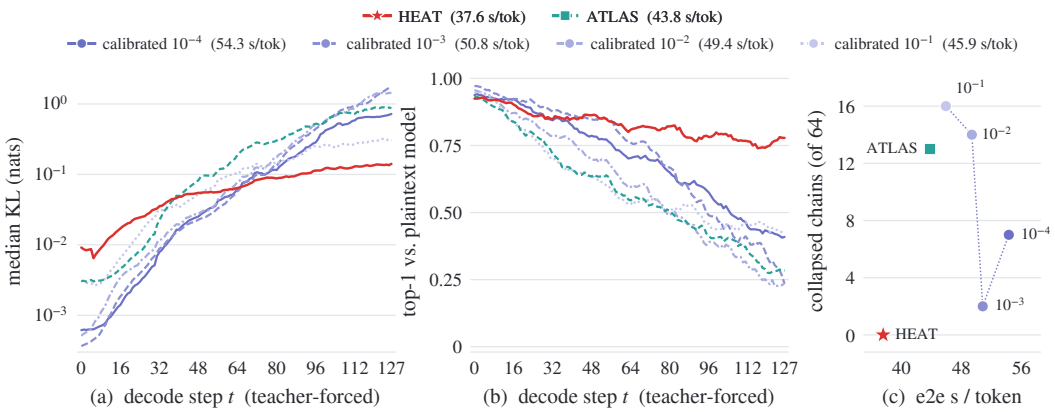


Figure 5: Per-position fidelity of each encrypted circuit against its own plaintext logits over 64 sequences of 128 teacher-forced decode steps. (a) Median encrypted KL over the sequences that do not collapse. The calibrated circuits start near-exact and degrade with position, the looser tolerances faster; HEAT starts higher and stays almost flat. (b) Top-1 agreement over all sequences, the calibrated ladder falls to 23–42% by $t$=128 while HEAT ends at 78%. Per-step latency is position-flat for every circuit under the Cachemir layout. (c) Collapsed sequences of 64 per method, *i.e.* sequences where more than 50% of the tokens couldn't be decrypted.

chains of 128 tokens, *i.e.* we test model preservation and stability under encryption. ATLAS is near-lossless in plaintext yet collapses on 13 of 64 sequences under encryption. This is an encrypted effect that a noise-free search objective cannot see (Appendix E.5). Along the sequence, the calibrated baseline starts near-exact and degrades. The median KL grows 395× from the first 32 positions to the last (0.0013 → 0.519) and top-1 agreement falls from 91% to 47%. HEAT starts from a slightly higher floor and grows slower (0.016 → 0.129; top-1 90% → 77%). Pooled over all decoded tokens, top-1 agreement is 70.3% for the baseline and 82.9% for HEAT, and over the last 32 positions HEAT's KL is 4× lower. The shallower circuit is thus not only faster but more faithful to its plaintext analog than the calibrated one is to its own, thanks to the approximation tolerance it learned and less overall bootstrap noise injection.

**Ablation study.** Table 2 separates HEAT's two losses, and explores two single-component alternatives: ① trains the weights with the range loss $\mathcal{L}_{range}$ *only* and lets the calibrator set the counts afterwards; ② trains the halting logits on frozen pretrained weights. ③ is the artisan's work, fixing every count by hand and fine-tuning the weights (397 is the shallowest circuit this recipe reaches without producing a circuit that always collapses as a starting point). Fine-tuning under $\mathcal{L}_{\text{range}}$ alone reduces only part of the latency by narrowing the approximation intervals and thus the iteration count needed to meet the precision target, but does not reach the co-adapted circuit HEAT learns. Conversely, transplanting HEAT's learned counts onto the ① *squeezed* and calibrated checkpoints breaks both in plaintext.

The two collapsing options, ② and ③, fail for different reasons. In ② the counts descend until samples diverge and carry no gradient, so the task loss offers no way back. The artisan's ③ works in plaintext with counts cut to what it just tolerates; under encryption every bootstrap adds error to every slot, compounding along the sequence until one approximation pushes the next input outside its convergence interval, the outputs explode and the sequence is lost. This is what motivates HEAT's gradual descent: the prior lowers the counts slowly enough that the weights adapt before any site leaves its convergence domain, and the range loss shrinks the approximations' operative margins so that fewer samples collapse in the first place. It is this combination that lets HEAT decrease the learned depths stably where either component alone cannot.

## 5 Related Work

**FHE frameworks.** Private transformer inference is either interactive, with protocols that split the computation between client and server (Knott et al., 2021; Hao et al., 2022; Li et al., 2022; Gupta et al., 2024; Pang et al., 2024; Dong, Ye et al., 2025; Kei & Chow, 2025; Lu et al., 2025;

Table 2: Ablation of HEAT's three components. The range loss alone gives a circuit that replicates its plaintext model but stays deep; learned depth alone, on frozen weights, cuts the iterations by a third (*vs.* the calibrate baseline) but blows up in 62 of 64 samples, the weights never having adapted to the shallow circuit. Only the combination is both shallow and stable.

| | $\mathcal{L}_{\text{range}}$ | $\mathcal{L}_{\text{iter}}$ | trained weights | it/fwd(↓) | e2e(↓) | coll.(↓) | perplexity $\texttt{FHE}_{\text{(plain)}}$ |
|---|---|---|---|---|---|---|---|
| ① squeezed | ✓ | ✗ | ✓ | 455 | 43.6 | 5/64 | $32.6_{(32.5)}$ |
| ② learned depth only | ✗ | ✓ | ✗ | 456 | 46.6 | 62/64 | – |
| ③ fixed counts | ✗ | ✗ | ✓ | 397 | 43.1 | 58/64 | – |
| **HEAT (ours)** | ✓ | ✓ | ✓ | **228** | **37.6** | **0/64** | $\mathbf{30.7}_{(31.0)}$ |

Hou et al., 2026), or non-interactive under FHE, where the server evaluates the whole model on ciphertext. The latter builds on CKKS (Cheon et al., 2017; 2018) and bootstrapping (Gentry, 2009); Badawi et al. (2026) systematize existing frameworks along a packing axis for linear blocks and a model-preservation axis for nonlinear ones. On the packing axis, NEXUS (Zhang et al., 2025), MOAI (Zhang et al., 2026), and STIP (Wang et al., 2026) cut rotations through column, diagonal, and complex-slot layouts; EncryptedLLM (de Castro et al., 2025) and Cachemir (Yu et al., 2026) target GPU-accelerated decoding, the latter with an encrypted KV cache and the THOR (Moon et al., 2025) nonlinear circuits we also adopt; GPU libraries (Agulló-Domingo et al., 2025) provide the primitives, and FHE compilers (Viand et al., 2021; Cheon et al., 2024; Krastev et al., 2024; Ebel et al., 2025) automate packing and bootstrap placement for a given circuit, though most surveyed frameworks still place bootstraps by hand. All of these take the pretrained model as given; HEAT is a fine-tuning that yields a model tolerant of a shallower circuit, and deploys on any of them.

**FHE-friendly approximations.** Since CKKS exposes only additions, multiplications, and rotations, nonlinearities must be rebuilt from them. Badawi et al. (2026) split frameworks into *model-preserving* ones, which approximate the original layers, and *model-modifying* ones, which substitute them and retrain. In the first class, division and inverse square root use Goldschmidt (Goldschmidt, 1964) and Newton iterations (Qu & Xu, 2023), Softmax uses a scaled exponential with an iterative reciprocal (Lee et al., 2023; Moon et al., 2025), GELU a polynomial composition (Moon et al., 2025), and the decoding $\arg\max$ iterative sharpening (Avitan et al., 2025). In the second, networks are trained with polynomial activations (Gilad-Bachrach et al., 2016; Baruch et al., 2024; Zimerman et al., 2024b; Ling et al., 2025), ReLU counts are pruned or searched (Ghodsi et al., 2020; Jha et al., 2021), Softmax is replaced by a Gaussian kernel (Rho et al., 2025) or a power function (Zimerman et al., 2024a; Park et al., 2025), and normalization is dropped altogether (Zhu et al., 2025).

**FHE iterative approximations reduction.** Most systems fix one iteration count or polynomial degree per operator type, hand-tuned for the hardest site. SLOTHE (Nam et al., 2025) approximates only the non-arithmetic sub-parts of each function under a tunable accuracy-latency cost model, Lee et al. (2026) choose per-layer polynomial degrees by dynamic programming over each layer's estimated impact on accuracy, ELLMo (Güzelhan et al., 2026) redesigns Softmax and LayerNorm to cut their depth, and ATLAS (Xie et al., 2026) searches per-layer iteration counts and degrees against a network-level objective. All operate on frozen weights and can only harvest the tolerance the pretrained model already has.

## 6 Conclusions

Encrypted inference pays for precision it does not need, as every iterative approximation is usually set to a fixed count, and with the weights frozen, that count can only be lowered as far as the pretrained model already tolerates. HEAT lifts this limit by making the counts learnable and training the weights under the approximations they will meet at deployment. On encrypted GPT-2 decoding, the learned circuit runs 3.1× fewer solver iterations, 1.6× fewer bootstraps, and serves a token in 37.6 s against 54.3 s, while its encrypted perplexity stays within 0.3 of plaintext, where the calibrated circuit drifts substantially. Neither learned depth nor weight adaptation can reach this alone. Proposition 1 and Theorem 1 relate iteration count to an effective bit width and bound, at fixed weights, the gap to mixed-precision QAT; this frames learning the counts as a precision allocation and motivates, without guaranteeing, that fine-tuning absorbs the error. The result is a model architecturally identical to its parent that any CKKS pipeline could deploy.

## USE OF AI

Large language models were used extensively to write and refactor the training code, the `CKKS` backend integration, and the Lean 4 formalization. All code was reviewed and run by the authors, every proof was checked by the Lean kernel, and the method, experiments, and analysis are the authors' own, who take full responsibility for the content.

## ETHICAL STATEMENT

Lowering the cost of encrypted inference lets users in privacy-sensitive settings query hosted models without disclosing their data. The same property impacts the provider's ability to inspect prompts or outputs, so server-side content filtering, abuse detection, and audit cannot be applied, and a faster encrypted pipeline serves bad actors as readily as hospitals. `HEAT` does not create this trade-off, which is intrinsic to `FHE`, but brings it closer to practice. Our threat model also does not hide sequence length or protect the weights; Appendix F discusses these points further.

## REPRODUCIBILITY STATEMENT

Code, Lean 4 proofs, and all configurations are at https://anonymous.4open.science/r/he-aware-training/; weights will be released upon publication. Cryptographic parameters, initialization, and schedule are in Appendix C, the circuits in Appendix D. Fine-tuning takes 7 GPU-hours on one `A100`; every latency and encrypted-fidelity number (Tables 1, 2, 5 and 8, Figures 4 and 5) comes from a deployed `CKKS` run on our modified `FIDESlib` build (OpenFHE 1.4.2, CUDA 12.6); the sensitivity, downstream, drift and scaling analyses of Appendix E are plaintext evaluations of the deployed circuits and are labeled as such.

