Journal on
Artificial Intelligence

Tech Science Press

https://doi.org/10.32604/jai.2026.092496

ARTICLE

Accuracy-Invisible Safety Degradation in a Compressed On-Device Prompt-Injection Classifier

S. M. Nihal Ahmed*, Afrim Hossen Khan, Sabikun Nahar Sinthia and Md. Rashedul Alam

Department of Software Engineering, Daffodil International University, Dhaka, Bangladesh

*Corresponding Author: S. M. Nihal Ahmed. Email: sm.nihal4@gmail.com

Received: 07 September 2026; Accepted: 27 October 2026; Published: 13 November 2026

ABSTRACT: Smartphone-integrated large language model (LLM) assistants that consume untrusted content—emails, web pages, documents—are exposed to prompt injection. A purpose-built guard placed ahead of the assistant is the standard mitigation, but on-device use requires compression, and compression is not known to be safety-neutral: prior work reports accuracy-invisible degradation in generative alignment under quantization, and guardrails are rarely re-evaluated after compression. This paper compresses Qwen2.5-1.5B-Instruct into a binary prompt-injection classifier and scores every stage on three pooled in-distribution corpora and the held-out Benchmarking Indirect Prompt Injection Attacks (BIPIA) corpus. After an 84.8% parameter cut, the final fake-quantized model retains 98.4% of the uncompressed pooled-validation F1 score (the harmonic mean of precision and recall), 0.9726 vs. 0.9888; packed to a 202 MB GPT-Generated Unified Format (GGUF) file in 4-bit `Q4_K_M` form it scored 0.9470 (false-positive rate 0.063), a loss we traced to an export fault that truncated the file's vocabulary: the same file read with the training tokenizer scores 0.9713, and a corrected re-export scores 0.9710. Held-out BIPIA F1 rises (0.2298→0.4924) while Youden's J collapses (0.0974→0.0174) and the area under the receiver operating characteristic curve (ROC-AUC) falls from 0.590 to 0.500: compression removes the little ranking ability the uncompressed model had, rather than merely relocating its threshold, and no operating point on the compressed model separates the classes. That ability was already thin—document length alone separates them at ROC-AUC 0.659—and compression removes it entirely. Applying each step alone to the uncompressed model shows that structured pruning removes it (ROC-AUC 0.513; 0.545 after recovery), while recovery training (0.595) and 4-bit quantization (0.613) do not. A never-compressed 0.5B model from the same family has higher in-distribution F1 (0.9863), and a fine-tuned Llama-3.2-1B is equally weak on BIPIA (ROC-AUC 0.537). On the same phone the corrected 205 MB file is 1.9× smaller than a 0.5B model packed the same way, loads 3.1× faster and uses 41%–47% less browser memory, but is 18% slower per document; PIGuard, an encoder guard, scores F1 0.78 zero-shot on our validation set. No adaptive, multilingual or obfuscated attack is evaluated. The GGUF is downloaded from a local area network (LAN) website and run in-browser as WebAssembly (WASM). Classification of the first-shipped file re-run on a 4 GB Realme C25 (800 documents of at most 504 tokens) matches a native llama.cpp run of the same file on 99.0% of rows, at a median 0.15 s (validation) to 2.9 s (BIPIA); iOS/WebKit fails to allocate.

KEYWORDS: Prompt injection; LLM guardrails; model compression; quantization; on-device inference; small language models

1 Introduction

Smartphone assistants increasingly act on content the user did not write: forwarded emails, scraped web pages, shared documents, and other text an on-device or cloud-backed large language model (LLM) ingests as part of answering a request. Prompt injection exploits exactly this channel, embedding adversarial

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

instructions in that untrusted content so that the assistant executes the attacker's intent rather than the user's. Consumer-facing systems are already shown to be vulnerable at scale: undefended attack success exceeds 90% across six consumer LLMs, 27 curricula vitae (CVs), and four scenarios [1], and the same study documents an "aware-but-bypassed" failure mode in which a model can recognize an injection as suspicious yet still act on it. Detecting an injection is therefore not the same problem as neutralizing its effect on the output, which motivates a dedicated guard placed ahead of the primary assistant rather than relying on the assistant's own judgment.

Placing that guard on-device rather than behind a cloud classifier helps latency, offline availability and privacy, since input never leaves the phone before it is screened. It is also constrained: handset memory, energy and thermal budgets make a full-scale LLM guard impractical, so the guard must be compressed. The systems literature treats compression—pruning, quantization, distillation—as largely solved for general-purpose small language models [2,3]. It does not treat as routine whether compression preserves the behavior a *security* classifier exists for. Rath and Maliakkal [4] report that post-trained chat models can revert toward pre-alignment behavior under low-precision quantization while task accuracy and perplexity stay flat, and Dong et al. [5] observe that published guardrail numbers are overwhelmingly measured at full precision. A guard whose in-distribution accuracy survives compression is not thereby shown to keep its skill against attacks it was never trained on.

