CONTRACT-ALIGNED QUANTIZATION FOR FSS-BASED SECURE TRANSFORMER INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Function secret sharing (FSS) enables low-cost online two-party inference, but current quantized Transformers for private inference are trained with only bitwidth-level protocol awareness—i.e., an idealized forward pass that differs from what the deployed gates actually compute. In practice, FSS gates introduce share-wise rounding with correlated truncation error, public lookup-table (LUT) encodings, and gate-dependent input domains. We demonstrate that this train–deploy mismatch consistently reduces accuracy. To address it, we introduce contract-aligned quantization: each nonlinear operator first releases an explicit *operator contract* specifying the ring, fixed-point scale, input domain, LUT payload encoding, and truncation path, and the pruned, quantized model is then retrained using the exact integer semantics of the deployed gates in the forward pass. On SST-2, we preserve 98.5% of the teacher model's accuracy. Relative to SIGMA on a per-operator basis, Softmax attains about 77% lower latency and about 50% less communication, while GELU attains about 85% lower latency and about 36% less communication.

1 INTRODUCTION

Pre-trained Transformers power online services such as text classification, retrieval, question answering, and language generation Vaswani et al. (2017); Devlin et al. (2019); Radford et al. (2019). Their deployment creates a two-sided privacy challenge: client queries may contain sensitive information, while model weights are valuable proprietary assets. Privacy-preserving machine learning (PPML) enables private inference by revealing only the agreed prediction.

A dominant line of PPML realizes private inference with secure multiparty computation (MPC), alone or combined with homomorphic encryption or garbled circuits Mohassel & Zhang (2017); Mohassel & Rindal (2018); Rathee et al. (2020); Huang et al. (2022). However, private Transformer inference remains expensive: matrix multiplications incur substantial communication, and nonlinear layers require comparisons, exponentiation, reciprocal or inverse square root, and truncation. Function secret sharing (FSS) is attractive in the preprocessing model because it moves input-independent computation offline, leaving a lightweight online phase Boyle et al. (2015). Its low online communication and latency make FSS particularly attractive for private Transformer inference.

Private Transformer systems reduce secure-inference costs through specialized protocols, faster linear or nonlinear evaluation, model modification, and quantized execution Hao et al. (2022); Dong et al. (2025); Luo et al. (2024); Pang et al. (2024); Lu et al. (2025b); Li et al. (2024); Kei & Chow (2025); Li et al. (2023); Akimoto et al. (2023); Zeng et al. (2023); Wang & Kuang (2024); Zhang et al. (2025). Quantization provides another lever because secure costs grow with operand bitwidth: SecureQ8 established its value for MPC inference, and Ditto combines dyadic quantization, quantization-aware distillation, dynamic arithmetic rings, and secure type conversion for Transformers Dalskov et al. (2020); Wu et al. (2024). BOLT, CipherFormer, and PPQformer further explore quantized or mixed-bitwidth secure Transformer execution Pang et al. (2024); Wang & Kuang (2024); Dong & Chen (2025). FSS-based systems already support several forms of reduced-precision execution: Boyle et al. develop mixed-mode and fixed-point FSS Boyle et al. (2021); LLAMA accepts low- and mixed-bitwidth representations from an external converter; Pika truncates wide fixed-point inputs before nonlinear FSS evaluation; and FSSiBNN embeds bitwidth conversion into binary activation and max-pooling protocols Gupta et al. (2022); Wagh (2022);

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.

Yang et al. (2024). SIGMA develops FSS protocols for Transformer inference, whereas FuseFSS compiles fixed scalar-operator specifications into batched FSS evaluations Gupta et al. (2024); Ma et al. (2026). In these systems, the bitwidths and operator specifications are fixed during model conversion or protocol design, rather than optimized together with the model's training dynamics.

For an FSS backend, realizing this potential requires more than low-bit tensors. An FSS nonlinear operator is defined not only by its nominal operand bitwidth, but also by its finite input domain, signed headroom, LUT payload encoding, output ring, and rescaling rules. Two operators with the same tensor bitwidth may therefore require different FSS keys and produce different model-visible outputs. This creates three coupled challenges. First, each nonlinear operator must compress a potentially wide activation distribution into a small FSS domain without frequent saturation or an overly large table. Second, one operator's output grid and fixed-point scale must match the representation the next operator expects across additions, reductions, LUTs, and ring conversions. Third, pruning, calibration, and quantization-aware training can shift activation distributions, so a range valid during calibration may fail after fine-tuning or distillation. Without coordinating these choices, a system must either retain wide FSS domains, increasing key material and communication, or compress activations aggressively, risking saturation and training–deployment mismatch. Misaligned output scales can further introduce online ring conversion and truncation at operator boundaries.

In this work, we ask the following question:

Can model quantization and FSS protocol instantiation be jointly optimized so that training and secure execution share the same fixed-point encodings, activation ranges, FSS input domains, LUT grids, and rescaling, truncation, and ring-conversion rules?

We answer this question with PROTOQUANT , an FSS-aware quantization framework. Unlike prior FSS systems, which fix these choices during model conversion or protocol compilation, PROTOQUANT exposes them to calibration and quantization-aware training. PROTOQUANT maintains a unified fixed-point semantic with f fractional bits over $\mathbb{Z}_{2^{64}}$, where a ring element z represents the real value $\mathrm{int}_{64}(z)/2^f$. It further compresses the input of each costly nonlinear operator onto a calibrated low-bitwidth grid. For example, GELU inputs are rounded, using layer-specific scales, onto an 8-bit dyadic grid. These grids are designed so that each grid point directly serves as an FSS lookup index: ring conversion reduces to a local shift, while every table entry is pre-encoded at the fixed-point scale required by its consumer, eliminating online rescaling between operators.

Thus, quantization in PROTOQUANT is not merely bitwidth reduction. It jointly specifies the FSS input domain, LUT encoding, arithmetic ring, and truncation rules. Because the input-domain width directly determines FSS key sizes and key-generation tree depth, selecting quantization parameters also selects protocol costs. During calibration and training, the forward pass simulates the deployment-time grid mapping, lookup, truncation, and ring conversion, and incorporates operator input ranges as explicit constraints in the training objective. The resulting model therefore adapts to the numerical behavior actually exposed by the secure protocol.

Building on this framework, we develop quantized FSS protocols for Softmax, GELU, LayerNorm, residual connections, pooling, and classification heads. Softmax maps scores to an 8-bit domain and combines an FSS maximum tree with exponentiation and reciprocal LUTs; GELU uses a calibrated grid lookup; LayerNorm, residual connections, pooling, and classification heads share the same protocol-aligned fixed-point interface. These protocols retain the intended Transformer operators at their specified quantized grids, rather than replacing them with polynomial or architectural surrogates. Table 1 summarizes the distinction between PROTOQUANT and prior FSS-based systems.

Our results. We implement PROTOQUANT and evaluate it on multiple Transformer benchmarks under representative network settings. Compared with SIGMA, our direct FSS-based Transformer baseline, PROTOQUANT reduces Softmax online latency and communication by 77% and 50%, and GELU by 85% and 36%, while retaining 97.8–98.7% of the teacher's accuracy across three GLUE tasks; a same-kernel comparison(Appendix D) attributes the Softmax and GELU gains to the protocol rather than the implementation.

Contributions. Our main contributions are as follows:

- We propose an FSS-aware quantization framework that jointly selects model bitwidths, FSS input domains, lookup-table encodings, and fixed-point and ring-conversion rules. It simulates

Table 1: Comparison with prior FSS-based secure inference systems. A checkmark denotes full support, while △ denotes static or partial support without training-time co-design.

Feature	Mixed-Mode FSS	LLAMA	Pika	FSSiBNN	SIGMA	FuseFSS	PROTOQUANT
FSS backend	✓	✓	✓	✓	✓	✓	✓
Model-side low-/ mixed-bit quantization	✓	✓	✓	✓	×	✓	✓
FSS-domain compression or bitwidth conversion	△	✓	✓	✓	△	△	✓
End-to-end Transformer	×	×	×	×	✓	✓	✓
Calibrated per-operator FSS input grids and output scales	×	×	×	×	×	×	✓
Training simulates quantization, lookup, truncation, and ring conversion	×	×	×	×	×	×	✓

References. Mixed-Mode FSS Boyle et al. (2021); LLAMA Gupta et al. (2022); Pika Wagh (2022); FSSiBNN Yang et al. (2024); SIGMA Gupta et al. (2024); FuseFSS Ma et al. (2026).

deployment-time quantization and conversion during calibration and training, enabling models to accommodate rounding, range, and scale effects.

- We design efficient, operator-faithful quantized protocols for Softmax (an FSS maximum tree with exponentiation and reciprocal lookup tables), GELU (calibrated grid lookup), LayerNorm, residual connections, pooling, and classification heads. Based on these protocols, we build an end-to-end private Transformer inference system.
- We implement and release our framework as open-source code[1], and evaluate it on multiple benchmarks against SIGMA, our direct FSS-based Transformer baseline, achieving up to 85% lower operator latency and 63% lower operator communication with pruning.