## REFERENCES

Carlos Agulló-Domingo, Óscar Vera-López, Seyda Nur Güzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar, Rashmi S. Agrawal, David R. Kaeli, Ajay Joshi, and José L. Abellán. Fideslib: A fully-fledged open-source FHE library for efficient CKKS on gpus. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2025, Ghent, Belgium, May 11-13, 2025*, pp. 1–3. IEEE, 2025. doi: 10.1109/ISPASS64960.2025.00045. 6, 9

Matan Avitan, Moran Baruch, Nir Drucker, Itamar Zimerman, and Yoav Goldberg. Efficient decoding methods for language models on encrypted data. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP-AACL 2025, Mumbai, India, December 20-24, 2025*, pp. 1777–1794. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-IJCNLP.111. 6, 7, 9, 22

Ahmad Al Badawi, Andreea Alexandru, Yuriy Polyakov, and Vinod Vaikuntanathan. SoK: Private LLM inference using approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2026/935, 2026. URL https://eprint.iacr.org/2026/935. 9

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *CoRR*, abs/2107.05407, 2021. URL https://arxiv.org/abs/2107.05407. 2, 4, 20

Moran Baruch, Nir Drucker, Gilad Ezov, Yoav Goldberg, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. Polynomial adaptation of large-scale cnns for homomorphic encryption-based secure inference. In Shlomi Dolev, Michael Elhadad, Miroslaw Kutylowski, and Giuseppe Persiano (eds.), *Cyber Security, Cryptology, and Machine Learning - 8th International Symposium, CSCML 2024, Be'er Sheva, Israel, December 19-20, 2024, Proceedings*, volume 15349 of *Lecture Notes in Computer Science*, pp. 3–25. Springer, 2024. doi: 10.1007/978-3-031-76934-4_1. URL https://doi.org/10.1007/978-3-031-76934-4_1. 2, 4, 9

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *23rd International Conference on the Theory and Applications of Cryptology and Information Security*, volume 10624 of *Lecture Notes in Computer Science*, pp. 409–437. Springer, 2017. doi: 10.1007/978-3-319-70694-8_15. URL https://doi.org/10.1007/978-3-319-70694-8_15. 3, 6, 9

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pp. 347–368. Springer, 2018. doi: 10.1007/978-3-030-10970-7_16. URL https://doi.org/10.1007/978-3-030-10970-7_16. 3, 6, 9, 19

Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 6993–7010, Philadelphia, PA, 2024. USENIX Association. ISBN 978-1-939133-44-1. 9

Leo de Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. Encryptedllm: Privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. 2, 7, 9, 22

Dong, Ye, Lu, Wen-Jie, Zheng, Yancheng, Wu, Haoqi, Zhao, Derun, Tan, Jin, Huang, Zhicong, Hong, Cheng, Wei, Tao, Chen, Wen-Guang, and Zhou, Jianying. Puma: Secure inference of llama-7b in five minutes. *Security and Safety*, 4:2025014, 2025. doi: 10.1051/sands/2025014. URL https://doi.org/10.1051/sands/2025014. 8, 22

Austin Ebel, Karthik Garimella, and Brandon Reagen. Orion: A fully homomorphic encryption framework for deep learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 734–749, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710797. doi: 10.1145/3676641.3716008. URL https://doi.org/10.1145/3676641.3716008. 9

Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009. URL https://searchworks.stanford.edu/view/8493082. 2, 3, 9

Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020. 2, 9

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 201–210. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/gilad-bachrach16.html. 9

Robert E Goldschmidt. Applications of division by convergence, 1964. URL http://hdl.handle.net/1721.1/11113. 3, 9

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: secure GPT inference with function secret sharing. *Proc. Priv. Enhancing Technol.*, 2024(4):61–79, 2024. doi: 10.56553/POPETS-2024-0107. URL https://doi.org/10.56553/popets-2024-0107. 8

Seyda Nur Güzelhan, Lohit Daksha, Carlos Agulló-Domingo, Gilbert Jonatan, John Kim, José L. Abellán, David R. Kaeli, and Ajay Joshi. Ellmo: Packing- and depth-aware encrypted transformer inference. *IACR Cryptol. ePrint Arch.*, 2026:198, 2026. URL https://eprint.iacr.org/2026/198. 9

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/64e2449d74f84e5b1a5c96ba7b3d308e-Abstract-Conference.html. 1, 8

Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Jiawen Zhang, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party GPT inference. *IEEE Trans. Dependable Secur. Comput.*, 23 (3):6748–6762, 2026. doi: 10.1109/TDSC.2026.3667722. 9

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 2704–2713. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00286. 3

Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, pp. 4839–4849. PMLR, 2021. 2, 9

Andes Y. L. Kei and Sherman S. M. Chow. SHAFT: secure, handy, accurate and fast transformer inference. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. 8

Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 4961–4973, 2021. 8

Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sánchez. A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption. *Proc. ACM Program. Lang.*, 8(PLDI):126–150, 2024. doi: 10.1145/3656382. URL https://doi.org/10.1145/3656382. 9

Junghyun Lee, Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, Yongwoo Lee, Yongjune Kim, and Jong-Seon No. Optimized layerwise approximation for efficient private inference on fully homomorphic encryption. *Neurocomputing*, 695:134016, 2026. ISSN 0925-2312. URL https://www.sciencedirect.com/science/article/pii/S0925231226014141. 2, 9

Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19010–19035. PMLR, 2023. URL https://proceedings.mlr.press/v202/lee23m.html. 9

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric Xing, and Hao Zhang. MPCFormer: fast, performant and private Transformer inference with MPC, 11 2022. URL https://arxiv.org/abs/2211.01452v2. 1, 8

Huaming Ling, Ying Wang, Si Chen, and Junfeng Fan. Peregrine: One-shot fine-tuning for FHE inference of general deep cnns. *CoRR*, abs/2511.18976, 2025. doi: 10.48550/ARXIV.2511.18976. URL https://doi.org/10.48550/arXiv.2511.18976. 2, 9

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7. 7, 21

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. Bumblebee: Secure two-party inference framework for large transformers. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. URL https://www.ndss-symposium.org/ndss-paper/bumblebee-secure-two-party-inference-framework-for-large-transformers/. 1, 8

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: secure transformer inference with homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pp. 3765–3779. ACM, 2025. doi: 10.1145/3719027.3765150. 2, 6, 9, 21, 22

Kevin Nam, Youyeon Joo, Seungjin Ha, and Yunheung Paek. SLOTHE : Lazy approximation of Non-Arithmetic neural network functions over encrypted data. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3083–3102, Seattle, WA, August 2025. USENIX Association. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/nam-slothe. 2, 9

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: privacy-preserving, accurate and efficient inference for transformers. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pp. 4753–4771. IEEE, 2024. doi: 10.1109/SP54263.2024.00130. URL https://doi.org/10.1109/SP54263.2024.00130. 8

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11090–11111, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.543. URL https://aclanthology.org/2025.acl-long.543/. 2, 9

Hongyuan Qu and Guangwu Xu. Improvements of homomorphic secure evaluation of inverse square root. In *Information and Communications Security - 25th International Conference, ICICS 2023, Tianjin, China, November 18-20, 2023, Proceedings*, volume 14252 of *Lecture Notes in Computer Science*, pp. 110–127. Springer, 2023. doi: 10.1007/978-981-99-7356-9_7. 3, 9, 21, 22

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest K. Ryu, and Jung Hee Cheon. Encryption-friendly LLM architecture. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. 2, 9

Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pp. 1092–1108. IEEE, 2021. doi: 10.1109/SP40001.2021.00068. 9

Zihao Wang, Rongmao Chen, Xinwen Gao, Yi Wang, Lin Liu, Zixin Lan, Zhaoyu Wang, Shaojing Fu, Qiong Wang, and Xinyi Huang. STIP: efficient and secure non-interactive transformer inference via compact packing. *IACR Cryptol. ePrint Arch.*, 2026:174, 2026. URL https://eprint.iacr.org/2026/174. 9

Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. Ditto: Quantization-aware secure inference of transformers upon MPC. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53346–53365. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/wu24d.html. 1

Jianhang Xie, Sicheng Tan, Vishnu Naresh Boddeti, and Zhichao Lu. Atlas: Automated approximation of transformers for efficient homomorphic inference in one hour, 2026. URL https://arxiv.org/abs/2607.23478. 2, 6, 7, 9, 25

Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. Cachemir: Fully homomorphic encrypted inference of generative large language model with KV cache. *CoRR*, abs/2602.11470, 2026. doi: 10.48550/ARXIV.2602.11470. 2, 6, 9

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. 2, 9

Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, HUAXIONG WANG, Pu Duan, and Kwok-Yan Lam. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qJn4HtTzhH. 2, 9

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pp. 14901–14911. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/CVPR52734.2025.01388. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zhu_Transformers_without_Normalization_CVPR_2025_paper.html. 9

Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Meiri, and Omri Soceanu. Power-softmax: Towards secure LLM inference over encrypted data. *CoRR*, abs/2410.09457, 2024a. URL https://doi.org/10.48550/arXiv.2410.09457. 9

Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. Converting transformers to polynomial form for secure inference over homomorphic encryption. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62803–62814. PMLR / OpenReview.net, 2024b. 4, 9

# A LIMITATIONS

HEAT acts only on the iterative sites; the polynomial GELU, the linear families, and the encrypted $\arg\max$ (a fixed $9.3\,\text{s}/\text{tok}$) are left at their calibrated depth. We evaluate one language model and one vision transformer on a single A100; whether the gains carry to billion-parameter models or other approximation families is untested. Finally, HEAT needs gradient access and $3.5 \times 10^7$ tokens of fine-tuning.

# B PROOFS: (F)HE-AWARE TRAINING AND QUANTIZATION-AWARE TRAINING

A machine-checked Lean 4 formalization of the proofs in this appendix is provided at https://anonymous.4open.science/r/he-aware-training/proofs/HEEquivQAT.lean.

**Purpose.** HEAT fine-tunes a network while its nonlinearities are replaced by iterative approximants. On its own this is a new training regime, and nothing says a priori that a network can absorb the error of an iterative method, or that gradient descent should find weights that do. The results of this appendix do not answer that question, but they relate it to a known one, by *analogy* rather than by a new theory.

Proposition 1 shows that the worst-case error of an $n$-step quadratically convergent iterator is that of a uniform quantizer at $b^\star(n) = \Theta(2^n)$ bits, so the iteration count is a precision knob with a known exchange rate to bits.