The gap is sharper for prompt injection. Existing mobile-agent security work evaluates defenses at full precision on server hardware [6] or treats cross-deployment generalization, rather than compression, as the open question [7,8]. No study reviewed here reports stage-wise in-distribution and held-out accuracy of a compressed prompt-injection guard together with a timed in-browser serving path on consumer handset hardware.

This paper addresses that gap for a single pipeline. Starting from Qwen2.5-1.5B-Instruct, a compact decoder chosen for its favorable accuracy-to-parameter ratio in the 1–2B on-device band, the guard model is compressed through structured feed-forward network (FFN) pruning, residual-stream layer pruning, two alternative post-pruning recovery strategies, vocabulary pruning, and 4-bit activation-aware weight quantization (AWQ), with a light low-rank adaptation (LoRA) or quantized LoRA (QLoRA) recovery pass after every structural change. Every stage is benchmarked independently on three pooled in-distribution prompt-injection corpora and on a fourth corpus, derived from the Benchmarking Indirect Prompt Injection Attacks (BIPIA) benchmark, that is held out in full and contains structurally distinct, document-embedded injections. The packed GPT-Generated Unified Format (GGUF) file is served from a page on the local network: the handset downloads the file and runs it in-browser. The handset classifies sampled validation and BIPIA documents end to end, and its predictions are compared row by row with a native llama.cpp run of the same file; timing also uses an Apple M1 laptop, and iOS/WebKit. This design is intended to answer three research questions (RQs):

- **RQ1:** To what extent do iterative pruning and quantization, each followed by recovery fine-tuning, affect the in-distribution and held-out prompt injection detection accuracy of a compact (1–2B parameter) guard model, relative to its unoptimized counterpart?
- **RQ2:** Does the compression pipeline introduce the type of accuracy-invisible safety degradation documented in the general LLM compression literature [4], and if so, at which stage of the prune–quantize–fine-tune sequence does this degradation predominantly emerge?
- **RQ3:** What load time, classification latency, and memory footprint does the packed GGUF achieve when the handset downloads the model over a local network and classifies documents in-browser (WebAssembly, WASM), do its predictions match a native run of the same file, and does the same build initialize under iOS/WebKit?

Section 2 reviews related work and states the research gap, Section 3 the threat model, corpora and nine-stage pipeline, Section 4 the configuration, and Section 5 the stage-by-stage results and on-device performance. Section 6 answers RQ1–RQ3 and states what the results do and do not license.

2 Literature Review

2.1 The Prompt Injection Threat Landscape

Prompt injection—adversarial instructions embedded in a user prompt or in external content the model subsequently consumes—is among the most consequential vulnerabilities of large language models (LLMs) deployed in consumer, operating-system (OS)-level assistants. Milani et al. [1] formalize the indirect case, in which malicious instructions are hidden inside uploaded documents: across six consumer LLMs, 27 CVs, and four scenarios, undefended attack success exceeds 90%, and a model's ability to *detect* an injection does not reliably translate into neutralizing its influence on the output (an "aware-but-bypassed" failure mode). A guard that only flags suspicious input is therefore not equivalent to one that actually blocks that input's influence on the downstream response.

Three systematic reviews spanning 2022–2025 converge on the same trajectory. Geng et al. [9] (a Kitchenham-method review of 128 studies) document the shift from direct injections toward multimodal and automated attacks, again with >90% success on unprotected systems. Gulyamov et al. [10] (January 2023–October 2025, mapped to the Open Worldwide Application Security Project (OWASP) Top 10 for LLM applications) rank prompt injection as the leading threat for agentic and retrieval-augmented deployments and note that even current detectors fail against adaptive, iteratively refined exploits. Mathew [11] offer a comparable attack taxonomy and stress that most defenses are evaluated on static benchmarks rather than against attackers who adapt to a deployed filter—the relevant standard for any guard's held-out evaluation. Derner et al. [12] organize prompt risks along the confidentiality–integrity–availability (CIA) triad by target (user, model, application) and mechanism; an input-side injection guard maps onto their integrity-preserving controls. Yao et al. [13] divide the broader LLM security literature into constructive, adversarial, and vulnerability-exposing ("ugly") threads and observe that the last has outgrown defensive work, which argues for deployable, lightweight defenses over further threat catalogs. Across these reviews, static pattern filters consistently underperform learned classifiers against paraphrased or obfuscated attacks.