2 PRELIMINARIES

2.1 NOTATION

For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. Bold lowercase and uppercase letters denote vectors and matrices, respectively. For $b \in \mathbb{N}$, $\mathbb{Z}_{2^b}$ denotes the ring of integers modulo 2^b, and $[\![x]\!]_b$ denotes an additive sharing of $x \in \mathbb{Z}_{2^b}$. We use signed fixed-point encoding in $\mathbb{Z}_{2^{64}}$ with $f = 12$ fractional bits. Namely, an integer $z \in \mathbb{Z}_{2^{64}}$ represents the real value $\mathrm{int}_{64}(z)\, 2^{-f}$, where $\mathrm{int}_b(\cdot)$ denotes the signed two's-complement interpretation. We write $\mathrm{ltrunc}_t(\cdot)$ for a local arithmetic right shift by t bits. A quantized tensor x is represented as $x \approx s_x q_x$, where q_x is an integer code and $s_x = 2^{e_x}$ is a public dyadic scale. For an FSS lookup table, $\mathrm{LUT}^{B_{\mathrm{in}} \to B_{\mathrm{out}}}$ denotes evaluation on a B_{in}-bit secret index that returns a shared value in $\mathbb{Z}_{2^{B_{\mathrm{out}}}}$.

2.2 THREAT MODEL

We assume a static, semi-honest PPT adversary corrupting at most one party. Public: model architecture, tensor dimensions, sequence length, padding mask, frozen operator contracts, embedding tables, and LayerNorm affine parameters (γ, β). The embedding carve-out is necessary without a dealer: keying a private-table lookup would make per-inference key material proportional to table size. Client inputs, all other model parameters, and intermediate shared values stay private. Malicious deviations and side-channel leakage are outside our scope.

We target simulation-based security for both preprocessing and online execution, relying on the underlying distributed-FSS and secure arithmetic protocols (Xing et al., 2025). Unlike SIGMA Gupta et al. (2024), which uses a trusted dealer, our two parties interactively generate the required FSS keys without access to client inputs (Section 3.4).

[1] https://anonymous.4open.science/r/anonymous-code-6D2E/

2.3 QUANT FRAMEWORK

Post-training quantization (PTQ) Nagel et al. (2021); Xiao et al. (2023). PTQ starts from a trained floating-point checkpoint and quantizes it without further gradient-based optimization. A calibration subset is used to determine activation ranges and scales; the resulting weight and activation quantizers are then frozen for deployment.

Quantization-aware training (QAT) Jacob et al. (2018). QAT incorporates quantization into training by replacing forward-pass tensors with fake-quantized values while keeping floating-point parameters for optimization. Since rounding is non-differentiable, QAT commonly uses a straight-through estimator to approximate its backward gradient Bengio et al. (2013). Hence the model can adapt its weights to the errors induced by the frozen quantization grids before integer deployment.

2.4 FUNCTION SECRET SHARING

FSS represents a function $f : \mathcal{X} \to \mathcal{G}$ by distributing two keys to two parties. The randomized key-generation algorithm $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)$ takes the security parameter λ and the function f. Each party receives only its own key k_b and, on input $x \in \mathcal{X}$, locally computes an output share $y_b \leftarrow \mathsf{Eval}(k_b, x)$. Correctness requires that combining the two shares recovers $y_0 + y_1 = f(x)$, where $+$ is the group operation in $\mathcal{G}$, e.g., XOR for bit-valued outputs or addition modulo 2^ℓ for outputs in $\mathbb{Z}_{2^\ell}$. Neither party should learn the function from its key alone Boyle et al. (2015).

3 METHOD

This section presents PROTOQUANT , a two-party private inference pipeline that jointly designs model quantization and secure operator arithmetic. Section 3.1 outlines the pipeline; Sections 3.2–3.4 detail model compression, quantized nonlinear operators, and contract-aligned training and execution.

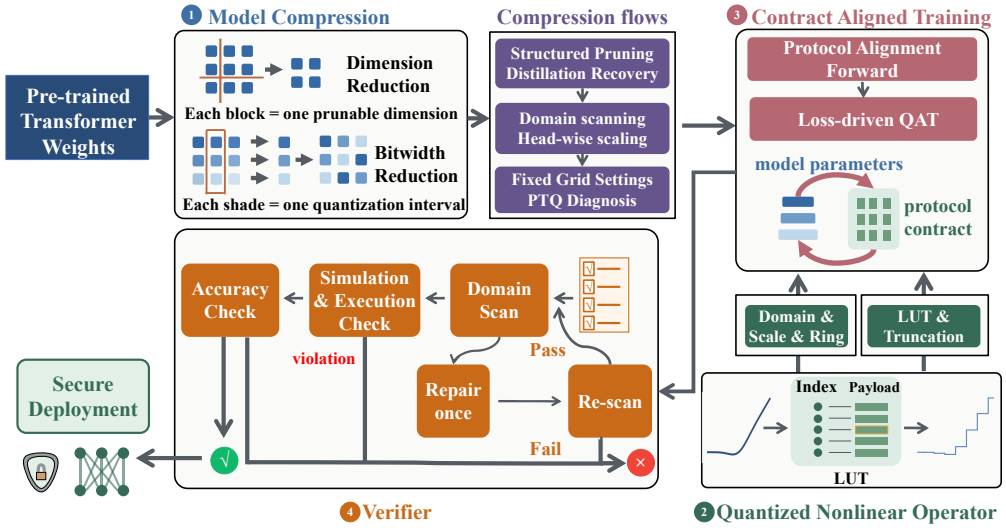

Figure 1: Overview of PROTOQUANT .

3.1 DESIGN OVERVIEW

Figure 1 illustrates four stages. First, we compress the model and freeze its tensor shapes and quantization scales. Second, we instantiate secure operators with frozen numerical contracts specifying domains, scales, rings, truncation rules, and LUT contents. Third, we retrain the model against these contracts, reproducing the deployed lookup, truncation, and indexing semantics. Finally, we check domain compliance, protocol–simulation agreement, and task accuracy, applying the pre-registered repair rule at most once when needed and deploying only artifacts that pass all checks.

3.2 MODEL COMPRESSION

The first stage shrinks the model before any secure operator is involved. Its two budgets are the tensor dimensions, which decide how many times secure primitives are invoked, and the numerical precision, which decides the width of their input domains.

Dimension reduction. We prune attention heads with a task-aware gradient criterion, selecting them globally across layers under a fixed budget while keeping at least one head per layer. Removing heads is the most effective structural lever for private inference: each removed head eliminates an entire Softmax—the most expensive secure operator—rather than merely shrinking a matrix. We then distill the pruned student from the fine-tuned teacher with the usual logit objective. Reducing the FFN width was also evaluated but not retained: its accuracy cost outweighs the limited protocol benefit, since FFN blocks are already handled cheaply by the known-weight linear protocol.

Freezing the quantization grid. On the compressed model, we place 8-bit fake-quantization grids on linear weights and at activation interfaces between operators; the final logits stay in the protocol's fixed-point output format. Each grid uses a public dyadic scale, so every inter-operator rescaling reduces to a prescribed integer shift and truncation. Weight scales are estimated per output channel from the checkpoint, and activation scales are calibrated on a training split; both are then frozen.

Score-range repair. This failure mode is invisible to bitwidth-only quantization: PTQ on the frozen grids already violates the contract. On QNLI, pre-Softmax attention scores exceed the FSS Softmax gate's comparison domain $2{,}125{,}043$ times on the calibration split, rendering the checkpoint undeployable. Violations concentrate in a few attention heads whose score distributions sharpen after pruning. We therefore repair the compressed model once, before training, with a pre-registered head-wise rule: for each head h, we measure its peak score p_h on the calibration split and rescale its query and key projections by $f_h = \min\left(1, \sqrt{\tau_r / p_h}\right)$, where the repair target $\tau_r = 14$ lies strictly inside the Softmax contract domain $[-16, 15.875)$ (Section 3.3.1); the margin absorbs the gap between calibration-split peaks and unseen inputs. Shapes and all other weights stay unchanged, and the factors come from training data only. On QNLI this eliminates all $2{,}125{,}043$ violations without retraining. Weight and activation scales are then re-derived on the repaired checkpoint and frozen. Repair removes the structural violations, but training under the frozen grids can itself push scores back out of range, motivating the contract-aligned training of Section 3.4.

3.3 QUANTIZED NONLINEAR OPERATOR DESIGN

At the model level, we preserve the original definitions of Softmax, GELU, and LayerNorm. Rather than replacing them with a polynomial or another learned surrogate, we evaluate their quantized forms on finite integer grids. Thus, the remaining numerical error arises from the prescribed quantization, clipping, and integer execution path, rather than from substituting a different nonlinearity.

As the protocol layer, we adopt an existing FSS scheme: the comparison and LUT decomposition of SIGMA provides the secure evaluation primitives for maximum, comparison, and table lookup, and we build directly on top of it. What this work adds is the quantization layer above those primitives. For each operator, we co-design its fixed-point encoding, finite FSS input domain, LUT payload encoding, output ring, and truncation rule with the quantized model, and all gate material is generated through dealerless two-party preprocessing instead of a trusted dealer.