Theorem 1 shows that, at fixed weights and matched precision, a network evaluated with iterative approximants and the same network evaluated with quantized weights differ in task loss by an amount that vanishes doubly exponentially in $n$, *i.e.* the two error-injection models have per-layer errors within the same budget.

Training under iterative approximations can therefore be read as quantization-aware training in the currency of multiplicative depth, and learning a per-site iteration count as mixed-precision allocation. What HEAT inherits from this analogy is the framing of QAT, which trains networks to tolerate a bounded per-layer perturbation of matched magnitude, rather than a new hypothesis about what networks tolerate.

**Iterative approximators.** Fix constants in $\mathbb{R}$, $M > 0$, $r \in (0, 1)$ and, for $n \in \mathbb{N}$, write

$$\varepsilon_n \;=\; M r^{2^n} \tag{8}$$

to be an iterator's error budget. As defined in Equation (3), the $n$-step iterator $F_n$ approximates the nonlinearity $f$ with

$$|F_n(x) - f(x)| \;\le\; \varepsilon_n \quad \text{for all } x \in \mathcal{D}, \tag{9}$$

where $\mathcal{D}$ is the set of inputs such that the iterator's seed strictly lies in the convergence interval. Since $0 < r < 1$, the sequence $\varepsilon_n$ is strictly decreasing and $\varepsilon_n \to 0$ doubly exponentially.

**Uniform quantizer.** A uniform quantizer $Q$ rounds to the nearest of $2^b$ uniformly spaced buckets on an interval $I$ of width $R > 0$, at bit width $b \in \mathbb{N}_{\ge 1}$ and with step $s(b) = R/(2^b - 1)$. For $w \in I$ it holds $|w - Q(w)| \le s/2$; outside $I$ the quantizer saturates and the bound fails, so we assume no overload, *i.e.* every entry of every $W_\ell$ lies in $I$. This is always possible with $I = [-\kappa, \kappa]$, since the row-norm bound below gives $|W_{ij}| \le \kappa$. Since $s(b)$ is decreasing in $b$, so is the worst-case rounding error. In classic neural network quantization, only weights are quantized; activations stay in full precision, so rounding reaches them only through the linear maps (Lemma B.1).

**Network's assumptions and definitions.** We define a layer as a linear map $\mathbb{R}^d \to \mathbb{R}^d$, followed by a nonlinear function applied component wise. Writing $f(u) \in \mathbb{R}^d$ for the entrywise application of $f\colon \mathbb{R} \to \mathbb{R}$ to $u \in \mathbb{R}^d$, the layer with weights $W \in \mathbb{R}^{d \times d}$ and nonlinearity $f$ is the map $h \mapsto f(Wh)$. Activations live in $(\mathbb{R}^d, \|\cdot\|_\infty)$. Given per-layer weights $W_1, \dots, W_L$ and an input $h_0$, we define the layers notation for the two possible error injection models discussed in (Section 3.2) the quantization

error $\cdot^Q$ and the approximation error $\cdot^H$:

$$h_\ell = f(W_\ell\, h_{\ell-1}), \qquad h_\ell^{\mathrm{H}} = F_n(W_\ell\, h_{\ell-1}^{\mathrm{H}}), \qquad h_\ell^{\mathrm{Q}} = f\big(Q(W_\ell)\, h_{\ell-1}^{\mathrm{Q}}\big),$$

Respectively the exact network, the `HEAT` network at fixed iteration count $n$, and the `QAT` network at bit width $b$.

We assume $f$ is $\lambda$-Lipschitz, with $\lambda > 0$, and every weight matrix has rows of $\ell_1$-norm at most $\kappa$, *i.e.* $\max_i \sum_j |W_{ij}| \le \kappa$. Together these make each layer $\rho$-Lipschitz with $\rho = \lambda\kappa$. We further assume the activations of the *quantized* network satisfy $\|h_\ell^{\mathrm{Q}}\|_\infty \le B$ for all $\ell \le L$ with $B > 0$, that the loss is $\xi_{\mathrm{lip}}$-Lipschitz in the final activations, and that every component of the pre-activation of the `HEAT` network, $(W_\ell\, h_{\ell-1}^{\mathrm{H}})_i$, lies in $\mathcal{D}$, so that (9) applies to it component wise. This last assumption is a hypothesis on the input. Calibration in practice does not imply it for every sample, and Table 2 shows the collapse that follows when it fails.

### B.1 Proof of Proposition 1

We restate the Proposition 1:

**Proposition B.1.** *Let $R > 0$ be the quantizer range width, let $M > 0$ and $r \in (0, 1)$ be the real valued constants of (9), and let $n \in \mathbb{N}$. Then $b$ satisfies $s/2 = \varepsilon_n$, with $s = R/(2^b - 1)$, if and only if*

$$b \;=\; b_n \;:=\; \log_2\!\Big(1 + \tfrac{R}{2Mr^{2^n}}\Big),$$

*Consequently $b_{n+1}/b_n \to 2$ as $n \to \infty$.*

*Proof.* The quantizer's worst-case error is $s/2 = R/\big(2(2^b - 1)\big)$; the iterator's is $\varepsilon_n = Mr^{2^n}$ by (9). Equating the two and solving for $b$,

$$\frac{R}{2(2^b-1)} = Mr^{2^n} \iff 2^b = 1 + \frac{R}{2Mr^{2^n}} \iff b = \log_2\!\Big(1 + \frac{R}{2Mr^{2^n}}\Big),$$

For the decomposition, write

$$1 + \tfrac{R}{2\varepsilon_n} = \tfrac{R}{2\varepsilon_n}\big(1 + \tfrac{2\varepsilon_n}{R}\big)$$

and take logarithms:

$$b_n = \log_2 \tfrac{R}{2\varepsilon_n} + \log_2\!\Big(1 + \tfrac{2\varepsilon_n}{R}\Big) = 2^n \log_2 \tfrac{1}{r} + \log_2 \tfrac{R}{2M} + \log_2\!\Big(1 + \tfrac{2\varepsilon_n}{R}\Big)$$

Since $\ln(1 + x) \le x$

$$\le 2^n \log_2 \tfrac{1}{r} + \log_2 \tfrac{R}{2M} + \tfrac{2\varepsilon_n}{R \ln 2}$$

Dividing by $2^n$, the last two terms vanish as $n \to \infty$, so $b_n/2^n \to \log_2 \frac{1}{r} > 0$; hence $b_n = \Theta(2^n)$ and $b_{n+1}/b_n \to 2$. □

### B.2 Proof of Theorem 1

Fix an input, and recall the three trajectories $h_\ell$, $h_\ell^{\mathrm{H}}$, $h_\ell^{\mathrm{Q}}$ of the definitions, and write

$$\delta_\ell^{\mathrm{H}} = \|h_\ell^{\mathrm{H}} - h_\ell\|_\infty, \qquad \delta_\ell^{\mathrm{Q}} = \|h_\ell^{\mathrm{Q}} - h_\ell\|_\infty.$$

the $\ell$-th layer difference with the non-approximated model. By Lemma B.1, weight rounding at bit width $b$ perturbs each output coordinate by at most $\lambda \frac{s(b)}{2}\, d\, B$. We fix $b$ at the *matched bit width*

$$b^\star(n) \;:=\; \min\big\{b \in \mathbb{N}_{\ge 1} : \lambda \tfrac{s(b)}{2}\, d\, B \le \varepsilon_n\big\},$$

the smallest integer bit width $b \ge 1$ at which the quantizer's per-layer error does not exceed the $n$-step iterator's. Since $\lambda\frac{s(b)}{2}dB$ is the worst-case error of a uniform quantizer over a range of width $\lambda dBR$, which is positive by the assumptions above, Proposition 1 applied with $R$ replaced by $\lambda dBR$ gives $b^\star(n) = \lceil b_n(\lambda dBR)\rceil \ge 1$, as $b_n > 0$. Here $b_n$ is a real-valued effective precision and $b^\star(n)$ its integer ceiling, so the rounding error at $b^\star(n)$ is at most, not equal to, $\varepsilon_n$. The replacement changes only the additive constant $\log_2 \frac{\lambda dBR}{2M}$, and $b_n \le b^\star(n) < b_n + 1$, so $b^\star(n) = \Theta(2^n)$.

The proof of the theorem has four steps. First the exact layers are Lipschitz with an explicit constant (Step 0), each discretization injects a per-layer error of at most $\varepsilon_n$ relative to the exact layer (Steps 1–2), these errors accumulate identically across depth (Step 3), and the loss inherits the resulting activation bound (Step 4).

**Theorem B.1** (Matched-precision objective bound,Theorem 1)**.** *Let $\mathcal{L}^{(n)}_{\text{approx}}$ denote the task loss of a network using $n$-step iterative approximations, and let $\mathcal{L}^{(b)}_{\text{QAT}}$ denote the task loss of the same network, with the same weights, under matched $b$-bit weight quantization. If the loss is $\xi_{\text{lip}}$-Lipschitz in the final activations and $b = b^\star(n)$ is the matched bit width, then*

$$\left|\mathcal{L}^{(n)}_{\text{approx}} - \mathcal{L}^{(b)}_{\text{QAT}}\right| \le 2\xi_{\text{lip}} M r^{2^n} \sum_{j=0}^{L-1} \rho^j. \tag{10}$$

*The bound vanishes doubly exponentially with $n$.*

*Proof. Step 0.* For any $x, y$ and any weight matrix with rows of $\ell_1$-norm at most $\kappa$, coordinate wise

$$|f((Wx)_i) - f((Wy)_i)| \;\le\; \lambda\Big|\sum\nolimits_j W_{ij}(x_j - y_j)\Big| \;\le\; \lambda \sum\nolimits_j |W_{ij}|\, \|x-y\|_\infty \;\le\; \lambda\kappa\, \|x-y\|_\infty,$$

so each exact layer is $\rho$-Lipschitz with $\rho = \lambda\kappa$.

*Step 1 (per-layer error of the iterator).* Every pre-activation of the HEAT network lies strictly in $\mathcal{D}$ by assumption, so applying $F_n$ instead of $f$ to it changes each output coordinate by at most $\varepsilon_n$, directly by (9).

*Step 2 (per-layer error of the quantizer).* Rounding the weights reaches the activations only indirectly:

**Lemma B.1.** *Let $w, q_w, h \in \mathbb{R}^d$ with $\|w - q_w\|_\infty \le s/2$ and $\|h\|_\infty \le B$. Then*

$$\big|f(\langle w, h\rangle) - f(\langle q_w, h\rangle)\big| \;\le\; \lambda \cdot \frac{s}{2}\, d\, B.$$

*Proof of Lemma B.1.* $|\langle w - q_w,\, h\rangle| \le \sum_{i=1}^d |w_i - q_{w,i}|\,|h_i| \le \frac{s}{2}\, dB$ by the triangle inequality; the $\lambda$-Lipschitz bound on $f$ gives the claim. □

Applied row-wise, the quantized layer differs from the exact layer, *evaluated on the same $B$-bounded input*, by at most $\lambda\frac{s(b)}{2}dB$ in every coordinate, which at $b = b^\star(n)$ is at most $\varepsilon_n$ by definition.