2.2 Detection-Based Defenses and Guardrail Architectures

Pre-inference classification of contaminated prompts is the dominant defense class, and the one relevant here. Lan et al. [14] combine banned-term matching, embedding similarity, and a fine-tuned Bidirectional Encoder Representations from Transformers (BERT) model into a real-time detector that is substantially cheaper than an LLM-as-judge. The same architectural bet is supported outside the injection literature: Jamal et al. [15] find that compact, task-tuned DistilBERT/RoBERTa phishing-spam classifiers (IPSDM) outperform generic transformer baselines. Phishing and prompt injection share the underlying structure of adversarially crafted natural language aimed at a downstream system, so a small, specialized classifier is a motivated design choice rather than an efficiency-only shortcut.

Dong et al. [5] characterize the guardrail literature as a black-box, post-hoc paradigm—Llama Guard, NVIDIA NeMo Guardrails, Guardrails AI—built to enforce hallucination, fairness, privacy, robustness, and toxicity constraints. Existing systems are predominantly loosely coupled neural-symbolic stacks: a small classifier sitting in front of a much larger generator. That size asymmetry is treated in the literature as a requirement rather than an implementation detail, which is why this work uses a purpose-built detector rather than a general-purpose LLM judge. Dong et al. [5] also observe that almost all published guardrail

numbers are measured at full precision, prior to any pruning or quantization—a gap this study is designed to address.

2.3 Small Models and Compression

Zhu et al. [3] treat quantization, pruning, and distillation as complementary and flag an evaluation gap: compression papers report leaderboard accuracy far more often than safety-adjacent or out-of-distribution (OOD) behavior. Dikici and Bilgin [2] (a systematic review of 68 studies, 2022–2025, following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses [PRISMA] 2020 guideline) report that parameter-efficient fine-tuning (PEFT) methods such as LoRA yield 2–10× speedups at 95%–99% of baseline accuracy, and that privacy and security increasingly decide the case for on-device SLMs over cloud LLMs. Recovery is itself design-sensitive: Gu et al. [16] argue that uniform LoRA ranks waste budget on unimportant layers and propose a layer-adaptive La-LoRA. This pipeline uses uniform ranks (Section 4); La-LoRA is not run.

On-device deployment forces both pruning and quantization. Ratsapa et al. [17] cast layer pruning as a combinatorial search and find that early and late blocks resist pruning, with the middle third removed most often—a prior that this pipeline's residual-stream importance scores contradict (Section 3.9). Lin et al. [18] introduce activation-aware weight quantization (AWQ): roughly 1% of channels are salient, identifiable from activation statistics, and protected by per-channel scaling without backpropagation. The prevailing edge recipe is structured pruning followed by precision reduction, each with a light recovery step—the sequence this study adopts.

2.4 The Compression–Security Trade-Off

Compression is not safety-neutral. Beyond the alignment collapse of Rath and Maliakkal [4] and the observation of Dong et al. [5] that guardrails are rarely re-evaluated after compression, Das et al. [19] find that published mitigations are validated under one fixed configuration and seldom re-checked after compression, adapter merging, or further fine-tuning. A guard whose job is security classification cannot be certified on in-distribution accuracy alone: held-out injection behavior must be measured at each stage.

2.5 On-Device Deployment and Mobile Agent Security

Qu et al. [20] survey mobile edge intelligence for LLMs across on-device, edge-assisted, and cloud tiers: on-device inference offers the strongest privacy and latency but the tightest budgets, so compression is mandatory. Friha et al. [21] and Jiang et al. [22] find robustness to adversarial input treated as secondary to latency and throughput, and security-focused edge-LLM evaluation thin; the reviews of Wang et al. [23], Chen et al. [24], and Zheng et al. [25] do not treat adversarial robustness as a first-class constraint.

Mobile-agent security work is sparser still. Alzahrani [6] proposes an injection-resilient framework but evaluates it only on server hardware. Deng et al. [7] identify prompt injection as the dominant vector in the first of four agent-threat gaps, and He et al. [8] conclude that defenses validated on one agent in a controlled setting routinely fail in a different deployment context, which applies directly to a smartphone assistant under real memory, latency, and connectivity constraints.

2.6 Adjacent Threat Models

Backdoor attacks and prompt injection both make a model obey an attacker, but they enter at different points. A backdoor is planted through the model's training data or weights and fires only when a trigger appears at inference [26]; prompt injection needs no access to training and attacks a benign model through

content it later consumes. Concrete backdoor techniques make this difference precise. Poisoning as few as one hundred instruction-tuning examples ties a trigger phrase to a chosen output [27]; the trigger can be the task instruction itself rather than an input token [28]; and a syntactic template rather than a rare word makes the trigger invisible to token-level filters [29]. Compression is a training-time surface in its own right: a model can be built to behave benignly in full precision and maliciously only after quantization [30], and the same holds for the zero-shot quantization schemes widely used to deploy LLMs [31]. Our pipeline contains several such events: the recovery fine-tunes, distillation, AWQ with QLoRA, and GGUF packing. We train every stage ourselves from a public checkpoint, so we do not test a quantization-conditioned backdoor, and the results here should not be read as evidence that the compressed guard is free of one.