3.3.1 SOFTMAX

We preserve numerically stable Softmax $y_i = \frac{\exp(x_i - x_{\max})}{\sum_{j\in[k]} \exp(x_j - x_{\max})}, \qquad x_{\max} = \max_{j\in[k]} x_j$, on secret-shared score rows, evaluating it via an FSS comparison-and-lookup decomposition (maximum reduction, exponential LUT, reciprocal LUT; Appendix C.1).

Because preprocessing is dealerless, the parties generate DPF material for each comparison and lookup gate, so lookup index width determines preprocessing time, key material, and peak memory. We therefore confine scores to $[-16, 15.875)$ and map them to 256 slots of width $1/8$ *without interactive re-encoding*:a public offset plus local share-wise truncation. The reconstructed slot deviates

from ideal rounding by at most one grid point; this rounding is retained in the operator contract and reproduced during training, not corrected online. A comparison tree finds the row maximum, and each nonpositive difference is reduced modulo 2^8 to address the exponential table directly, avoiding online rescaling. The table returns exponentials on a 2^{-8} grid with a public tail cutoff, while the public mask zeros masked positions. A reciprocal table and one shared multiplication per position produce probabilities on a $1/256$ grid without re-normalization; this deviation is part of the training contract. Section 4.4 shows that the 8-bit index reduces key-generation time, peak memory, and key size by two orders of magnitude relative to a direct 16-bit index, at small, trainable accuracy cost.

3.3.2 GELU

For $x \in \mathbb{R}$, the GELU used is $\mathrm{GELU}(x) = \frac{x}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$. We retain this functional form: each layer-specific LUT is populated by quantizing its GELU values, rather than by substituting a polynomial, ReLU-based, or other surrogate activation. The calibrated dyadic scale of FFN1 defines the LUT input grid, while the LUT payload is encoded directly on the input grid consumed by FFN2. Consequently, the FFN1–GELU–FFN2 chain requires no additional rescaling or requantization at the LUT boundaries. It forms a single integer path, with only the intended LUT discretization, and avoids the extra conversion cost and rounding error of independently quantized operators. The exact slotization rule, per-layer table construction, and frozen parameters are given in Appendix C.2.

3.3.3 LAYERNORM

For a row $\mathbf{x}$, LayerNorm computes $\mathrm{LN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x}-\mu(\mathbf{x})}{\sqrt{\sigma^2(\mathbf{x})+\varepsilon}} + \beta$. The mean and variance use shared arithmetic; the only nonlinear operation is the reciprocal square root of the adjusted variance.

A direct LUT over the wide variance domain is impractical, so we adopt an existing binary-floating decomposition: the adjusted variance is converted on secret shares to a compact index (exponent and mantissa) addressing a reciprocal-square-root LUT, and neither the variance nor its exponent is opened. The result is multiplied with the centered row before applying the public affine parameters.

Our contribution is the layer-wise numerical instantiation. Although ε is a single fixed model constant, its integer code depends on the variance scale of layer ℓ. We therefore freeze $c_\ell = \lfloor \varepsilon / s_\ell^{\mathrm{var}} \rceil$ together with that layer's table and certified input domain, so every layer realizes the same real-valued ε rather than different effective constants after quantization. The full per-row flow, frozen per-layer parameters, and optional runtime range guard appear in Appendix C.3.

3.4 CONTRACT-ALIGNED TRAINING AND DEALERLESS EXECUTION

Training against the deployed arithmetic. We retrain the compressed model using a protocol-faithful integer forward pass: the same LUT entries, lookup-index discretization, and share-wise truncation rules used at deployment. Because the same integer semantics are composed end-to-end, errors that share an intermediate value—a slot rounding feeding a lookup feeding a normalization—are correlated in training exactly as at deployment; nothing is simulated per-operator in isolation. Gate domains enter the objective through an explicit domain-violation loss. Let $\mathcal{G}$ be the set of protocol-gated values (attention scores and LayerNorm statistics) with certified bounds $[-B_g, B_g)$ from the frozen contracts; the training objective is $\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \mathcal{L}_{\mathrm{KD}} + \lambda \sum_{s \in \mathcal{G}} \mathrm{relu}\big(|s| - B_g\big)$, where $\mathcal{L}_{\mathrm{task}}$ and $\mathcal{L}_{\mathrm{KD}}$ are the task and logit-distillation losses of Section 3.2, and the hinge pushes every gated value back inside its domain instead of relying on clipping. We use $\lambda = 0.01$ with $B = 15$ for SST-2 and $\lambda = 0.02$ with $B = 14$ for QNLI and MRPC; the choice is swept in Appendix G for QNLI, the task under the strongest domain pressure; SST-2 uses the smaller λ without a sweep, since its repaired scores already lie inside the domain and the hinge is rarely active there. Bitwidth-only QAT does not model these operator-specific effects. In particular, LUT rounding, local truncation, and share-dependent carries are not generic quantization noise; they are fixed numerical behaviors that the model must learn to absorb.

Certification and deployment. After training, each candidate deployment artifact — the model checkpoint, with its frozen scales, LUTs, and FSS input domains — is executed by the integer reference chain on a fixed certification split from the training data. We inspect every input to each

bounded FSS comparison and lookup gate. A scan has *zero violations* if every attention score, LayerNorm statistic, and other bounded gate inputs lie within their corresponding frozen domains. A candidate is accepted for deployment only after a zero-violation scan. This is an empirical certification of the final artifact on the designated scan, rather than a claim about all possible future inputs. If a candidate has residual structural attention-score violations, we apply the same pre-registered head-wise rescaling rule of Section 3.2 at most once, without further training, preserving the model shape and all unrelated parameters. The resulting artifact is re-evaluated for both zero violations and task accuracy; candidates that still fail are rejected, and only a final passing version is deployed. Deployability is therefore an explicit, auditable acceptance criterion rather than an assumption.

4 EXPERIMENTS

We evaluate PROTOQUANT with BERT-base on three datasets: SST-2, QNLI, and MRPC to assess utility under different downstream data (§4.1). We then compare the online cost of our Softmax, GELU, and LayerNorm operators with SIGMA under matched settings (§4.2), and measure the resulting end-to-end inference cost (§4.3). In the ablation study (§4.4), we study (1) compressing the FSS lookup domain relative to a direct wide-domain lookup and (2) training against the complete operator contract rather than bitwidth-only QAT.

Experiment Settings. We evaluate on a server-grade testbed consisting of an Intel Xeon Gold 6238R (112 hardware threads) and two NVIDIA A10 (22 GB) GPUs used for training and calibration; all protocol measurements are CPU-only. The two parties run as isolated processes on one host and communicate over TCP. We simulate two network settings: a LAN setting over loopback (0.1 ms RTT), and a loopback-emulated WAN setting using `tc netem` with 305 Mbit/s bandwidth and 60 ms round-trip latency, matching the WAN parameters of SIGMA Gupta et al. (2024).

Baselines. Our primary baseline is **SIGMA** Gupta et al. (2024), the direct FSS-based two party Transformer system with a trusted dealer. We compare against its reported operator level numbers under SIGMA's S-CPU configuration.

4.1 ACCURACY

Table 2 reports accuracy across the model–protocol pipeline; Appendix Table 7 lists the pruned dimensions. Head pruning preserves utility while removing 25% of Softmax calls: the distilled student matches or exceeds the teacher on all three tasks, and on MRPC the selected pruned checkpoint (seed 1) matches the teacher's validation accuracy ($358/408$ each) with a slightly different F1. Bitwidth-only quantization is insufficient. PTQ violates the frozen FSS domain on QNLI. On SST-2, generic QAT drops from 92.89% in its idealized simulation to 82.57% under the deployed integer reference chain, a 10.32-point gap. On QNLI and MRPC, simulated accuracies are 91.21% and 87.99%, but 35 and 3 attention-score values on the validation split violate the Softmax domain, triggering fail-closed evaluation. These held-out failures are distinct from training-split certification

Table 2: Accuracy along the model–protocol co-design pipeline. Stage codes: P+KD = pruning and knowledge distillation, PTQ = post-training quantization, G-QAT = generic QAT, and PA-QAT = protocol-aware QAT. Values are mean±std over certified seeds when available.

Stage	Training	SST-2	QNLI	MRPC	
		Acc	Acc	Acc	F1
FP32	–	92.43	91.27	87.75	91.35
P+KD	3 ep. / MRPC: 9 ep.	92.89[a]	91.52±0.02	87.75	91.38
PTQ	–	69.95	undepl.	83.33	87.64
G-QAT	3 ep.	82.57	80.18[e]	82.84[e]	–
PA-QAT	3 ep.	**91.02±0.33**	**90.06±0.23**	**85.78±0.74**	**89.75±0.48**
Ret.	–	98.5%	98.7%	97.8%	98.2%

'–' denotes unavailable results; 'undepl.' denotes undeployable checkpoints. [a] single seed; [e] after one application of the repair rule (Section 4.1).

(Appendix I) and mirror the rejected QNLI seed 3. The pre-registered repair rule eliminates observed violations by rescaling 21 and 15 heads, respectively, yet deployed accuracies remain 80.18% and 82.84%: domain compliance alone does not close the semantic gap. Protocol-aware QAT substantially narrows this gap, achieving 91.02 ± 0.33 on SST-2, 90.06 ± 0.23 on QNLI, and 85.78 ± 0.74 Acc / 89.75 ± 0.48 F1 on MRPC, retaining 98.5%, 98.7%, and 97.8% / 98.2% of the corresponding teacher metrics. All reported checkpoints pass the certification scan with zero violations; QNLI results average only the two certified seeds out of four trained (Appendix I). These results support training against deployed lookup, truncation, and indexing semantics beyond bitwidth alone.