*Step 3 (accumulation across depth).* How the error accumulates across layer of the network can be modeled as

**Lemma B.2** (Trajectory bound)**.** *If $\delta_0 = 0$ and $\delta_{\ell+1} \le \rho\,\delta_\ell + \varepsilon_n$ for all $\ell < L$, then $\delta_\ell \le \varepsilon_n \sum_{j=0}^{\ell-1} \rho^j$ for all $\ell \le L$.*

*Proof.* Induction on $\ell$: the base case is the empty sum, and $\delta_{\ell+1} \le \rho\big(\varepsilon_n \sum_{j=0}^{\ell-1}\rho^j\big) + \varepsilon_n = \varepsilon_n \sum_{j=0}^{\ell}\rho^j$. □

Both trajectories satisfy this recurrence with $\rho = \lambda\kappa$. For the iterative network, insert the exact layer evaluated at $h^{\text{H}}_\ell$ and split by the triangle inequality:

$$\delta^{\text{H}}_{\ell+1} \;\le\; \underbrace{\big\|F_n(W_{\ell+1}h^{\text{H}}_\ell) - f(W_{\ell+1}h^{\text{H}}_\ell)\big\|_\infty}_{\le\,\varepsilon_n\ \text{(Step 1)}} \;+\; \underbrace{\big\|f(W_{\ell+1}h^{\text{H}}_\ell) - f(W_{\ell+1}h_\ell)\big\|_\infty}_{\le\,\lambda\kappa\,\delta^{\text{H}}_\ell\ \text{(Step 0)}}.$$

For QAT, insert the exact layer evaluated at $h^{\text{Q}}_\ell$, which is $B$-bounded by assumption:

$$\delta^{\text{Q}}_{\ell+1} \;\le\; \underbrace{\big\|f(Q(W_{\ell+1})h^{\text{Q}}_\ell) - f(W_{\ell+1}h^{\text{Q}}_\ell)\big\|_\infty}_{\le\,\varepsilon_n\ \text{(Step 2, matched budget)}} \;+\; \underbrace{\big\|f(W_{\ell+1}h^{\text{Q}}_\ell) - f(W_{\ell+1}h_\ell)\big\|_\infty}_{\le\,\lambda\kappa\,\delta^{\text{Q}}_\ell\ \text{(Step 0)}}.$$

Since $\delta^{\text{H}}_0 = \delta^{\text{Q}}_0 = 0$, Lemma B.2 yields $\delta^{\text{H}}_L,\, \delta^{\text{Q}}_L \le \varepsilon_n \sum_{j=0}^{L-1}(\lambda\kappa)^j$.

*Step 4 (from activations to losses).* The loss is $\xi_{\text{lip}}$-Lipschitz in the final activations, so, passing through the exact trajectory,

$$\begin{aligned}|\mathcal{L}_{\text{approx}} - \mathcal{L}_{\text{QAT}}| \;&\le\; \xi_{\text{lip}} \left\|h_L^{\text{H}} - h_L^{\text{Q}}\right\|_\infty \;=\; \xi_{\text{lip}} \left\|(h_L^{\text{H}} - h_L) + (h_L - h_L^{\text{Q}})\right\|_\infty \\ &\le\; \xi_{\text{lip}} \Big( \left\|h_L^{\text{H}} - h_L\right\|_\infty + \left\|h_L - h_L^{\text{Q}}\right\|_\infty \Big) \;=\; \xi_{\text{lip}} \big(\delta_L^{\text{H}} + \delta_L^{\text{Q}}\big) \\ &\le\; \xi_{\text{lip}} \Big( \varepsilon_n \sum_{j=0}^{L-1} (\lambda\kappa)^j + \varepsilon_n \sum_{j=0}^{L-1} (\lambda\kappa)^j \Big) \;=\; 2\,\xi_{\text{lip}}\, M r^{2^n} \sum_{j=0}^{L-1} (\lambda\kappa)^j,\end{aligned}$$

each trajectory spends one $\varepsilon_n$-budget against the exact network and the bound vanishes doubly exponentially in $n$. □

**Scope.** The bound compares task-loss *values* at fixed weights and a fixed count. It does not identify the two perturbations, their gradients, or the optimization trajectories they induce, and it does not cover $\mathcal{L}_{\text{iter}}$ and $\mathcal{L}_{\text{range}}$. Its purpose is to show that training a network under iterative approximations can be related to a well-known field, quantization-aware training, in which training under approximation error is established practice.

**Non-entrywise operators.** During the proof we used the fact that the layer is $\lambda\kappa$-Lipschitz (Step 0), and replacing the exact nonlinearity by its deployed approximant perturbs the layer output by at most $\varepsilon_n$ on the calibrated domain (Step 1).

These two hypotheses do not require the nonlinearity to act entry-wise, Theorem 1 therefore extends to the Softmax and LayerNorm blocks of `GPT-2`, with the block's own Lipschitz constant in place of $\lambda\kappa$ and its output error in place of $\varepsilon_n$ (the solver error scaled by the rest of the block, *e.g.* by $|\gamma_i(x_i - \mu)|$ in LayerNorm), constants we do not compute. These blocks have cross entry operations like the variance computation, or the Softmax denominator division, where the iterator sits inside the operator and its error propagates through the remainder of the block before reaching the next layer.

**Per-site counts and fidelity to the exact network.** Two extensions follow from the proof with no new argument. First, the iteration count enters only through the error $\varepsilon_n$ injected at each layer (Step 1), so if site $\ell$ runs $n_\ell$ iterations the recurrence of Lemma B.2 reads $\delta_\ell \le \rho\,\delta_{\ell-1} + \varepsilon_{n_\ell}$ and the same induction gives

$$\delta_L \;\le\; \sum_{\ell=1}^{L} \rho^{\,L-\ell}\, \varepsilon_{n_\ell},$$

which reduces to $\varepsilon_n \sum_{j<L} \rho^j$ when $n_\ell \equiv n$. Second, Step 3 already bounds the distance of each perturbed trajectory from the *exact* one, so Step 4 applied to $\delta_L^{\text{H}}$ alone yields the fidelity bound, and applied to both trajectories the per-site form of the theorem:

$$\left|\mathcal{L}_{\text{approx}}^{(n_1,\dots,n_L)} - \mathcal{L}_{\text{exact}}\right| \;\le\; \xi_{\text{lip}} \sum_{\ell=1}^{L} \rho^{\,L-\ell}\, M r^{2^{n_\ell}}, \tag{11}$$

$$\left|\mathcal{L}_{\text{approx}}^{(n_1,\dots,n_L)} - \mathcal{L}_{\text{QAT}}^{(b^\star(n_1),\dots,b^\star(n_L))}\right| \;\le\; 2\,\xi_{\text{lip}} \sum_{\ell=1}^{L} \rho^{\,L-\ell}\, M r^{2^{n_\ell}}. \tag{12}$$

Equation (11) bounds the *analytic-network* error, exact nonlinearities against $n_\ell$-step solvers, in plaintext and at fixed weights. It is not the quantity Section 4 reports. Table 1 and Figure 5 compare each encrypted run with a plaintext reference on the same weights, a gap that includes `CKKS` and bootstrapping noise, which no iteration count removes; top-1 agreement additionally requires a logit margin larger than twice the activation error. Equation (12) says that a network with site-specific iteration counts is within this bound of a mixed-precision quantized network with bit widths $b^\star(n_\ell)$, which is the sense in which learning the counts is a precision allocation.

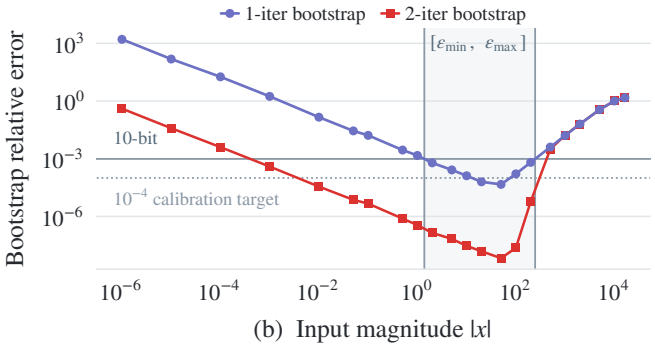


Figure 6: Bootstrap precision at different magnitude of the underlying plaintext with fixed correction factor 7, `FIDESlib`'s default.

Table 3: Each `CKKS` primitive timed in isolation on one custom `A100-64GB` at the deployment parameters of Appendix C.1 above, 100 repetitions each. *levels* is the multiplicative depth the operation consumes; the bootstrap row reads $-8$ as it returns a ciphertext from the refresh trigger at level 24 to level 16 (due to instability of deeper levels). Arithmetic is timed at that same input level 16.

| **primitive** | **levels** | **mean** (ms) | **min** (ms) |
|---|---|---|---|
| add (ct + ct) | 0 | 0.047 | 0.045 |
| sub (ct − ct) | 0 | 0.047 | 0.045 |
| mult (ct × ptx) | 1 | 0.066 | 0.042 |
| mult (ct × ct) | 1 | 0.580 | 0.559 |
| square ($ct^2$) | 1 | 0.566 | 0.544 |
| rotate (one key-switch) | 0 | 0.575 | 0.565 |
| **bootstrap** | **−8** | **82.1** | 81.9 |

# C Implementation Details

## C.1 Cryptographic settings

All runs use the `RNS` variant of `CKKS` (Cheon et al., 2018) at 128-bit security (enforced by `OpenFHE`). We instantiate a cyclotomic ring of dimension $N{=}2^{16}$ ($2^{15}$ slots) with a 60-bit first modulus and a uniform tower layout, every rescaling prime is 53 bits, pinning the scaling factor at $\Delta{=}2^{53}$ on every level of a bootstrappable modulus chain of 28 `RNS` limbs, for a total multiplicative depth of 27. A bootstrap returns ciphertexts at level 16, leaving 11 levels for the arithmetic between refreshes, however only 8 are usable in practice. Key switching is hybrid with $\mathrm{dnum}{=}7$ (four special primes), and we use a dense uniform ternary secret with sparse-secret encapsulation for bootstrapping (a transient Hamming-weight-32 key). Bootstrapping runs a single iteration with a $(4, 3)$ CoeffsToSlots/SlotsToCoeffs level budget; measured over uniform inputs of increasing magnitude, the refresh keeps the worst-case relative error below $2^{-10}$ for $|x| \in [1.38, 246]$ and below $10^{-4}$ ($\approx$13 bits) in the inner band $|x| \in [12.85, 76.4]$, 10–13 bits depending on the magnitude, from which we estimated the iteration calibration target.