Two further lines of work show how the same channels can be used in either direction. Wang et al. [32] make an LLM forget by editing what the retrieval layer supplies rather than by changing weights. That is the same lever indirect prompt injection pulls: control over retrieved content is enough to change behaviour. A guard placed on that channel, as ours is, therefore sits on the path both uses share. Ye et al. [33] show that model extraction, membership inference and model inversion can be run without any training data, using generative models to synthesise queries against a black-box target. A small guard that runs on the user's device is an easy target for such an attack: an extracted substitute would let an attacker search for injections that evade the guard offline, which is the adaptive setting our threat model (Section 3.2) excludes. None of these attacks is evaluated here.

2.7 Research Gap and Research Questions

Three observations follow from this literature. First, small, purpose-built classifiers are an established first line of defense against prompt injection, and keeping them light relative to the generator they protect is treated as a design requirement—a choice corroborated by parallel work in phishing detection [5,14,15]. Second, structured pruning, activation-aware quantization and recovery fine-tuning can cut size and latency while retaining most in-distribution accuracy [3,16–18]. Third, compression—quantization in particular—can degrade safety-relevant behavior in ways standard accuracy misses [4,19], a risk the edge and mobile literature has not treated as first-class [21,22].

These threads have not been examined jointly for a dedicated prompt-injection guard that is pruned, recovered, quantized, and recovered again, then scored on both in-distribution and held-out attacks, with a timed in-browser serving path on handset hardware. Alzahrani [6] evaluates only at full precision; Rath and Maliakkal [4] study general-purpose chat models rather than an iteratively compressed security classifier; and Deng et al. [7] and He et al. [8] flag cross-deployment generalization, not compression, as the open question. No reviewed work reports stage-wise in-distribution and held-out scores for a compressed guard together with a timed in-browser serving path on consumer handset hardware. These observations motivate RQ1–RQ3 (Section 1).

3 Methodology

This section specifies a compact binary classifier intended to intercept prompt-injection attempts before they reach a primary on-device language model. The remaining subsections cover the task and base architecture (Section 3.1), the training and evaluation corpora (Section 3.3), the compression pipeline (Sections 3.4–3.14), the evaluation protocol (Section 3.15), on-device serving (Section 3.16), and the scope of the experiments (Section 3.17). Headline result tables appear in Section 5; corpus, split and hyperparameter tables appear in Sections 3.3 and 4.

3.1 Problem Formulation and Base Architecture

Detection is binary sequence classification. An input x is mapped to logits $z = (z_0, z_1)$ over benign ($y = 0$) and injection ($y = 1$). The injection probability is $\hat{y} = \mathrm{softmax}(z)_1$, trained with binary cross-entropy

$$\mathcal{L}_{\mathrm{cls}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\right]. \tag{1}$$

A two-way softmax (rather than a single sigmoid) keeps the head compatible with temperature-scaled distillation in Section 3.10.

The base model is Qwen2.5-1.5B-Instruct [34] (Apache 2.0 license; a favorable accuracy-to-parameter ratio within the 1–2B on-device band). A decoder-only architecture is used for three reasons: (i) the assistants being guarded are themselves decoder models; (ii) the compression–safety literature this study builds on [4] studies decoders, so a decoder-based guard is the directly comparable case; and (iii) the intra-family baselines of Section 3.11 would be architecturally confounded by an encoder such as ModernBERT [35], which shares neither tokenizer nor pre-training lineage with the Qwen2.5 family. The language-modelling head is replaced with a randomly initialized two-class head that pools the last non-padding token. Classification is therefore a single forward pass—no decode loop and no malformed-output risk—which also makes the model a simpler object for the pruning and quantization stages that follow.

3.2 Threat Model

Defender and asset. A smartphone assistant reads content it did not write—an email, a web page, a shared file—and the guard labels that text benign or injection before the assistant sees it. The guard is a 202–205 MB file (202 MB as first shipped, 205 MB corrected; Section 5.9) the handset downloads and runs locally.

Attacker. The attacker controls the content of a document the assistant will read and nothing else: not the user's prompt, the assistant's system prompt, the guard's weights, or the handset. The goal is to make the assistant follow an instruction hidden in that content, the indirect injection of Yi et al. [36]. A user attacking their own assistant directly is a different threat and is not modelled.

What is evaluated. Only a non-adaptive attacker. Every injection comes from a fixed corpus written without knowledge of this guard and is scored once. This is the weaker setting; the reviews of Section 2 note that detectors that do well on static benchmarks fail against attackers who adapt.