4.2 Operator cost

Table 4 compares the online cost of our nonlinear operators with SIGMA under matched BERT-base settings, and Table 3 reports their sensitivity to network latency (GELU uses the LUTSS path). For the unpruned configuration, ProtoQuant reduces Softmax latency and communication by 77.3% and 50.1%, respectively. GELU achieves an 84.6% latency reduction and a 36.0% communication reduction, while LayerNorm has comparable latency to SIGMA and uses 25.8% less communication.

Table 3: Online cost under LAN and WAN settings.

Operator	Time (s)		Comm. (MB)
	LAN	WAN	
Softmax	1.04	14.1	184.5
GELU	0.57	5.19	160
LayerNorm	0.68	15.0	118.8

These online gains stem from operator-specific integer contracts: consumer-grid LUT payloads and batched FSS evaluation eliminate conversion steps. Lookup-domain compression mainly enables practical dealerless preprocessing by reducing key-generation time, key material, and peak memory; we evaluate this offline impact separately in Section 4.4, same-core check in Appendix D.

Table 4: Comparison of secure Transformer inference. SIGMA Gupta et al. (2024) is measured under its S-CPU configuration. "Ours" is the full model, while "Ours_p" denotes the pruned variant.

Module	Online time (s)					Comm (MB)				
	SIGMA	Ours	↓ (%)	Ours_p	↓ (%)	SIGMA	Ours	↓ (%)	Ours_p	↓ (%)
Softmax	4.59	1.04	77.3%	1.01	78.0%	370	184.5	50.1%	138.3	62.6%
GELU	3.71	0.57	84.6%	0.57	84.6%	250	160	36.0%	160	36.0%
LayerNorm	0.70	0.68	2.9%	0.68	2.9%	160	118.8	25.8%	118.8	25.8%

4.3 Inference latency

Table 5 decomposes the online cost of one BERT-base layer and places the full model against the end-to-end baseline. The full model costs $4.8\,\mathrm{s}$ and $684\,\mathrm{MB}$ online. Offline, dealerless key generation for the full model takes $9.4\,\mathrm{min}$ on LAN and produces $16.3\,\mathrm{GB}$ of stored key material per party (Appendix E). The largest share is the linear layers, whose local matrix multiplication dominates CPU time; the nonlinearities that are the focus of this paper are already cheap. Against CrypTen's GPU-accelerated execution we are $4.5\times$ faster with $12\times$ less communication, while running on CPU only. While training against the contract keeps deployments inside the certified domains, no training-time guarantee covers arbitrary future inputs; sites requiring fail-closed behavior at runtime can enable the optional FSS range guard (Appendix C.3), which adds

Table 5: Performance breakdown of BERT-base and end-to-end performance comparison with CrypTen Knott et al. (2021). Times are in ms and communication costs in MB; LN denotes LayerNorm. The CrypTen row quotes the GPU numbers reported in SIGMA's evaluation Gupta et al. (2024). The last row extrapolates the per-layer row over 12 layers (unpruned model).

Operation	Shape × count	Rounds	Time(ms)	Comm(MB)
QKV	$(128\times768\times768)\times3$	3	47	4.7
$QK^\top$	$(128\times64\times128)\times12$	12	31	1.6
Softmax	$(128\times128)\times12$	27	87	15.4
AV	$(128\times128\times64)\times12$	12	34	3.1
Attn	$(128\times768\times768)\times1$	1	16	1.6
LN	128×768	16	27	4.8
FFN1	$(128\times768\times3072)\times1$	1	38	1.6
GELU	128×3072	2	48	13.3
FFN2	$(128\times3072\times768)\times1$	1	44	6.3
LN	128×768	16	27	4.8
Per-layer		91	**399**	**57.0**
CrypTen (GPU)			21550	8340
Ours (×12)			**4800**	**684**

$\sim$8.5 s online and $\sim$173 MB communication (end-to-end totals in Appendix C.3). Our gains are orthogonal to GPU acceleration: the same known-weight and domain-reduction techniques apply to a GPU-backed linear layer, which we leave to future work.

4.4 Ablation Study

Our central claim: input-domain width, not the tensor bitwidth, drives FSS cost. Figure 2 isolates this by varying only the lookup index width of the Softmax exponential table while holding the checkpoint, input range, output scale, ring, truncation, backend, batch, and network fixed (exact values in Appendix Table 13). Compressing the index from a direct 16-bit score encoding to the deployed 8-bit grid cuts key-generation time by $85\times$, peak memory by $159\times$, and key size by $128\times$, while the deployed grid's output-probability error stays small (0.008 mean vs. 0.004 for 16-bit; Table 6). The error curves in Figure 2 are measured on uniformly spaced grids over $[-16, 0)$ for a controlled domain-width comparison; the deployed table is evaluated separately (Table 6).

Cost measurements start at 8 bits because the FSS key-generation implementation requires at least 8 input bits. The trade-off has a clear knee across the measured widths (error measured at 6–16 bits, cost at 8–16 bits, both including a measured 9-bit point): the error falls steeply up to about 9 bits and then saturates at the normalization chain's noise floor, so further index bits buy almost no accuracy, while all three cost measures grow roughly exponentially. Two implications follow. First, domain compression is not merely an efficiency optimization but a deployability requirement: the 16-bit variant already needs 8.3 GB of peak keygen memory for a single operator, so a direct wide-domain instantiation is infeasible without a trusted dealer. Second, the accuracy cost of the compressed grid is small and absorbable by contract-aligned training, which is exactly the trade our framework makes deliberately.

Table 6: Absolute Softmax probability error over 2,000 rows of 128 scores sampled uniformly from $[-16, 15.875)$.

	Max	Mean	P99
8-bit	0.0229	0.0082	0.0158
16-bit	0.0056	0.0038	0.0051
8-bit vs. 16-bit	0.0234	0.0080	0.0156

Per-row error is the maximum over positions. Columns report the maximum, mean, and P99 over rows. The first two rows compare against fp64, while the last compares the two grids.

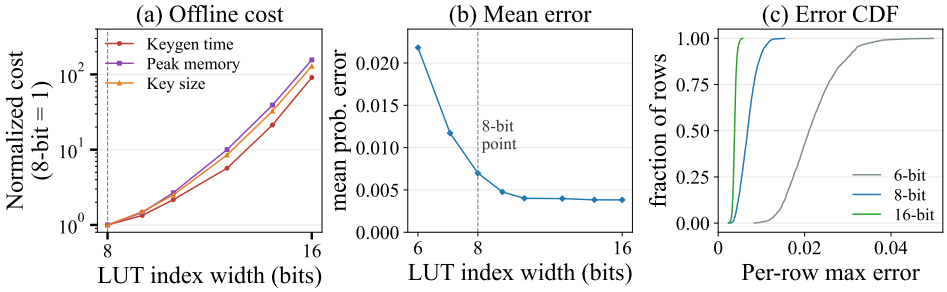

Figure 2: LUT-domain width vs. offline cost and Softmax error. Costs are normalized to their 8-bit values. Errors use uniformly spaced grids over $[-16, 0)$. Markers show every measured width (error: 6–16 bits; cost: 8–16 bits); the dashed line marks the deployed 8-bit configuration.

5 Conclusion

We introduced contract-aligned quantization, which trains models against the integer semantics of deployed FSS operators—ring choice, scaling, slot domain, LUT encoding, and truncation route—and combines fixed repair rules with empirical certification, making deployability an auditable acceptance condition. On BERT-base, bitwidth-only QAT loses 10.32 accuracy points on SST-2 and fails domain checks on QNLI and MRPC, whereas contract-aligned training retains 97.8–98.7% of teacher accuracy. Against SIGMA, our operators reduce online latency by up to 85% and communication by up to 63% (50% without pruning), while domain compression makes dealerless preprocessing practical. Remaining overhead is largely linear layers and one-time key generation; future work targets GPU acceleration of both and extensions beyond FSS backends (Appendix J).

AI USE STATEMENT

We used generative AI tools only for language polishing and code debugging; all AI-assisted work was reviewed and verified by the authors, who take responsibility for the final content.

REPRODUCIBILITY STATEMENT

We provide the frozen operator contracts—input and output domains, dyadic scales, table constructions, and truncation rules—together with per-operator algorithms and production parameters in Appendices C.1, C.2, and C.3. Model configurations and pruning settings appear in Appendix B.

REFERENCES

Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving transformer with MPC. In *2023 IEEE 8th European Symposium on Security and Privacy*, pp. 392–410, 2023. doi: 10.1109/EuroSP57164.2023.00031.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology – EUROCRYPT 2015*, pp. 337–367. Springer, 2015. doi: 10.1007/978-3-662-46803-6_12.

Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Advances in Cryptology – EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pp. 871–900. Springer, 2021. doi: 10.1007/978-3-030-77886-6_30.