**Primitive costs and precision.** Table 3 is the cost model behind the claim that multiplicative depth drives latency. On a single custom `A100-64GB` GPU, bootstrap dominates every other primitive by two to three orders of magnitude, so a model that tolerates a shallower circuit buys its latency almost entirely through the refreshes it avoids. We time each `CKKS` primitive in isolation, 100 repetitions each, at the deployment parameters above, on the machine used for every end-to-end run: one custom `NVIDIA A100-64GB`, an `Intel Xeon Platinum 8358` (8 cores and 128 GB of `RAM`), `CUDA` 12.6, `OpenFHE` 1.4.2, and a modified `FIDESlib` build. We also report the precision of the bootstrap operation over random sampled inputs against their magnitude Figure 6.

### C.2 MEMORY-EFFICIENT DEPTH GRADIENTS

The expected output $\widetilde{f}_l(x) = \sum_i p_i^l F_i^l(x)$ of Section 3 would, under reverse-mode autodiff, retain every intermediate solver state $F_i^l$ for the backward pass, so activation memory grows as $O(n_l)$ and the added memory traffic costs latency. Checkpointing removes the memory cost but pays for it with a recomputed forward pass *and* a backward sweep over it. Since no gradient is accumulated at any intermediate $F_i^l$, we can avoid the reverse sweep. We implement each iterative family (e.g. Algorithm 1) as a custom autograd function that stores only the solver's inputs and re-runs the recurrence during backward, propagating forward-mode tangents alongside the recomputed iterates. The recurrences are elementwise, so the Jacobian–vector sweep costs one extra forward pass rather than a forward and a backward pass, and the gradient through the truncated iterations is exact. Activation memory is thus constant in the iteration count. We run with mixed precision; we clamp the tangent accumulators, which off-domain rows can otherwise push past the `bf16` range.

Algorithm 1: Forward and backward pass of our Goldschmidt's $\widetilde{f}_l(x)$. Bars denote adjoints, $\bar{x} = \partial\mathcal{L}/\partial x$, with $N_0, D_0$ the inputs to BACKWARD. Tangents are $\nu = \partial N/\partial N_0$, $\eta = \partial N/\partial D_0$, $\delta = \partial D/\partial D_0$, $\varphi = \partial F/\partial D_0$.

**procedure** STEP($N, D, F$)
  $N \leftarrow NF; \quad D \leftarrow DF$
  **return** $N,\ D,\ 2 - D$

**procedure** FORWARD($N, D, F, \mathbf{p}$)
  $r \leftarrow N\,\mathbf{p}_0$
  **for** $i = 1, \ldots, |\mathbf{p}|$ **do**
    $N, D, F \leftarrow$ STEP($N, D, F$)
    $r \leftarrow r + N\,\mathbf{p}_i$
  **return** $r$

**procedure** TANGENTSTEP($N, D, F,\ \nu, \eta, \delta, \varphi$)
  $\nu \leftarrow \nu F; \quad \eta \leftarrow \eta F + N\varphi; \quad \delta \leftarrow \delta F + D\varphi$
  **return** $\nu,\ \eta,\ \delta,\ -\delta$

**procedure** BACKWARD($N, D, F, \mathbf{p}, g$) $\quad g = \partial\mathcal{L}/\partial r$
  $\nu, \eta, \delta, \varphi \leftarrow 1, 0, 1, 0$
  $\bar{N}_0,\ \bar{D}_0,\ \bar{\mathbf{p}}_0 \leftarrow g\,\mathbf{p}_0,\ 0,\ g\,N$
  **for** $i = 1, \ldots, |\mathbf{p}|$ **do**
    $\nu, \eta, \delta, \varphi \leftarrow$ TANGENTSTEP($N, D, F, \nu, \eta, \delta, \varphi$) $\quad$ pre-step $N, D, F$
    $N, D, F \leftarrow$ STEP($N, D, F$)
    $\bar{N}_0 \leftarrow \bar{N}_0 + g\,\mathbf{p}_i\nu; \quad \bar{D}_0 \leftarrow \bar{D}_0 + g\,\mathbf{p}_i\eta$
    $\bar{\mathbf{p}}_i \leftarrow g\,N$
  **return** $\bar{N}_0,\ \bar{D}_0,\ \bar{\mathbf{p}}$

### C.3 PARAMETERIZATION OF THE HALTING DISTRIBUTION

Section 3 treats each site's distribution $\mathbf{p}^s$ abstractly; in the implementation it is not stored as $n+1$ free probabilities but in the sequential form of Banino et al. (2021), which is what keeps it a distribution without an explicit normalization. Each site carries $n$ learnable *halting logits* $\ell_0^s, \ldots, \ell_{n-1}^s$; writing $h_i^s = \sigma(\ell_i^s) = \left(1 + e^{-\ell_i^s}\right)^{-1}$ for the probability of stopping at iteration $i$ *given* that iteration $i$ was reached,

$$p_i^s \;=\; h_i^s \prod_{j<i} \left(1 - h_j^s\right) \quad (i < n), \qquad p_n^s \;=\; \prod_{j<n} \left(1 - h_j^s\right), \tag{13}$$

*i.e.* the deepest state absorbs the remaining mass ($h_n^s \equiv 1$), so $\mathbf{p}^s$ is a distribution over $\{0, \ldots, n\}$ by construction while the $\ell_i^s$ stay unconstrained. We evaluate Equation (13) in log space, $\log p_i^s = \log \sigma(\ell_i^s) + \sum_{j<i} \log \sigma(-\ell_j^s)$, and read the last atom off as the complement $p_n^s = \left(1 - \sum_{i<n} p_i^s\right)_+$. This is the product in Equation (13) exactly, and the clamp only absorbs floating-point round-off.

Nothing in Section 3 depends on this choice. The expected approximation, $\mathcal{L}_{\text{iter}}$ and the exported mode $i^{s,\star}$ are functions of $\mathbf{p}^s$ alone, and gradients reach the $\ell_i^s$ only through it. At a site with a floor $m^s > 0$ (Softmax initialization and refinement, LayerNorm-Goldschmidt $m^s{=}1$; LayerNorm-Newton $m^s{=}0$) the logits $\ell_0^s, \ldots, \ell_{m^s-1}^s$ are dropped from the recursion altogether, so $\mathbf{p}^s$ is supported on $\{m^s, \ldots, n\}$: states below the floor carry exactly zero mass, receive no gradient, and the exported index is offset by $m^s$.

### C.4 Initialization and schedule

Each halting distribution spans the solver's states from a per-family floor $m^s$ up to a per-family maximum count. Each distribution is initialized by a monotone ramp on the halting logits of Equation (13), $\ell_i^s = \gamma\,(i - (n-1))$ with $\gamma{=}0.7$. $h_i^s$ is thus near zero at shallow $i$ and reaches $\frac{1}{2}$ one step below the maximum. The resulting $\mathbf{p}^s$ is monotone non-decreasing, its two deepest states carrying equal mass, so training starts at the top of each site's support and the descent is free to settle above the calibrated count where the task demands it, as LayerNorm-Newton does at blocks 0–1. A flat law spiked on the deepest state, which we tried first, leaves the shallow states without gradient and slow to reduce. During training, the inputs of every approximated operator are kept inside their calibrated domains by hard clamps carrying a quadratic push-back gradient (clamp in the forward pass, penalty gradient in the backward); the clamps are a training-time guard only, and are absent during encrypted inference (as no comparison primitive exists). At export, the deployed count of every site is the mode of its halting distribution. The prior of site $s$ is a negative-binomial law on the active support $k = i - m^s$, $p_k^{s,\star} \propto \binom{k+r-1}{k} p^r (1-p)^k$, renormalized on the finite support, with shape $r = 1 + (w^s + \frac{1}{2})\,p/(1-p)$ so that its mode sits at the target $w^s$ (a geometric law when $w^s{=}0$).

### C.5 Training configuration (GPT-2 small)

Table 4 is the recipe behind every HEAT row of Tables 1 and 2. The three phases of Section 3.1 run back to back on one GPU at context 1024 on OpenWebText, with AdamW (Loshchilov & Hutter, 2019) ($\beta{=}(0.9, 0.99)$, $\epsilon{=}10^{-10}$, weight decay 0.1, gradient norm clipped at 1). Phase one trains the weights alone under $\mathcal{L}_{\text{task}} + \lambda_{\text{range}}\mathcal{L}_{\text{range}}$, with the pretrained model as a self-distillation teacher ($\mathcal{L}_{\text{task}}$ is the KL divergence to the teacher at temperature 1, with no cross-entropy term) and the Softmax evaluated exactly under its score squeeze. Phase two starts from that point, calibrated once at $\varepsilon{=}10^{-4}$ to seed every site, releases the halting distributions and ramps $\mathcal{L}_{\text{iter}}$ in over the next 500; at the switch the weights' learning rate steps to its phase-two value and decays by cosine to the floor over the remaining budget. Phase three freezes every count at its learned mode and continues the weights for 1000 updates at the floor with $\lambda_{\text{iter}}{=}0$. The halting logits carry their own learning rates. The guards of $\mathcal{L}_{\text{range}}$ bound the GELU input at 32, the attention scores at the squeeze bound, the Softmax denominator at 7, the LayerNorm variances at 16, and keep the inverse-square-root input inside its calibrated domain. The halting initialization and the clamps are those of Appendix C.4; validation is the mean over 32 held-out batches every 250 updates.

## D The Approximated Operators

Despite the limited operation set offered by CKKS, there is a well-studied variety of algorithms that, using only additions, multiplications, and rotations, allow nonlinear functions to be computed. Which algorithms to choose, how to compose them, and how to initialize their recursions determine part of the encrypted inference system. A common transformer, and especially GPT-2, includes three main nonlinear chokepoints that must be converted to the available operations to allow computation under CKKS: the Softmax, the LayerNorm, and the GELU activation. The Softmax and GELU circuits follow THOR (Moon et al., 2025), and the inverse-square-root circuit follows Qu & Xu (2023); we summarize them here so the paper is self-contained, and point out where the iteration counts that HEAT learns sit inside each circuit. The Goldschmidt reciprocal of $d > 0$ starts from a seed $y_0 = \alpha - \beta d$, where $(\alpha, \beta)$ is the minimax linear initializer for the calibrated interval $[d_{\min}, d_{\max}]$, and iterates $y_{i+1} = y_i\,(2 - d\,y_i)$; writing $d\,y_i = 1 - e_i$ gives $e_{i+1} = e_i^2$, so each iteration squares the error. The Newton inverse square root of $z$ iterates $y_{k+1} = \frac{1}{2}\,y_k\,(3 - z\,y_k^2)$ and converges quadratically likewise. Every learned halting site of Section 3 wraps one of these two loops.