What is not evaluated. (i) An adaptive attacker. Because the guard ships as a downloadable file, the realistic adaptive attacker is white-box with unlimited offline queries, and can optimize a payload against the very weights on the phone; a query-limited black-box attacker is the easier case. We ran no gradient-based or search-based attack and measured no query-budget curve. An adaptive evaluation is informative only for a guard that first passes the non-adaptive one, and the held-out results (Section 5.10) show this guard does not, so none of our results speaks to resistance. (ii) Agent-level attack success. InjecAgent [37] places attacker instructions in tool responses and measures whether a tool-using agent acts on them. Scoring the guard on those tool-response texts would extend the BIPIA evaluation here, but we did not run it, and nothing in this paper measures the attack-success rate of an assistant with the guard placed in front.

Claims. The conclusions are about non-adaptive, document-embedded injections in the corpora of Section 3.3. They do not extend to adaptive attackers, to agent behaviour, or to the language and obfuscation settings noted in Section 3.17.

3.3 Dataset Curation and Preprocessing

Training and in-distribution evaluation draw on three public corpora—xTRam1, S-Labs, and NeurAlchemy—pooled after message-digest 5 (MD5) deduplication of normalized text and partitioned with group-aware splitting so that near-duplicates cannot cross the train/validation boundary. A fourth corpus, derived from BIPIA, is held out in full (Table 1).

Table 1: Corpora used for training, in-distribution benchmarking, and held-out evaluation. Repository addresses are those of the cited sources.

Corpus	Source	Role	Distinguishing Characteristics
xTRam1	Li et al. [38]	Train/benchmark	Synthetic, GLAN-style generation; broad attack-category coverage
S-Labs	S-Labs [39]	Train/benchmark	Hard-negative examples; balanced validation and test splits
NeurAlchemy	NeurAlchemy [40]	Train/benchmark	~16,900 examples; group-aware, leakage-checked
BIPIA-derived	Alamsabi et al. [41]	Held-out (never trained on)	Indirect, document-embedded injection; structurally distinct attack surface; benign half generated with GPT-4o-mini

xTRam1 is synthetic, with broad attack-category coverage; Li et al. [38] is cited for the Generalized Instruction Tuning (GLAN) generation procedure used to build it, not as a corpus paper. S-Labs [39] supplies hard negatives—benign text that only superficially resembles an attack—with balanced validation and test splits. NeurAlchemy [40] contributes about 16,900 group-aware, leakage-checked examples. The BIPIA-derived corpus [41] contains indirect, document-embedded injections and is never used for training or validation.

BIPIA is an attack-success benchmark, not a detection benchmark [36]: it publishes malicious examples only, so a binary detector needs a constructed negative class. The corpus scored here is the derived set of Alamsabi et al., which pairs 35,000 malicious BIPIA examples with 35,000 benign examples generated with GPT-4o-mini; we sample 1500 of each. Each example is rendered as one `Context:` string and labelled by the presence of an injected instruction.

This construction has a consequence. Positives are real BIPIA documents with an instruction inserted; negatives are model-generated text, so the classes differ both in the instruction and in whether the document is real. Negatives are not the positives with the instruction removed: a 120-character interior probe matches a positive on 2.8% of documents, the same as against a shuffled positive set. Positives average 409.7 tokens against 291.4 for negatives. A detector could therefore separate the classes on realism or length without detecting injection; Section 5.10 shows the compressed models do not exploit this (ROC-AUC $\approx$ 0.50), and that the uncompressed model reaches only 0.590.

Two things qualify BIPIA as held-out. First, the four-stage recovery (Section 3.10) applies BIPIA-*like* wrappers to pooled training text only; no BIPIA example is used, but the attack *family* is no longer unseen after Strategy 1. Second, vocabulary pruning (Section 3.12) retains the token IDs needed to encode BIPIA, a representability constraint rather than supervision. At a 512-token truncation the training and validation corpora alone leave 37,120 embedding rows; including BIPIA raises this to 44,871, so BIPIA contributes 7751

rows (17.3%, 22.7 MB of FP16 embedding). Under the restricted vocabulary 6.1% of BIPIA token occurrences would lack a row, at 6.12% for positives and 6.16% for negatives, so the choice is not label-correlated. Repeating vocabulary pruning and recovery without BIPIA in the active set (one seed, same pre-pruning checkpoint) gives held-out ROC-AUC 0.5017 against 0.5092 with it; the 0.0075 difference is within the sampling error of 3000 documents and both values are chance.

Two further candidate corpora were dropped: one lists BIPIA-derived and xTRam1 among its own sources, which would have leaked the held-out set; the other is an undocumented twelve-source aggregate whose overlap could not be verified.

3.4 Compression Pipeline Overview

After task-adaptive fine-tuning, the model is compressed in the stages of Fig. 1. Width is reduced first (structured FFN pruning, then two FFN-only rounds), then depth (layer pruning), then recovery is compared (four-stage focal-loss vs. offline distillation), then the embedding table and weight precision are reduced. Each stage is benchmarked before the next, isolating the cost required by RQ2. The figure includes the two recovery branches and the in-browser GGUF serving path.