Yuntian Chen, Tianpei Lu, Zhanyong Tang, Bingsheng Zhang, Zhiying Shi, Yuxiang Luan, and Zhuzhu Wang. Sok: Private transformer-based model inference. In *35th USENIX Security Symposium (USENIX Security 26)*, pp. 1127–1146, Baltimore, MD, August 2026. USENIX Association. ISBN 978-1-939133-58-8. URL https://www.usenix.org/conference/usenixsecurity26/presentation/chen-yuntian.

Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, 2020. doi: 10.2478/popets-2020-0077.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019. doi: 10.18653/v1/N19-1423.

Benchang Dong and Zhili Chen. PPQformer: Privacy-preserving quantized transformer for efficient and secure inference. In *Proceedings of the 2025 2nd International Conference on Generative Artificial Intelligence and Information Security*, pp. 115–119. ACM, 2025. doi: 10.1145/3728725.3728743.

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

Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. LLAMA: A low latency math library for secure inference. *Proceedings on Privacy Enhancing Technologies*, 2022 (4):274–294, 2022. doi: 10.56553/popets-2022-0109.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2024(4):61–79, 2024. doi: 10.56553/popets-2024-0107.

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*, volume 35, pp. 15718–15731, 2022.

Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *31st USENIX Security Symposium*, pp. 809–826, 2022. URL https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Andes Y. L. Kei and Sherman S. M. Chow. SHAFT: Secure, handy, accurate and fast transformer inference. In *Proceedings of the 2025 Network and Distributed System Security Symposium*, 2025. doi: 10.14722/ndss.2025.242287.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 4961–4973. Curran Associates, Inc., 2021.

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P. Xing, and Hao Zhang. MPCFormer: Fast, performant and private transformer inference with MPC. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CWmvjOEhgH-.

Zhengyi Li, Kang Yang, Jin Tan, Wen-Jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. Nimbus: Secure and efficient two-party inference for transformers. In *Advances in Neural Information Processing Systems*, volume 37, pp. 21572–21600, 2024. doi: 10.52202/079017-0680.

Tianpei Lu, Bingsheng Zhang, Xiaoyuan Zhang, and Kui Ren. A new PPML paradigm for quantized models. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025a.

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 *Proceedings of the 2025 Network and Distributed System Security Symposium*, 2025b. doi: 10.14722/ndss.2025.230057.

Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13333–13348. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.790.

Yuhan Ma, Yong Li, and Stefan Schmid. FuseFSS: Efficient secure LLM inference with function secret sharing. *arXiv preprint arXiv:2606.09551*, 2026. Accepted at the 43rd International Conference on Machine Learning.

Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52, 2018. doi: 10.1145/3243734.3243760.

Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pp. 19–38, 2017. doi: 10.1109/SP.2017.12.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy*, pp. 4753–4771, 2024. doi: 10.1109/SP54263.2024.00130.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 325–342, 2020. doi: 10.1145/3372297.3417274.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017.

Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022(4):351–377, 2022. doi: 10.56553/popets-2022-0113.

Weize Wang and Yi Kuang. Cipherformer: Efficient transformer private inference with low round complexity. In *2024 27th International Conference on Computer Supported Cooperative Work in Design*, pp. 1–8, 2024. doi: 10.1109/CSCWD61410.2024.10580655.

Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. Ditto: Quantization-aware secure inference of transformers upon MPC. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 53346–53365. PMLR, 2024. URL https://proceedings.mlr.press/v235/wu24d.html.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Pengzhi Xing, Hongwei Li, Meng Hao, Hanxiao Chen, Jia Hu, and Dongxiao Liu. Distributed function secret sharing and applications. 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/distributed-function-secret-sharing-and-applications/.

Peng Yang, Zoe Lin Jiang, Jiehang Zhuang, Junbin Fang, Siu-Ming Yiu, and Xuan Wang. FSSiBNN: FSS-based secure binarized neural network inference with free bitwidth conversion. In *Computer Security – ESORICS 2024*, Lecture Notes in Computer Science, pp. 229–250. Springer, 2024. doi: 10.1007/978-3-031-70879-4_12.

Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-Jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. MPCViT: Searching for accurate and efficient MPC-friendly vision transformer with heterogeneous attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5029–5040, 2023. doi: 10.1109/ICCV51070.2023.00466.

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

A Related work

A.1 Protocol optimizations for private transformer inference

A large body of work reduces the cost of private Transformer inference Chen et al. (2026) through specialized secure protocols and system-level optimizations Hao et al. (2022); Dong et al. (2025); Luo et al. (2024); Pang et al. (2024); Lu et al. (2025b); Li et al. (2024); Kei & Chow (2025); Li et al. (2023); Akimoto et al. (2023); Zeng et al. (2023); Wang & Kuang (2024); Zhang et al. (2025). These systems combine MPC, homomorphic encryption, and garbled circuits in different ways to optimize the secure evaluation of Transformer operators, including linear layers and nonlinear functions. Together, they demonstrate the importance of protocol design and system integration in reducing the latency and communication costs of private inference.

A.2 FSS-based private inference

FSS is attractive for private inference because it can move input-independent computation to preprocessing and support low-communication evaluation of nonlinear functions Boyle et al. (2015). Prior work has developed mixed-mode and fixed-point FSS representations Boyle et al. (2021). LLAMA supports low- and mixed-bitwidth inputs through an external converter Gupta et al. (2022); Pika truncates wide fixed-point inputs before nonlinear FSS evaluation Wagh (2022); and FSSiBNN incorporates bitwidth conversion into binary activation and max-pooling protocols Yang et al. (2024). At the system level, SIGMA develops FSS protocols for Transformer inference Gupta et al. (2024), while FuseFSS compiles fixed scalar-operator specifications into batched FSS evaluations Ma et al. (2026).

A.3 Quantization for private transformer inference

Quantization reduces secure-computation costs by lowering the bitwidths of model parameters, activations, and intermediate values. SecureQ8 demonstrates the benefits of low-bitwidth MPC inference Dalskov et al. (2020), while Lu et al. study a PPML paradigm tailored to quantized models Lu et al. (2025a). Ditto combines dyadic quantization with quantization-aware distillation, dynamic arithmetic rings, and secure type conversion for Transformers Wu et al. (2024). BOLT, CipherFormer, and PPQformer also explore quantized or mixed-bitwidth execution for private Transformer models Pang et al. (2024); Wang & Kuang (2024); Dong & Chen (2025). None of these works simulates the deployed FSS gate semantics during training.

B Model and pruning settings

Table 7 lists the retained structures used throughout the experiments. Head selection is global under a fixed total budget with at least one head per layer, so the retained counts are heterogeneous across layers rather than uniform. Two points are worth noting. First, pruning removes 25% of the Softmax calls ($144 \rightarrow 108$ heads) while reducing parameters by only 6.5%: attention heads carry few parameters but each one invokes a full secure Softmax, which is exactly the leverage targeted in Section 3.2. Second, the FFN width is retained in all three tasks, since FFN computation is handled by the cheaper known-weight linear protocol; and all training seeds share a single distilled initialization, so pruning is performed once per task.

Table 7: Model dimensions before and after structured head pruning.

Dataset	Attention heads Dense → Pruned	Retained heads per layer	FFN width	Parameters (M) Dense → Pruned	Retained / Reduction
SST-2	144 → 108	[12, 11, 10, 11, 9, 8, 10, 9, 8, 8, 4, 8]	3072	109.48 → 102.40	93.5% / 6.5%
QNLI	144 → 108	[8, 8, 8, 11, 10, 11, 11, 7, 9, 7, 8, 10]	3072	109.48 → 102.40	93.5% / 6.5%
MRPC	144 → 108	[11, 10, 9, 10, 8, 9, 9, 8, 10, 9, 9, 6]	3072	109.48 → 102.40	93.5% / 6.5%

Retained = Pruned / Dense; Reduction = 1 − Retained. Dense = BERT-base.

C DETAILED OPERATOR CONTRACTS

This appendix specifies the exact integer semantics that are frozen by the contracts of Section 3.3 and simulated by the training forward pass.

C.1 SOFTMAX PROTOCOL

Protocol backbone. We follow the FSS decomposition used by SIGMA Gupta et al. (2024): a comparison tree computes the maximum of each secret-shared score row; the nonpositive differences from that maximum are evaluated with a public exponential LUT; the results are summed in the ring; and a second public LUT returns an approximation to the reciprocal of the denominator. One shared multiplication per position followed by share-wise truncation produces the output probabilities. In our dealerless implementation, both comparison and lookup material are generated interactively by the two parties. The lookup domains and numerical grids are co-designed with the quantized model.

Slotization. Let $s_i^{(12)}$ denote the fixed-point score code with $f = 12$, so that $x_i = s_i^{(12)}/2^{12}$. A score is in-domain when $-2^{16} \le s_i^{(12)} \le 2^{16} - 513$, which corresponds to $-16 \le x_i < 15.875$. The protocol adds the public offset 2^{16} and applies ltrunc_9 locally to each share, with one compensation on the SERVER share (SERVER denotes the model-holding party, CLIENT the input owner), producing an 8-bit secret slot. Because the discarded bits are distributed across the shares, the reconstructed slot is $q_i^\star$ or $q_i^\star + 1$, where $q_i^\star = \left\lfloor \frac{s_i^{(12)} + 2^{16}}{2^9} \right\rfloor$. No party performs a local signed conversion or clamp. This share-wise behavior is part of the frozen contract and is reproduced during training. Masked positions receive the designated zero slot, and the public mask forces their exponential outputs to zero, so they do not contribute to the denominator.