Table 4: HEAT fine-tuning recipe for GPT-2 small, one column per phase of Section 3.1. $\lambda_{\text{iter}}$ ramps in over the first 500 updates of phase two. The prior lowers its mode once the learned mode has saturated on it for 75 micro-batch steps, at least 150 updates apart.

| | **phase 1** squeeze | **phase 2** HEAT | **phase 3** cool-down |
|---|---|---|---|
| updates | 500 | 3500 | 1000 |
| tokens per update | 32768 | 4096 | 4096 |
| weights lr | $10^{-4}$ | $2.5\times10^{-4}$ | $3\times10^{-5}$ |
| halting-logit lr | – | $2\times10^{-3}$ | 0 |
| $\lambda_{\text{range}}$ | $3\times10^{-2}$ | 0.3 | 0.3 |
| $\lambda_{\text{iter}}$ | – | 0.6 | 0 |
| score squeeze bound | 10 | 14 | 14 |
| halting | – | soft (Equation (4)) | hard |
| prior $p$, start / end | – | 0.6 / 0.9 | – |
| back-off patience / gap | – | 75 / 150 | – |
| self-distillation | yes | – | – |

**Softmax.** The circuit is the two-phase exp-and-reciprocal construction of THOR. Let a score row lie in the calibrated range $[a, b]$ of width $M = b - a$. The row is first shifted by the plaintext constant $(a + b)/2$: under oblivious execution no encrypted row-max is available, so the classical max-subtraction is replaced by a calibration-time midpoint. Two scaling factors $(\delta_1, \delta_2)$ govern the rest. The shifted row is scaled by $1/(\delta_1\delta_2)$, placing its entries in a window of width $M/(\delta_1\delta_2)$ narrow enough for a degree-8 Chebyshev fit of $e^x$; the fit is then raised to the $\delta_1$-th power by $k = \log_2 \delta_1$ ciphertext squarings. Phase one closes by normalizing, the row is summed and multiplied by the Goldschmidt reciprocal of the sum, the Softmax-initialization site. By construction the result approximates $\mathrm{Softmax}(\mathbf{x}/\delta_2)$, and since $\mathrm{Softmax}(2\mathbf{x})_i = y_i^2/\sum_j y_j^2$ for $\mathbf{y} = \mathrm{Softmax}(\mathbf{x})$, phase two applies $\log_2 \delta_2$ square-and-normalize passes, each one squaring plus one Goldschmidt reciprocal of the new sum, the Softmax-refinement site, to recover $\mathrm{Softmax}(\mathbf{x})$. Moon et al. (2025) select $(\delta_1, \delta_2)$ by greedy search on a training set; our calibration performs the per-block equivalent, and this refinement loop is where the learned depth stays structured across layers (Figure 7 and Figure 3a).

**LayerNorm inverse square root.** The variance $z$ enters a calibrated domain $[z_{\min}, z_{\max}]$, and the cost of evaluating $1/\sqrt{z}$ under CKKS is dominated by how the Newton loop is seeded. Qu & Xu (2023) proposes two options, a Taylor's polynomial and rational initializers. We instantiate the rational variant. We fit a degree-$(3, 1)$ Remez rational approximation of $1/\sqrt{z}$ on the calibration domain, its division is realized by multiplying the numerator with the Goldschmidt reciprocal of the denominator, the LayerNorm-Goldschmidt site, and the resulting quotient seeds the Newton loop, the LayerNorm-Newton site. Both counts are learned; the seeds $(\alpha, \beta)$ and the rational coefficients are calibration state, held fixed during training and re-fitted once on the final weights.

**GELU.** Following THOR (Moon et al., 2025), and unlike previous comparison-based methods (de Castro et al., 2025; Dong, Ye et al., 2025), the activation is a composition of two low-degree polynomials rather than a single high-degree fit, or a piecewise one. $\mathrm{GELU}(x) \approx x\left(P_2(P_1(x/S)) + \frac{1}{2}\right)$ with $\deg P_1 = 31$, $\deg P_2 = 27$, and $S$ a calibrated input bound, evaluated with the Paterson–Stockmeyer algorithm at deployment. The circuit is a fixed-depth polynomial with no iterative loop, which is why GELU carries no halting distribution and enters the training machinery only through its range guard. During HEAT we evaluate the analytic GELU and fit the polynomials directly on the fine-tuned weights.

**Encrypted** $\arg\max$**.** Decoding ends with an encrypted $\arg\max$ over the vocabulary logits, following the CutMax construction of Avitan et al. (2025). The row is driven to a one-hot indicator of its maximum by a short sequence of *sharpening* rounds. Each round re-centers and rescales the row, raises it to a small power, widening the leader's margin over the rest, and renormalizes; the normalizer is itself evaluated iteratively, by a chord-seeded Newton inverse-square-root cascade of the same kind as LayerNorm's. The deployed schedule runs five rounds (powers $9, 3, 3, 3, 7$, with

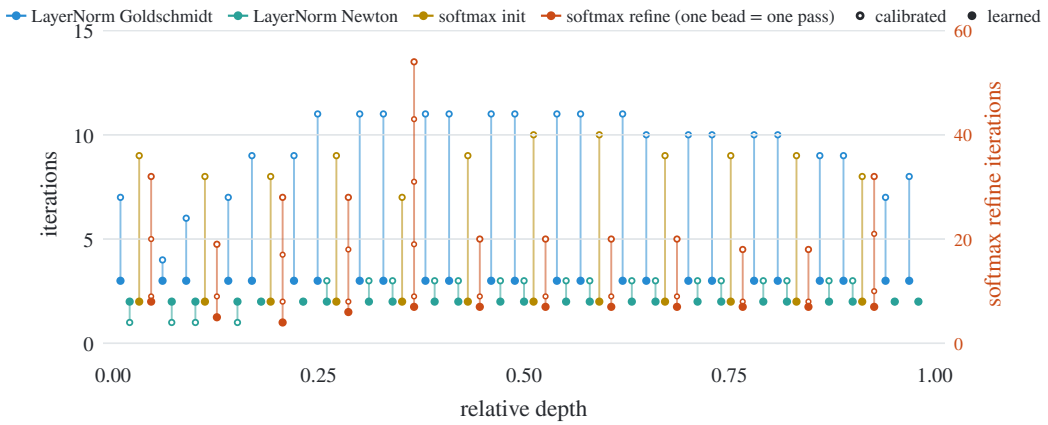


Figure 7: Per-site iteration counts, `GPT-2`: calibrated baseline (open) vs. `HEAT`-learned deployment (filled), one column per halting site in depth order. Family totals move $235/63/105/309 \rightarrow 75/50/24/79$ (LayerNorm-Goldschmidt/Newton, Softmax initialization/refinement), *i.e.* $712 \rightarrow 228$ iterations per forward.

one to three Newton passes each) and is derived at calibration time. Unlike the four halting families above, the $\arg\max$ carries no halting site. Its schedule is calibration state, frozen once and shared byte-identical by every approach we compare, so it contributes the same $\approx 9.3\,\text{s}/\text{tok}$ to every token and none of the reported latency differences. Future work might consider including the $\arg\max$ in the optimization, despite being unstable.

## E Additional Results

`GPT-2` exposes 74 halting sites, one per solver stage of Appendix D. LayerNorm-Goldschmidt (25), LayerNorm-Newton (25), Softmax initialization (12), and Softmax refinement (12, at $\delta_2{=}2$ for every layer after squeeze); the GELU is a fixed polynomial and carries no halting site. Figure 7 compares every site's calibrated count versus what `HEAT` learns. The open marker is the calibrated baseline count, the filled marker the learned deployed count. First, we note the LayerNorm families collapse to shallow, uniform counts. Goldschmidt moves from per-site 4–11 to a flat 3, Newton from 1–3 to 2 everywhere, an *increase* at blocks 0–1, where the calibrator had settled on 1. Second, Softmax refinement is the one family that allocates precision in a *structured* way: block 0 retains 8 refinement iterations while the later blocks settle at 4–7. Figure 3a traces how that structure emerges during training. The layers separate early in the descent and each settles at the depth its approximation demands, rather than following a global schedule.

### E.1 Sensitivity to the seed and to the iteration penalty

We rerun the three phases of Table 4 with three further seeds and, at one seed, over $\lambda_{\text{iter}} \in \{0.3, 0.6, 1.2\}$ crossed with the prior's end probability $p_{\text{end}} \in \{0.8, 0.9, 0.95\}$; everything else is unchanged. These runs are in plaintext, and their perplexity is the validation loss at context 1024 over 160 held-out OpenWebText batches shared by every run, not the 128-token windows of Table 1. Every seed recovers the shipped circuit exactly, the same count at every site and hence 228 iterations per forward, while the weights differ (perplexity $22.96 \pm 0.07$ over the held out samples). Over training, the four runs of the recipe, the shipped run included, step down at slightly different times but coincide from update 3325 on, and their validation losses differ by no more than the evaluation noise (Figure 8b). Raising $\lambda_{\text{iter}}$ removes iterations and costs perplexity (Figure 8a). The end probability moves neither quantity by more than 3 iterations or 0.2 perplexity, with one exception, at $\lambda_{\text{iter}}{=}0.3$, $p_{\text{end}}{=}0.8$ barely compresses (362 iterations, against 240 at $p_{\text{end}}{=}0.9$).

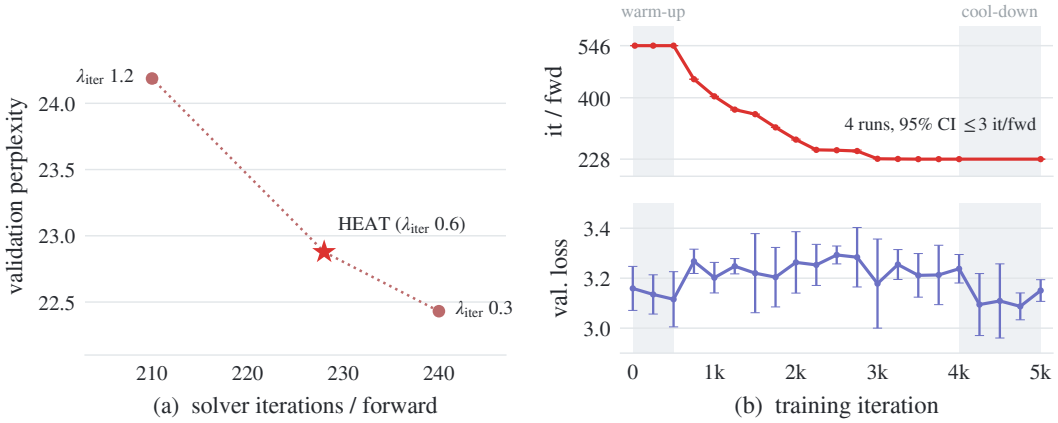


Figure 8: (a) Raising $\lambda_{\text{iter}}$ trades solver iterations for validation perplexity (one seed, $p_{\text{end}}$=0.9, everything else as in Table 4); the star is the shipped setting. (b) Four runs of the shipped recipe, the shipped run and three further seeds, as mean and 95% confidence interval ($n$=4). The iteration count follows the same descent starting from the sum of the modes of the initialized distributions; the median interval on the validation loss (±0.087) equals the one at update 0 (±0.088), where the runs share their weights and differ only in their 32 evaluation batches. Shaded: phase one warm-up and phase three cool-down.