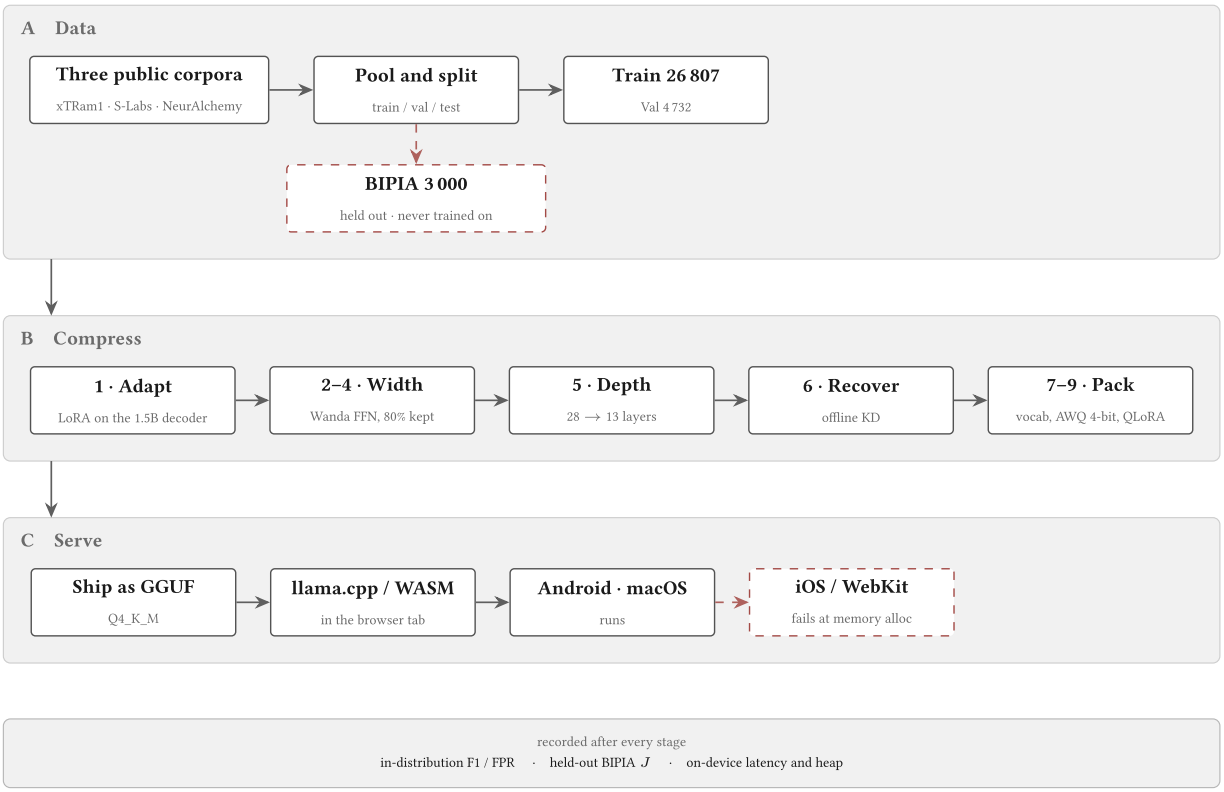


Figure 1: Compression and serving pipeline. (**A**) Three public corpora are pooled and split into train (26,807) and validation (4732); BIPIA (3000) is held out in full and never trained on. (**B**) The 1.5B decoder is LoRA-adapted, then compressed in width (Wanda FFN sweep, 80% kept, two further −5% cuts), depth (28→13 layers by residual-stream score), recovery (offline knowledge distillation, KD), and packing (vocabulary pruning, AWQ 4-bit, QLoRA). (**C**) The `Q4_K_M` GGUF is served by llama.cpp/WASM in a browser tab; Chromium on Android and macOS runs, while iOS/WebKit fails at `WebAssembly.Memory` allocation. In-distribution F1 and false-positive rate (FPR), and held-out BIPIA Youden's J are recorded after every stage; the on-device readout is end-to-end classification (predictions, latency, browser-reported memory), not a native process monitor.

3.5 Initial Fine-Tuning of the Base Model

Qwen2.5-1.5B-Instruct is adapted with LoRA [42]. For a frozen $W_0 \in \mathbb{R}^{d\times k}$,

$$\begin{aligned} W &= W_0 + \Delta W = W_0 + BA, \\ B &\in \mathbb{R}^{d\times r}, \quad A \in \mathbb{R}^{r\times k}, \quad r \ll \min(d,k). \end{aligned} \tag{2}$$

Adapters sit on q, k, v, o, gate, up, and down projections of every layer; the classification head is trained jointly. LoRA is used so every later stage fits on one 16 GB graphics processing unit (GPU). Adapters are merged into W_0 before pruning so Wanda calibration (Section 3.6) sees a single dense matrix per layer.