Table construction. The public exponential table is defined by $\ell(j) = \begin{cases} 0, & j = 0, \\ 2^{-3}(j - 256), & 1 \le j \le 255, \end{cases}$ $E[j] = \mathrm{round}\big(2^8 \exp(\ell(j))\big)$, using the same half-up rounding rule as the production table. Thus, $E[0] = 256$ encodes $\exp(0) = 1$, while entries corresponding to the tail below -16 round to zero. The maximum is obtained by a comparison tree over the 8-bit slots. The stabilized difference $\hat{z}_i = q_i - q_{\max} \le 0$ is represented in a 9-bit domain, which provides headroom for differences in $[-255, 0]$. Reducing $\hat{z}_i$ modulo 2^8 gives the exponential-LUT index.

Normalization. Let $D = \sum_i e_i$ be the integer denominator. For rows of length $n \le 128$, every valid nonempty row satisfies $256 \le D \le 256n \le 2^{15}$. An all-masked row has $D = 0$. We use the reciprocal table $R_{15}[d] = \begin{cases} 2, & d = 0, \\ \mathrm{round}(2^{16}/d), & 1 \le d < 2^{15}, \end{cases}$ addressed by $D \bmod 2^{15}$. The special value $R_{15}[0] = 2$ covers both $D = 2^{15}$ and $D = 0$. In the all-masked case, the public mask already forces all outputs to zero, so the reciprocal value has no effect. For $129 \le n \le 255$, we use a 16-bit reciprocal table; longer rows are outside the current contract.

For each position, the protocol computes one 64-bit Beaver product $u_i = e_i r$ and applies $\mathrm{ltrunc}_8(u_i)$. Since u_i represents approximately $2^{16} p_i^{\mathrm{true}}$ before truncation, the output is represented on the nominal $1/256$ grid, with the prescribed share-wise truncation error included in the protocol contract. Outputs are not re-normalized. Reciprocal-table rounding satisfies $\left|Dr - 2^{16}\right| \le \frac{D}{2}$.

Row-sum bias of share-wise truncation. The final share-wise shift biases each position downward by about one grid point in expectation. On the certified model's calibration rows, the single-party output sums to 0.94–1.00 per row depending on score sharpness (pure table and truncation error), while two-party share truncation lowers the row sum to ≈ 0.50: each of the 128 positions loses about $1/256$ in expectation, and a zero position can reconstruct as $-1/256$ through the one-ULP downward borrow. No position is negative in the output grid itself. This bias is part of the frozen operator contract and is simulated during training, so the deployed model is calibrated to it; the integer-chain validation (Appendix H) confirms that the two-party execution matches the trained

Algorithm 1 Protocol-Aligned Softmax

Require: Secret-shared score codes $\{[\![s_i^{(12)}]\!]_{64}\}_{i=1}^n$, public mask $\mathbf{m} \in \{0,1\}^n$, and $1 \le n \le 128$. For every $m_i = 1$, $-2^{16} \le s_i^{(12)} \le 2^{16} - 513$; otherwise abort in aggregate.
Ensure: $\{[\![p_i]\!]_{64}\}_{i=1}^n$ on the nominal $1/256$ grid, with masked entries equal to zero.

for $i = 1, \dots, n$ **do**
  if $m_i = 1$ **then**
    $[\![q_i]\!]_8 \leftarrow \mathrm{ltrunc}_9\big([\![s_i^{(12)} + 2^{16}]\!]_{64}\big) \bmod 2^8$ ▷ public offset and share-wise truncation; SERVER-side $+1$ compensation
  else
    $[\![q_i]\!]_8 \leftarrow [\![0]\!]_8$
  end if
end for
$[\![q_{\max}]\!]_8 \leftarrow \mathrm{SecureMax}\big(\{[\![q_i]\!]_8\}_{i=1}^n, \mathbf{m}\big)$ ▷ masked maximum; comparisons use one headroom bit
for $i = 1, \dots, n$ **do**
  $[\![j_i]\!]_8 \leftarrow \big(\mathrm{extend}_{8,9}([\![q_i]\!]_8) - \mathrm{extend}_{8,9}([\![q_{\max}]\!]_8)\big) \bmod 2^8$ ▷ direct LUT index after maximum subtraction
  $[\![e_i]\!]_{64} \leftarrow m_i \cdot \mathrm{LUT}^{8\to 64}([\![j_i]\!]_8)$ ▷ masked entries contribute zero
end for
$[\![D]\!]_{64} \leftarrow \sum_{i=1}^n [\![e_i]\!]_{64} \pmod{2^{64}}$ ▷ $D = 0$ only for an all-masked row
$[\![d]\!]_{15} \leftarrow [\![D \bmod 2^{15}]\!]_{15}$
$[\![r]\!]_{64} \leftarrow \mathrm{LUT}^{15\to 64}([\![d]\!]_{15})$ ▷ $R[0] = 2$; otherwise $R[d] \approx 2^{16}/d$
$\{[\![u_i]\!]_{64}\}_{i=1}^n \leftarrow \{[\![e_i]\!]_{64} \cdot [\![r]\!]_{64}\}_{i=1}^n$ ▷ batched Beaver multiplication
for $i = 1, \dots, n$ **do**
  $[\![p_i]\!]_{64} \leftarrow m_i \cdot \mathrm{ltrunc}_8([\![u_i]\!]_{64})$ ▷ nominal $1/256$ output grid; no re-normalization
end for
return $\{[\![p_i]\!]_{64}\}_{i=1}^n$

simulation within rounding noise. A correction exists: adding one public unit to one party's truncated share per position, gated by the public mask. It is free at runtime and re-centers the row sum at one. We deliberately do not deploy it, for two reasons. First, it buys no accuracy: the bias is input-independent in expectation, the downstream LayerNorm and classifier rescale the attenuated probabilities, and the accuracies of Table 2 are achieved with the bias in place. Second, and central to this paper's point, the correction is not a runtime toggle: it changes the operator's semantics, so enabling it post hoc would confront the deployed checkpoint with a fresh train–deploy mismatch. A deployment that requires sum-one probabilities should adopt the corrected contract from the start and train and certify against it, just as with any other contract change.

Algorithm 1 summarizes the complete flow, and Table 8 lists the frozen production domains.

C.2 GELU PROTOCOL

We evaluate the exact error-function form of GELU, $g(x) = \frac{x}{2}\big(1 + \mathrm{erf}(x/\sqrt{2})\big)$, through one public 256-entry table per FFN layer. All rounding below is half-away-from-zero, matching the C++ `llround` convention.

Activation slotization. The FFN1 output is not independently re-shared before slotization: the raw accumulator P_l at $2f{=}24$ fractional precision (excluding bias) keeps the production truncation in the same chain. Let β_l be the SERVER-held bias code on the $f{=}12$ grid and $2^{E_{a,l}}$ the public dyadic input scale. Define $s_l = E_{a,l} + 12, \quad M_l = 2^{s_l}, \quad R_l = P_l + 2^{12}\beta_l, \quad D_l = 2^{12} M_l$. The protocol applies an arithmetic right shift by 12 to each share, the SERVER adds β_l, both shares are shifted by s_l with one SERVER-side compensation, and the result is reduced modulo 2^8. Writing $q_l = \lfloor R_l / D_l \rfloor$ and $t_l = R_l - q_l D_l \in [0, D_l)$, and assuming the low bits of the additive shares are uniform, the pre-reduction index satisfies $\Pr[\tilde{q}_l = q_l] = \frac{D_l - 1 - t_l}{D_l}, \quad \Pr[\tilde{q}_l = q_l + 1] = \frac{t_l + 1}{D_l}$. No public Bernoulli bit is sampled; the apparent stochasticity is induced by the low bits of the shares. The index is $\mathrm{idx} = \tilde{q}_l \bmod 2^8$, decoded as a signed two's-complement code $q_x \in [-128, 127]$

Table 8: Production protocol domains and bitwidths for the protocol-aligned Softmax.