### E.2 Image classification task

Table 5 repeats the protocol of Table 1 on encrypted `EuroSAT` classification with a `ViT-B/16` (86M parameters). 1024 test images drawn at random, images whose logits do not decrypt counted as misses. `HEAT` is the fastest circuit, with 2.5× fewer solver iterations and 1.8× fewer bootstraps than the calibrated `ViT`, 99.4 against 122.5 s per image, and 1.10× faster than `ATLAS`. The gain is smaller than on `GPT-2` decode because the 80 tokens (patches) of an image are processed at once, which moves a larger share of the per-image time into the linear families (Figure 9); `HEAT` acts only on the iterative sites, so less of the wall time is within its reach. Classification is also a single forward pass. Even in this setting, `HEAT` yields the most stable circuit, with the fewest images failing during decryption.

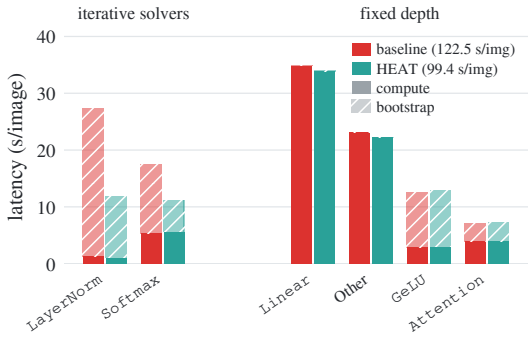


Figure 9: Per-family decode-latency composition, calibrated `ViT-B16` vs. `HEAT`: paired bars in absolute seconds per image, the hatched cap of each bar being the bootstrap time spent inside that family. `HEAT`'s saving concentrates the iterative methods' bars, where in this scenario the main source of latency is actual linears' computation and memory management.

Table 5: Encrypted EuroSAT classification at deployment (custom A100-64GB). 1024 validation images drawn uniformly at random from the test set. it/fwd and bts/img: iterations of the iterative approximations and bootstraps per encrypted forward; s / image: wall time of the 12 encoder blocks (Transf.) and end to end (e2e), the latter including the ≈22 s head, median over the 1024 images (± one s.d.); fail.: images whose logits do not decrypt, counted as misses; top-1 under encryption over all 1024 images, with the same circuit evaluated in plaintext on the full split as subscript.

| | circuit | | s / image | | 1024 images | |
|---|---|---|---|---|---|---|
| **System** | it/fwd(↓) | bts/img(↓) | Transf.(↓) | e2e(↓) | fail.(↓) | top-1 $\texttt{FHE}_{\text{(plain)}}$(↑) |
| ViT (plaintext) | – | – | – | – | – | 98.2 |
| ViT | 617 | 575 | $99.4_{(\pm 0.4)}$ | $122.5_{(\pm 0.6)}$ | 44/1024 | $94.4_{(96.0)}$ |
| $\mathcal{L}_{\text{range}}$ only | 529 | 464 | $88.3_{(\pm 0.3)}$ | $110.8_{(\pm 0.6)}$ | 21/1024 | $91.2_{(93.0)}$ |
| ATLAS (Xie et al., 2026) | 358 | 425 | $87.0_{(\pm 0.5)}$ | $109.7_{(\pm 1.4)}$ | 14/1024 | $96.7_{(96.7)}$ |
| **HEAT (ours)** | **250** | **326** | $\mathbf{77.2}_{(\pm 0.3)}$ | $\mathbf{99.4}_{(\pm 0.6)}$ | **3/1024** | $\mathbf{97.9}_{(98.1)}$ |

Table 6: Zero-shot GLUE accuracy for GPT-2 small. The pretrained model, and the two deployed circuits evaluated in plaintext. GPT-2 small sits at the majority-class floor on zero-shot GLUE, so the axis is whether HEAT loses anything. Its circuit matches or exceeds the calibrated one on $5/8$ tasks and on average.

| | GPT-2 | **deployed circuit** | |
|---|---|---|---|
| **Task** (↑) | (plaintext) | GPT-2 | **HEAT** |
| SST-2 | 55.0 | 55.0 | **56.1** |
| MRPC | 56.1 | 56.1 | **67.4** |
| QQP | 37.3 | **37.3** | 36.9 |
| MNLI-m | 33.7 | 33.7 | **35.2** |
| MNLI-mm | 33.2 | 33.2 | **34.8** |
| QNLI | 50.2 | **50.2** | 49.5 |
| RTE | 53.1 | **53.1** | 51.6 |
| WNLI | 42.2 | 42.2 | **43.7** |
| **Average** | 45.1 | 45.1 | **46.9** |

### E.3 Downstream capability

HEAT changes the weights, and every perplexity in Tables 1 and 2 is measured on OpenWebText, the fine-tuning corpus. Tables 6 and 7 therefore evaluate the deployed circuits in plaintext on standard zero-shot benchmarks. The calibrated circuit is within 0.1 of the pretrained model on every metric. HEAT is within 1 point on HellaSwag and PIQA and 1.8 perplexity on C4.

### E.4 ATLAS replication

The ATLAS row of Table 1 comes from its two-stage NSGA-II search, run with the released driver, operators and budget. We adapt the encoding to our circuit. The decision vector is the iteration counts, *i.e.* per LayerNorm, Goldschmidt $\in [1, 13]$ and Newton $\in [0, 3]$; per Softmax, the Goldschmidt initialization $\in [2, 9]$ and the refinement count $\in [1, 12]$. Everything else stays at its calibrated value, *e.g.* the fitted constants, the number of refinement passes, the exponential and GELU polynomials, and CutMax. Changing any of these requires re-fitting the constants, which is a new calibration rather than a search. ATLAS's activation-degree axis has no counterpart because GELU places no bootstrap in any of our plans. The objectives are the total approximation depth, with levels per iteration measured on our circuit, and the mean absolute error of the last block's hidden states against the exact pretrained GPT-2, on $4 \times 512$ OpenWebText tokens in place of ATLAS's 10 sentences. We set the feasibility bound to $\varepsilon$=1.0, that admits every circuit within 2% of the calibrated perplexity, and, as in ATLAS, the task re-evaluation makes the final choice. Stage one searches a single configuration shared by all layers as in the released code, with population 48 for 50 generations. Stage two, seeded with stage one's Pareto set, searches per layer with population 96 for 225 generations. The total is 24,144 evaluations, that take ≈46 minutes on one GPU. We re-evaluate all

Table 7: Zero-shot downstream capability of GPT-2 small. The pretrained model, and the deployed circuits evaluated in plaintext at their deployed iteration counts.

| Benchmark | Metric | GPT-2 (plaintext) | deployed circuit GPT-2 | ATLAS | **HEAT** | $\mathcal{L}_{\text{range}}$ only |
|---|---|---|---|---|---|---|
| C4 | PPL (↓) | 27.9 | **28.0** | 28.6 | 29.8 | 28.7 |
| HellaSwag | acc (↑) | 28.9 | **28.9** | **28.9** | 28.8 | 28.7 |
| | $\text{acc}_{\text{norm}}$ (↑) | 31.1 | **31.1** | 31.0 | 30.5 | 31.0 |
| PIQA | acc (↑) | 62.9 | **62.9** | 62.7 | 62.0 | 62.8 |
| | $\text{acc}_{\text{norm}}$ (↑) | 62.5 | 62.5 | **62.6** | 62.3 | **62.6** |

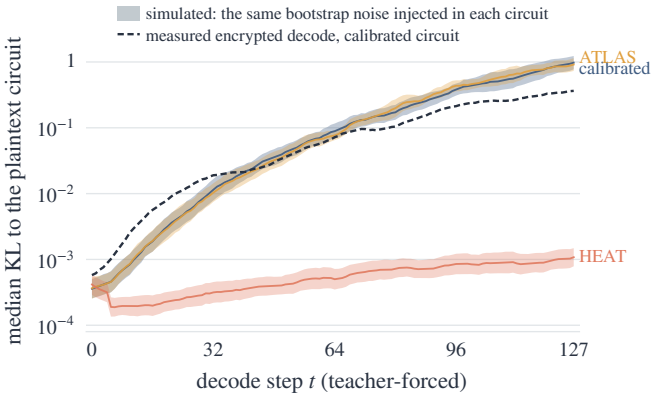


Figure 10: Median KL to the plaintext circuit when the same measured bootstrap noise is injected into the calibrated circuit, into ATLAS and into HEAT, over 64 held-out OpenWebText windows of 128 positions, disjoint from the data the ATLAS search used. Dashed: the measured encrypted decode of the calibrated circuit.

96 points returned by stage two by plaintext perplexity on $8 \times 1024$ OpenWebText tokens, and keep the cheapest within 2% of the calibrated circuit. Over three seeds, the cheapest such points sit at 438, 468 and 522 iterations per forward; Table 1 reports the cheapest. It cuts approximation depth from 838 to 454 ($-46\%$, against the $\approx 35\%$ ATLAS reports on its own baseline) and iterations from 712 to 438.

### E.5 Where the encrypted drift comes from

A bootstrap adds error to every slot, whatever the slot holds. We inject into plaintext models the error measured at the scheduled bootstraps of Table 1. The calibrated circuit loses 34% perplexity with its KL rising $0.001 \rightarrow 0.45$, while the HEAT one loses nothing under the same noise (Figure 10). Injecting the noise selectively to one approximation at the time identifies the Softmax as the main noise amplification and thus source of drift. The source of this is the refinement of Appendix D. Each square-and-normalize pass amplifies the relative error of its rows. The calibrated circuit's two to five passes per block push it well above 50% by the end of the decode, whereas HEAT, whose halved score ranges and reduced iterations lets the calibrator use a single shallow pass, stays below 10% (Figure 11). The ATLAS circuit of Table 1 inherits the calibrated refinement unchanged and, under the same injected noise, drifts like the calibrated circuit (Figure 10). The injected noise gives between 2 and 43 collapsed windows of 64 depending on the assumed noise floor, thus including the 13 registered under actual FHE inference. The measured error is zero-mean with $\sigma = 1.7 \times 10^{-4}$ per slot, independent of the plaintext amplitude up to $\approx 30$; we resample it at every scheduled bootstrap, on 64 held-out windows with three seeds, and fit no parameter to the encrypted runs. A search scored under such a noise model might close part of this gap; however it would be a new method rather than a baseline.