3.6 Structured Pruning: Initial Sparsity Sweep

FFN intermediate neurons are scored with Wanda [43],

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2, \tag{3}$$

where X_j is the calibration activation of input feature j (training-corpus subset; one forward pass, no Hessian). Magnitude pruning degrades sharply on LLMs at moderate sparsity; SparseGPT is more accurate per layer but substantially more expensive. Unstructured or $N{:}M$ Wanda yields no speedup without sparse kernels, which the target mobile platform lacks, so scores are applied *structurally*: for each intermediate neuron u,

$$I_u = \sum_{(i,j)\in u} S_{ij}, \tag{4}$$

and whole units are removed. For target width-sparsity $s \in \{0.60, 0.70, 0.80, 0.90\}$,

$$s = 1 - \frac{|\Theta_{\text{kept}}|}{|\Theta_{\text{total}}|}, \tag{5}$$

where Θ counts prunable FFN-intermediate units, not total parameters. The four variants are recovered and scored independently (Section 3.7).

3.7 Recovery Fine-Tuning and Sparsity-Level Selection

Each sweep variant is recovered with a fresh LoRA of the form (2) on the pooled training corpus, then scored on the three in-distribution corpora and on held-out BIPIA. The parent of later stages is chosen on pooled-validation F1 vs. parameter count. BIPIA is reported but is not the selection criterion: RQ2 is precisely the claim that held-out safety can move independently of in-distribution accuracy, and the sweep-stage BIPIA F1 of the 80% variant is not reproduced on the canonical re-evaluation (Section 5.3). Unselected sparsities are ablations only.

3.8 Additional FFN-Only Pruning Rounds

From the recovered 80% checkpoint, two further rounds each drop 5% of surviving FFN width (no extra attention-head pruning). Importance is a column-wise Wanda variant on the SwiGLU (Swish Gated Linear Unit) down-projection,

$$I_u^{\text{FFN}} = \|X_u\|_{\text{rms}} \cdot \|W_{\cdot,u}^{\text{down}}\|_1, \tag{6}$$

not the unit-aggregated score (4). The lowest-scoring neurons are removed by shrinking gate/up output dimensions and the down input dimension (dense, not sparse, matrices). Each round is followed by the same LoRA recovery as Section 3.7. The second-round checkpoint enters Section 3.9.

3.9 Layer Pruning

Depth is then reduced. Layer ℓ is scored by mean residual displacement on the same 256-example calibration set,

$$I_\ell = \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{x \in \mathcal{D}_{\text{cal}}} \left\| h_\ell^{\text{out}}(x) - h_\ell^{\text{in}}(x) \right\|_2. \tag{7}$$

Low I_ℓ means the block barely moves the residual stream; residual addition lets it be deleted without changing neighbouring tensor shapes. The 15 lowest-scoring blocks are removed (including layer 0, which this criterion ranks lowest, contrary to the early/late resistance prior of Ratsapa et al. [17]). The 13-layer model enters Section 3.10.

3.10 Post-Pruning Recovery Strategies

Two recoveries are run on the same 13-layer backbone.

Strategy 1: four-stage focal-loss recovery.

Each stage has its own data, objective, and LoRA budget; adapters are merged between stages; early stopping uses patience 3–4 epochs.

Stage 1 (recall). Heavy augmentation over seven surfaces: prefix and suffix injections; document-embedded wrappers (email, code, hypertext markup language [HTML], JavaScript Object Notation [JSON], table, chat); obfuscation (double spacing, zero-width characters, leetspeak, character alternation, spaced keywords); multilingual prefixes across ten languages, in the spirit of the cross-lingual robustness argument of Conneau et al. [44]; role-play/persona hijack; and delimiter and markdown abuse. Wrappers follow a BIPIA-like taxonomy but are applied only to pooled training sources; class-0 counterparts use the same wrappers on non-adversarial text so that document structure is not itself an injection cue. Selection score $0.65 \cdot \text{Recall} + 0.35 \cdot \text{F1}$. Focal loss [45] with $\alpha = 0.85$, $\gamma = 2.5$; label smoothing 0.03.

Stage 2 (precision). Clean pooled data; score $0.65 \cdot \text{Precision} + 0.35 \cdot \text{F1}$; $\alpha = 0.55$, $\gamma = 1.5$; smoothing 0.08.

Stage 3 (F1). Same wrapper taxonomy as Stage 1 at lower multiplicity, still on pooled sources only; standard F1; $\alpha = 0.70$, $\gamma = 2.0$.