Component	Production choice	Role
Arithmetic ring	$\mathbb{Z}_{2^{64}}$	Beaver multiplication and truncation
Fixed-point scale	$f = 12$	Cross-layer numerical interface
Score input	f=12 in $\mathbb{Z}_{2^{64}}$; unmasked range $[-16, 15.875)$	Direct protocol input; no independent 16-bit score ring
Score slots	8-bit unsigned; $\Delta_s = 2^{-3}$	$q_i \in \{0, \dots, 255\}$
Public keep mask	$m_i \in \{0, 1\}$	Padded entries never win the maximum; zero exponential and zero output
Max comparison	9-bit domain	One headroom bit; $n - 1$ secure comparisons
Exp LUT	8→64	256-entry table $E[j] = \mathrm{round}(2^8 \exp(\ell(j)))$
Reciprocal LUT	15→64 for $n \le 128$; 16-bit fallback for $129 \le n \le 255$	$R[0] = 2$ and rounded reciprocal values
Normalization	64-bit Beaver multiplication plus sharewise shift by 8	Nominal $1/256$ probability grid; no re-normalization
Probability output	$1/256$ grid	Masked entries are exactly zero; at most one-ULP downward borrow

Algorithm 2 Per-layer GELU grid lookup (FFN1–GELU–FFN2 chain)

Require: Shared FFN1 accumulator P_ℓ ($2f$=24 fractional precision, bias excluded); SERVER-held bias code β_ℓ on the f=12 grid; public per-layer exponents $E_{a,\ell}$, $E^{\mathrm{out}}_{a,\ell}$; frozen public table T_ℓ (256 entries)

Ensure: Shared GELU output $[\![g]\!]_{64}$ on the exact f=12 grid that FFN2 consumes

1: Both parties shift their shares of P_ℓ right by 12 ▷ the production $2f \to f$ truncation stays in the chain
2: SERVER adds β_ℓ to its share ▷ bias folded in locally
3: Both parties shift right by $s_\ell = E_{a,\ell} + 12$; SERVER adds one compensation; reduce modulo 2^8 ▷ slotization; ± 1 stochastic rounding from share low bits
4: $[\![g]\!]_{64} \leftarrow \mathrm{LUT}^{8\to 64}(\mathrm{idx}; T_\ell)$ ▷ payload pre-encoded at $2^{E^{\mathrm{out}}_{a,\ell}+12}$
5: **return** $[\![g]\!]_{64}$ ▷ consumed by FFN2 as-is; no truncation or ring conversion

with grid value $x_{\mathrm{grid}} = q_x 2^{E_{a,l}}$. This is a ring reduction, not saturation ($127 + 1 \equiv -128$); the signed interpretation applies only after reconstruction. Valid configurations obey $E_{a,l} \le 0$ and $1 \le E_{a,l} + 12 \le 12$ and are rejected before table generation otherwise.

Table construction and FFN integration. For each layer the table is populated directly from erf-GELU: $q_y = \mathrm{clamp}\Big(\mathrm{round}\Big(g\big(q_x 2^{E_{a,l}}\big)\ 2^{-E^{\mathrm{out}}_{a,l}}\Big), -128, 127\Big)$, $T_l[\mathrm{idx}] = \mathrm{Enc}_{64}\Big(q_y\, 2^{E^{\mathrm{out}}_{a,l}+12}\Big)$, where Enc_{64} denotes two's-complement encoding in $\mathbb{Z}_{2^{64}}$ and the output exponent $E^{\mathrm{out}}_{a,l}$ is the coarsest dyadic grid that keeps all 256 outputs inside the signed 8-bit range. The clamp is a table-generation guard only: a nonzero clamp count rejects the calibration configuration. Because the payload already carries the factor $2^{E^{\mathrm{out}}_{a,l}+12}$, the LUT output is an arithmetic share on the exact f=12 grid FFN2 consumes: no truncation, ring extension, or wrap correction is inserted at the table boundary, and FFN2 evaluates directly as $H_{2,l} = \mathrm{ltrunc}_{12}(G_l W_{2,l}) + b_{2,l}$.

Algorithm 2 collects the per-layer flow, and Table 9 lists the frozen parameters.

C.3 LAYERNORM AND RANGE GUARDS

Per-layer stabilization offset. The adjusted variance $\bar{\sigma}^2 = \sigma^2 + \varepsilon$ must be encoded on the variance scale s^{var}_ℓ of layer ℓ. Since ε is a fixed real constant while s^{var}_ℓ varies across layers, a single global

Table 9: Frozen GELU contract parameters (per FFN layer ℓ).

Component	Production choice	Constraint / role
Input scale	dyadic $2^{E_{a,\ell}}$, $E_{a,\ell} \in [-11, 0]$	index grid step $\Delta = 2^{E_{a,\ell}}$
Index	8-bit signed $q_x \in [-128, 127]$	256 entries/layer; ring reduction, not saturation
Chain	FFN1 $2f \to f$ truncation folded in	no independent re-share before slotization
Payload	$\mathrm{Enc}_{64}(q_y\, 2^{E^{\mathrm{out}}_{a,\ell}+12})$, q_y int8	clamp count > 0 rejects the configuration
Output	share on the f=12 grid in $\mathbb{Z}_{2^{64}}$	feeds FFN2 directly; zero boundary conversion

integer offset would implement a *different* effective ε at every layer. We freeze $c_\ell = \lfloor \varepsilon / s_\ell^{\mathrm{var}} \rceil$ per layer together with that layer's table and certified input domain, so every layer realizes the same real-valued ε (e.g. $c_\ell \approx 3146$ at $E_\ell^{\mathrm{var}} = -6$ versus 1 at $E_\ell^{\mathrm{var}} = 0$).

Zero-leakage exponent handling. The binary-floating index splits the variance into an exponent and a mantissa. A plain MSB-extraction design would open the exponent and leak the order of magnitude of every row's variance ($\approx \log_2 24$ bits per row across 25 LayerNorm instances). The production variant never opens it: the normalization factor 2^{24-e} is itself read from a small public power-of-two table addressed by the secret index, and the scaling is applied with one shared multiplication, adding one round per LayerNorm.

Public affine parameters. The affine parameters (γ, β) are public constants of the deployment contract. Applying them to shares is a local computation that introduces no new messages: γ scales each share directly, and β is added to one designated share. Relative to a fully shared affine, this removes one secure multiplication per LayerNorm element; β could equally well remain shared at no cost.

Algorithm 3 collects the per-row flow, and Table 10 lists the frozen parameters.

Optional runtime range guard. Deployment is guard-free under the semi-honest model: the domain constraint is internalized by training and enforced by the offline certification of Section 3.4. As an optional hardening we also implemented an FSS range guard that replaces online OT comparisons with an FSS carry chain and a single aggregate abort (fail-closed, zero per-item leakage). The guard checks the adjusted-variance domain of every LayerNorm row (25 instances $\times$ 128 rows $=$ 3,200 rows per inference at n=128); measured at 2.66 ms per row, this extrapolates to $\sim$8.5 s online and $\sim$173 MB communication on the full model, versus 181.5 s and 3.38 GB for the naive online-OT guard. Enabled end-to-end, the guard brings the full model to $\sim$13.3 s (4.8 s base + $\sim$8.5 s guard) and $\sim$857 MB (684 + 173) online; its offline key generation is $\sim$1.9 h and 35.6 GB.

Algorithm 3 Per-row LayerNorm with binary-floating rsqrt index

Require: Shared row on the f=12 grid $[\![x_1]\!]_{64}, \ldots, [\![x_d]\!]_{64}$, $d = 768$; per-layer offset c_ℓ and variance exponent E_ℓ^{var}; frozen per-layer rsqrt table T_ℓ (12$\to$64, 4096 entries); public affine codes γ, β

Ensure: Shared LayerNorm output row on the f=12 grid

1: $\mu \leftarrow \frac{1}{d}\sum_i x_i$; $\quad v_i \leftarrow x_i - \mu$ ▷ shared arithmetic
2: $S \leftarrow \frac{1}{d}\sum_i v_i^2$; $\quad$ SERVER adds c_ℓ ▷ Beaver squares; adjusted variance on the layer scale
3: require $S < 2^{24} - c_\ell - 1024$ ▷ certified offline; optional fail-closed guard below
4: $e \leftarrow$ MSB position of S, kept secret, $e \in \{0, \ldots, 23\}$ ▷ FSS comparisons; the exponent is never opened
5: $w \leftarrow 2^{24-e}$; $\quad \hat{s} \leftarrow w \cdot S$ ▷ power-of-two table by secret index + one Beaver multiply
6: $\mathrm{idx} \leftarrow 128\, e + \mathrm{mantissa}_7(\hat{s})$ ▷ 12-bit index; local carry correction
7: $[\![t]\!]_{64} \leftarrow \mathrm{LUT}^{12\to 64}(\mathrm{idx};\, T_\ell)$ ▷ payload $= 2^{12}\sqrt{768}\,/\,(2^{E_\ell^{\mathrm{var}}}\sqrt{\hat{v}})$, $\hat{v} = (128{+}m)\,2^{e-7}$
8: $[\![y_i]\!]_{64} \leftarrow \gamma_i \cdot (v_i \cdot t) + \beta_i$ for all i ▷ Beaver multiplies + public affine
9: **return** $\{[\![y_i]\!]_{64}\}_{i=1}^{d}$

Table 10: Frozen LayerNorm contract parameters (per layer ℓ).

Component	Choice	Role
Adj. variance	24-bit, $S < 2^{24} - c_\ell - 1024$	input domain; guard
Stab. offset	$c_\ell = \lfloor \varepsilon / s_\ell^{\mathrm{var}} \rceil$, $E_\ell^{\mathrm{var}} \in [-7, 0]$	same real ε
BF index	12-bit: exp. $\times$ 7-bit mantissa	$e \in [0, 23]$, $m \in [0, 127]$
Exp. handling	2^{24-e} by secret index + 1 Beaver mul	1 round; exp. unopened
rsqrt LUT	per-layer, $12 \to 64$, 4096 entries	payload pre-scaled by $2^{E_\ell^{\mathrm{var}}}$, $\sqrt{768}$, f=12
Output	f=12 share; Beaver mul + affine	no rescaling

D SAME CORE COUNTS CHECK

SIGMA's S-CPU numbers are measured with 4 threads while our main tables use 112 hardware threads. Table 11 re-measures our operators at `OMP_NUM_THREADS=4`: the Softmax and GELU advantages persist, so they are protocol-level, not a thread-count artifact. Communication is independent of thread count.