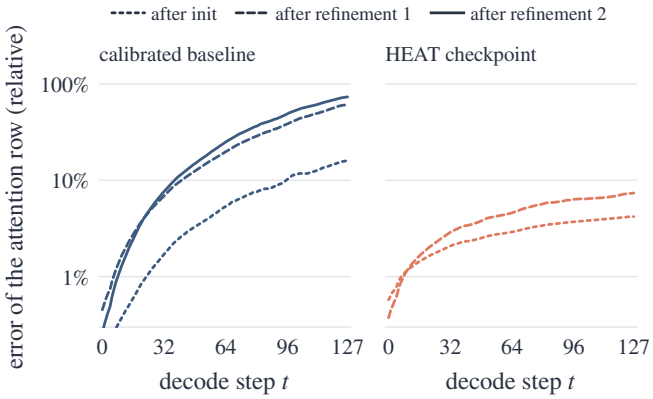


Figure 11: Relative error of the attention rows after each stage of the Softmax chain, noisy vs. noise-free. Left: calibrated circuit, two refinement passes. Right: HEAT, one.

Table 8: The calibrated circuit at four approximation tolerances, protocol and columns of Table 1. Loosening the tolerance buys iterations and time but not stability or fidelity. The ladder is not monotone (the $10^{-3}$ circuit collapses least and drifts most, the $10^{-1}$ circuit the reverse), every step keeps at least 2 collapsed sequences and a 25-66% perplexity loss against its own plaintext circuit, and none reaches HEAT on any column.

| | **circuit** (↓) | | **s/token** (↓) | | **fidelity** | | **perplexity** (↓) |
|---|---|---|---|---|---|---|---|
| **Calibration tolerance** ($\varepsilon$) | it/fwd | bts/tok | Transf. | **e2e** | coll. (↓) | top-1 (↑) | $\texttt{FHE}_{\text{(plain)}}$ |
| $10^{-1}$ | 571 | 418 | $36.6_{(\pm 0.3)}$ | $45.9_{(\pm 0.4)}$ | 16/64 | $60.9_{(\pm 3.4)}$ | $41.5_{(33.1)}$ |
| $10^{-2}$ | 642 | 456 | $40.1_{(\pm 0.3)}$ | $49.4_{(\pm 0.4)}$ | 14/64 | $60.1_{(\pm 2.9)}$ | $54.1_{(32.6)}$ |
| $10^{-3}$ | 678 | 474 | $41.5_{(\pm 0.3)}$ | $50.8_{(\pm 0.4)}$ | 2/64 | $70.5_{(\pm 1.7)}$ | $52.7_{(31.7)}$ |
| $10^{-4}$ (Table 1) | 712 | 513 | $45.0_{(\pm 0.3)}$ | $54.3_{(\pm 0.4)}$ | 7/64 | $70.3_{(\pm 2.6)}$ | $44.6_{(31.7)}$ |
| **HEAT** | **228** | **326** | $\mathbf{28.2}_{(\pm 0.2)}$ | $\mathbf{37.6}_{(\pm 0.3)}$ | **0/64** | $\mathbf{82.9}_{(\pm 0.5)}$ | $\mathbf{30.7}_{(31.0)}$ |

**Calibration tolerance.** Bootstrap can restore multiplicative depth while maintaining roughly 13 bits of precision for the underlying payload, and thus a reasonable calibration choice would be $\varepsilon{=}10^{-4}$ as the target. This ensures the encrypted circuit operates at the maximum precision possible. Table 8 shows what happens to the same pretrained models where the target is moved towards lower precision. None of the columns is alone enough to decide which is better as multiple factors must be taken into careful consideration. The plaintext perplexity of the circuit is roughly monotone with the precision, as one would expect; the encrypted circuit however has to be weighed between two factors: the collapsed rate and the top-1 accuracy. The first leaves only two candidates, namely $\{10^{-3}, 10^{-4}\}$, however the FHE perplexity of the $\varepsilon{=}10^{-3}$ option is unstable and far from its plaintext twin leaving only $10^{-4}$, *i.e.* FIDESlib bts precision on the table.

### E.6 Scaling to larger models

HEAT optimizes the circuit and it decides how many Goldschmidt, Newton and Softmax refinement steps every site needs, and that count is what we measure. Whether such a circuit is *affordable* under encryption at a larger size is a different question, dominated by costs that HEAT cannot optimize, like key material, the weight plaintexts and their movement, and the packing tier, all of which grow with the width of the model (the 1280-wide residual stream of GPT-2-large, for instance, has to be padded to a 2048-slot layout). These are implementation problems on an axis orthogonal to the scope of this paper, which we don't explore. What can be verified without encryption is whether the iteration cut survives scale, because the plaintext simulation *is* the deployed circuit and on GPT-2 the two agree (Section 4). Table 9 applies the recipe unchanged, apart from a smaller phase-2 learning rate on the largest model, to the GPT-2 family up to 774M parameters, with perplexity measured on the same OpenWebText validation slice at full context (hence not comparable with the first-128-token numbers of the main matter). The cut is nearly flat with $3.1\times$, $3.0\times$ and $2.8\times$ fewer iterations

Table 9: Applying the HEAT fine-tuning paradigm to the whole GPT-2 family up to 774M parameters confirming an almost flat 3× iterations gain while retaining performance. (Bigger models aren't tested in the FHE inference but just the plaintext scenario).

| | | calibrated | | squeezed | | **HEAT** | | |
|---|---|---|---|---|---|---|---|---|
| **Model** | **params** | it/fwd (↓) | ppl (↓) | it/fwd (↓) | ppl (↓) | it/fwd (↓) | ppl (↓) | **reduction** |
| GPT-2 | 124 M | 712 | 23.72 | 455 | 24.19 | **228** | **23.19** | **3.12**× |
| GPT-2-medium | 355 M | 1319 | 16.76 | 800 | 17.05 | **441** | **16.94** | **2.99**× |
| GPT-2-large | 774 M | 1897 | 15.17 | 1278 | 15.86 | **672** | **15.87** | **2.82**× |

per forward pass than the calibrated circuit, at a perplexity between $-2.2\%$ and $+4.6\%$ of the calibrated baseline and at half the cost of the range-regularized (squeezed) circuit. Larger models keep the lower perplexity floor their size affords, so the saving is not bought with the model's capacity.

### E.7 Free-running encrypted generation

Every fidelity number in the discussion is teacher-forced, *i.e.* each position receives the ground-truth token, so an encrypted error cannot propagate to the next step. We therefore also run the deployed HEAT circuit the way a server would, autoregressively and fully under encryption. The prompt is encrypted and processed token by token through the decode path of Table 1. From then on, each token is selected by the encrypted $\arg\max$ of Appendix D, whose one-hot output is mapped homomorphically to its token and positional embedding and fed back, after a bootstrap, as the next input. Figure 12 compares 32 encrypted tokens per prompt with the greedy decoding of the same circuit (the HEAT weights at the learned iteration counts) evaluated in plaintext. On three of the four prompts the two are identical token for token. On the fourth they branch at the third token, where the plaintext circuit ranks "the" over "a" by $0.005$ in logit and encryption swaps them, and both continuations remain fluent. Scored step by step against the plaintext circuit run on the encrypted prefix, the encrypted $\arg\max$ agrees at $126$ of $128$ steps, both misses being near-ties of this kind, with a median KL of $5 \cdot 10^{-4}$ to $8 \cdot 10^{-3}$ per prompt. For reference, the greedy continuation of the pretrained GPT-2 departs from HEAT's within the first seven tokens of every prompt, at the same token whether HEAT runs encrypted or in plaintext meaning that the text changes with the fine-tuning, not with the encryption. Greedy decoding of a model of this size falls into repetition within a sentence or two, and the classic remedies, repetition penalties and sampling, are not yet available for encrypted free-running generation, whose every token is the encrypted $\arg\max$. We therefore choose not to display the repetitions and cut each continuation where it starts repeating, while every statistic above is computed over all $32$ tokens. The length of decoding is fixed in advance as, by default, encrypted generation runs for a pre-set number of steps, since the server never sees the generated tokens and cannot stop on an end-of-sequence token.

## F Societal Impacts

FHE brings an additional dimension to the societal implications of deploying modern deep learning systems. While privacy-preserving inference can provide substantial benefits by reducing the exposure of sensitive user data, the privacy guarantees offered by FHE may also change how such systems can be deployed and used in practice. Deep learning models already have broad societal benefits, while at the same time lowering the barriers to a range of potentially harmful applications. Consequently, the implications of combining increasingly capable models with strong cryptographic privacy guarantees warrant careful consideration.

In particular, there remains a gap between the privacy properties demonstrated by FHE protocols in controlled settings and those of complete systems deployed in practice. Practical deployments involve additional components, assumptions, and operational choices that may affect the resulting privacy and security properties. Moreover, naively incorporating FHE into an existing inference pipeline does not necessarily provide the same guarantees as a carefully designed end-to-end private system. We therefore believe that the broader societal implications of deploying FHE-based inference systems, including both their potential benefits and unintended consequences, deserve further investigation.

**The capital of France is**

HEAT, encrypted = plaintext
the capital of the French Republic. The French Republic is the largest state in the world. [. . . ]

pretrained GPT−2
the capital of the French Republic, and the capital of the French Republic is the capital of the French Republic. [. . . ]

agree 32/32 · KL 0.0023

**Recently, scientists have discovered that**

HEAT, encrypted = plaintext
the brain is a complex system of neurons, which are connected to the brain through the spinal cord.↩ ↩ The researchers found that [. . . ]

pretrained GPT−2
the brain's ability to process information is impaired when it is exposed to a high-energy source of energy.↩ ↩ The findings, published in the journal Nature

agree 32/32 · KL 0.0014

**The best way to learn a new language is**

HEAT, encrypted = plaintext
to learn it from a teacher. [. . . ]

pretrained GPT−2
to learn it from a native speaker. [. . . ]

agree 32/32 · KL 0.0005

**It should also be noted that early on in this case, Hotfile**

HEAT, encrypted
was not a legitimate source for the files. [. . . ]

HEAT, plaintext
was not the only site that was targeted by the FBI. The FBI also targeted the site of the same name, [. . . ]

pretrained GPT−2
was not a file server. It was a file server that was used to store files on the server. [. . . ]

branches from its plaintext at a near-tie · agree 30/32 · KL 0.0076

Figure 12: Free-running generation of the deployed HEAT circuit under FHE, 32 greedy tokens per prompt. The prompt (black) is encrypted, and every token HEAT generates (blue) is chosen by the encrypted $\arg\max$ and fed back without decryption. In gray it's the plaintext greedy continuations, of the pretrained GPT−2 for reference.