Table 11: Operator time at an equal thread count (4 threads, LAN). Our GELU row uses the LUTSS evaluation path (zero-AES dot-product), which is the faster of our two evaluation paths at equal cores.

Operator	Online time (s)		Ratio
	Ours	SIGMA S-CPU	
Softmax	1.40	4.59	3×
GELU	0.64	3.71	5.8×
LayerNorm	0.72	0.70	parity

E PREPROCESSING COST

Table 12 reports full-model key-generation costs. The pruned Softmax configuration is measured directly at 1,152 rows per layer. Pruning reduces offline communication by exactly 25%, while key-generation traffic remains 815.9 KB per row; the measured time reduction slightly exceeds 25% because per-layer key generation carries a fixed overhead. The production dealerless offline pipeline, using a Ferret stream for full-model key generation and preprocessing, takes 9.4 min on LAN, produces 16.3 GB of stored material for one party, and reaches a peak RSS of 208 MB, with no malformed keys observed.

Key-generation communication and stored material are distinct. At 815.9 KB per row, the unpruned model's 144×128 Softmax rows require approximately 15 GB of key-generation traffic, and Table 12 reports the corresponding LAN and WAN times. The stored Softmax keys are an order of magnitude smaller, approximately 1.5 GB by structural count, and form part of the 16.3 GB single-party total above, which is otherwise dominated by linear-layer Beaver material; the 15 GB generation traffic is transferred during key generation but never stored. Within the stored material, public lookup-table contents are constructed once and reused across inferences, whereas masks, FSS lookup and comparison keys, and Beaver triples, including linear-layer Beaver material, are consumed per inference.

Table 12: Offline key-generation cost, full BERT-base model (n=128). Times are medians of three measured runs, except LayerNorm times (∼), which are composed from measured sub-component costs; key material (∼) is a structural computation, not measured.

Operator	LAN time (s)	WAN time (s)	Key material
Softmax (dense, 144 heads)	356	3,220	∼1.5 GB
Softmax (pruned, 108 heads)	255	2,379	∼1.1 GB
GELU	117	792	∼453 MB
LayerNorm (no guard)	∼69	∼268	∼214 MB

F EXACT VALUES FOR LUT DOMAIN COMPRESSION

Section 4.4 and Figure 2 plot the cost–accuracy trade-off across the lookup index width; Table 13 reports the exact measurements at the two endpoints of that sweep (batch size 4,096, LAN, medians of two runs). Three observations complement the figure. First, the $85\times/159\times/128\times$ reductions quoted in the main text are the ratios of these rows. Second, the cost growth is explained by the table size: 2^{16} entries cost $256\times$ the key material of 2^8, and peak key-generation memory reaches 8.3 GB—impractical for a single operator without a trusted dealer. Third, the online columns are nearly flat (0.29 vs. 0.41 ms and 34 vs. 36 B per lookup): the benefit of domain compression is realized almost entirely in preprocessing, not in online cost.

Table 13: Cost of Softmax LUT-domain compression. The compressed lookup is the deployed design, while the direct lookup uses the wider score domain. Batch size is 4,096.

Configuration	Index width (bits)	Entries (2^b)	Keygen (ms/LUT)	Peak memory (MB/batch)	Key size (B/LUT)	Online time (ms/query)	Online comm. (B/query)
Compressed (deployed)	8	256	0.048	52	64	0.29	34
Direct (wide-domain)	16	65,536	4.10	8,274	8,224	0.41	36
Direct / Compressed	–	256×	85×	159×	128×	1.4×	1.06×

G DOMAIN PENALTY STRENGTH

Figure 3 sweeps the hinge weight λ of the domain-violation loss on QNLI. Stronger penalties suppress out-of-domain positions during training, but accuracy peaks at $\lambda = 0.02$ and drops noticeably beyond it; we therefore deploy $\lambda = 0.02$ for QNLI and MRPC and $\lambda = 0.01$ for SST-2. The red curve counts positions that hit the penalty during training (a penalty-hit proxy), not post-training certification violations; $\lambda = 0.02$ attains the highest mean accuracy in this sweep while keeping penalty hits low and is the value selected for QNLI, not a statistically established optimum.

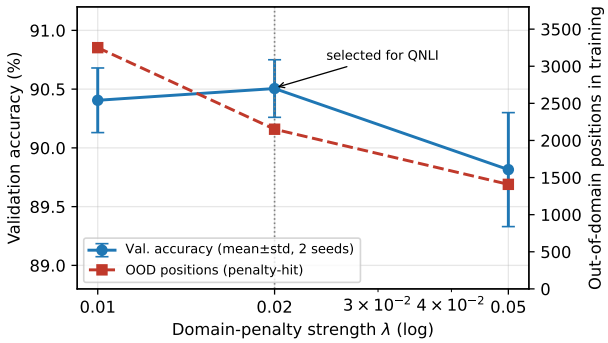

Figure 3: Domain-penalty strength sweep on QNLI (two seeds per point). Validation accuracy (blue, left axis) versus out-of-domain positions hit during training (red dashed, right axis). The dotted line marks the deployed $\lambda = 0.02$.

H INTEGER REFERENCE CHAIN VALIDATION

The reported accuracies are measured using a PyTorch fixed-point simulator that reproduces the protocol's integer semantics. We validate its agreement with secure two-party protocol execution by comparing final logits on one SST-2 input through the full 12-layer model (Table 14). Independent simulation runs exhibit logit differences of 0.035–0.046, providing a reference for rounding variability. The deployed configuration differs from the simulation by 0.035; variants with secret-shared LayerNorm affine parameters reach a maximum deviation of 0.072. All listed executions preserve the reference argmax prediction, and layer-by-layer replay passes all checks. These results support consistency between simulation and secure execution on the tested input, while broader deployment-accuracy validation requires additional inputs.

Table 14: Final logits from 12-layer secure inference and fixed-point simulation on one SST-2 input. All protocol variants preserve the reference prediction.

Configuration	Logits	Max $\lvert\Delta\rvert$
Fixed-point simulation (reference)	$[-4.2849,\ 3.8523]$	–
Secure inference: Beaver linear	$[-4.2593,\ 3.8220]$	0.030
Secure inference: known-weight linear (deployed)	$[-4.2634,\ 3.8171]$	0.035
With shared LN affine, run 1	$[-4.2632,\ 3.7805]$	0.072
With shared LN affine, run 2	$[-4.2468,\ 3.8105]$	0.042

Deployed inference uses public LN affine parameters (γ, β). The last two rows instead secret-share these parameters. Deviations are measured against the fixed-point reference.

I PERSEED-CERTIFICATION RECORD

Table 15 discloses every training seed under the pre-registered certification rule of Section 3.4: the scan uses training data only, the rescaling rule is applied at most once, and a candidate that still fails is rejected rather than further tuned. QNLI seed 0 required one application of the rule; seed 1 passed directly. QNLI seeds 2 and 3 are rejected and disclosed: seed 2 exceeds the pre-registered accuracy budget ($0.47 > 0.3$ points below the distilled student), and seed 3 exhibits a calibration blind spot (validation-split violation) that the training-data-only scan cannot see. We keep both failures in the record because they are what makes the certification an accept/reject test rather than a tuning loop.

Table 15: Per-seed certification outcomes under the pre-registered rule. Rejected seeds are disclosed with their failure reasons; the deployed accuracy averages certified seeds only (Table 2).

Dataset	Seed	Outcome	Deployed acc.
SST-2	0, 1, 2	certified (3/3)	91.02 ± 0.33
QNLI	0	certified after one repair	89.90
QNLI	1	certified, no repair	90.22
QNLI	2	rejected (accuracy budget)	–
QNLI	3	rejected (calibration blind spot)	–
MRPC	0, 1, 2	certified (3/3)	85.78 ± 0.74 / F1 89.75 ± 0.48

J DEPLOYMENT OVERHEADS AND GPU OUTLOOK

PROTOQUANT shrinks the input domains for nonlinear FSS and the amount of key material needed, but key generation is still a setup overhead: it does not depend on the input, yet it increases startup latency, storage, and communication, particularly for models with many nonlinear operators or following recalibration—costs common to FSS-based inference Gupta et al. (2024); Ma et al. (2026). On the computation side, our current implementation runs FSS operators on CPUs. GPUs can raise throughput by batching evaluations over tokens and operators, as FuseFSS shows Ma et al. (2026), but irregular table lookups and sequential dependencies during tree traversal constrain parallelism, while data movement and kernel-launch overheads eat into the gains, so the overall end-to-end improvement remains unclear. Straightforward ways to mitigate both issues include parallelizing key generation, streaming keys to cap peak memory usage, caching setup artifacts when allowed by the security model, and co-designing GPU kernels optimized for table lookup.