Stage 4 (stabilization). Clean data; score $0.50 \cdot \text{Accuracy} + 0.50 \cdot \text{F1}$; all seven projections ($r = 8$, $\alpha_{\text{LoRA}} = 16$), dropout 0.25, smoothing 0.10.

Stages 1 and 3 expose the student to document-embedded structure, so BIPIA movement under Strategy 1 is jointly attributable to depth reduction and this augmentation. Strategy 2, trained only on clean pooled data, is the cleaner attribution path.

Strategy 2: offline knowledge distillation.

Teacher: the uncompressed LoRA-tuned 1.5B model of Section 3.5. Teacher logits over the full training set are written to disk first, so teacher and student are never co-resident on GPU.

$$\begin{aligned} \mathcal{L}_{\text{distill}} = {} & \alpha_{\text{KD}}\, \mathcal{L}_{\text{CE}}(\hat{y}_s,\, y) \\ & + (1 - \alpha_{\text{KD}})\, T^2\, \text{KL}\Big(\sigma(z_s/T) \,\Big\|\, \sigma(z_t/T)\Big), \end{aligned} \tag{8}$$

with the T^2 rescale standard in temperature-scaled knowledge distillation [46]. Sub-stage 1: heavy teacher (α_{KD} = 0.30, T = 3.0, all seven projections, r = 32). Sub-stage 2: lighter teacher (α_{KD} = 0.70, T = 2.0, all seven projections, r = 16).

The two checkpoints are compared in Section 5.5. The model carried forward is chosen on in-distribution FPR and on the absence of BIPIA-family augmentation, not on BIPIA F1 alone (Section 3.15).

3.11 Intra-Family Benchmarking against Qwen2.5 Reference Models

Before further compression, the compressed line is benchmarked against two uncompressed intra-family reference models trained independently (Table 2), holding architecture, pre-training, tokenizer, and vocabulary fixed. (i) The Section 3.5 1.5B checkpoint: uncompressed task ceiling and accumulated pruning cost. (ii) Qwen2.5-0.5B-Instruct fine-tuned on the same corpus (494M): the native small-model alternative at a comparable parameter budget, with none of the compression pipeline. Both see the same suite: per-corpus and pooled validation, plus held-out BIPIA.

Table 2: Reference models used for intra-family comparison in Section 3.11. Both share the Qwen2.5 architecture, tokenizer, and pre-training lineage with the base model, ensuring that compression level is the sole variable distinguishing them from the compressed guard model at this stage.

Model	Parameters	Approach	Role
Qwen2.5-1.5B-Instruct (fine-tuned)	1562M	LoRA fine-tuning	Uncompressed upper bound
Qwen2.5-0.5B-Instruct (fine-tuned)	494M	LoRA fine-tuning	Small-model intra-family baseline

3.12 Vocabulary Pruning

The Qwen2.5 embedding table is 151,936 × 1536 and is conventionally stored in FP16, so its memory cost is independent of transformer compression. For this English binary task most rows are unused. The active set is

$$\mathcal{V}_{\text{active}} = \Big\{ v \in \mathcal{V} \,\Big|\, \sum_{x \in \mathcal{D}_{\text{all}}} \mathbf{1}\big[v \in \text{tok}(x)\big] > 0 \Big\}, \tag{9}$$

where $\mathcal{D}_{\text{all}} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{BIPIA}}$. BIPIA is included so held-out text remains encodable; the effect of excluding it is reported in Section 3.3. Unused rows are dropped. Qwen2.5 byte-pair encoding (BPE) merge rules index the original vocabulary, so the tokenizer is not rebuilt; IDs are remapped to a contiguous embedding space in the collator, and the model config (vocab size, pad ID) is updated to match.

Applied once, to the distillation checkpoint from Section 3.10. Reindexing shifts embedding geometry, so a light LoRA recovery on the remapped pooled corpus follows. Both the raw pruned model and the recovered model are benchmarked.

Export.

A GGUF file carries its own tokenizer, and the browser and llama.cpp tokenize with it, not with the training tokenizer. The converter builds that vocabulary from the first N = 44,871 entries of the original numbering, so a token whose original id is $\geq N$ is absent and is spelled letter by letter. Our first export saved the full tokenizer beside the pruned weights and so shipped such a file (Section 5.9). The corrected export rebuilds the vocabulary in the pruned order, adds the 4037 merge-ancestor pieces that byte-pair encoding needs (their embedding rows are copies of the filler row, which no token reaches), and makes the id map the identity, so the file emits pruned ids directly. It is checked against the training tokenizer on the full validation and BIPIA sets before use.

3.13 Post-Training Quantization

The vocabulary-pruned model is quantized to 4-bit weights with AWQ [18]. Per-channel scales protect the ~1% of channels that activations identify as salient,

$$s^* = \arg\min_s \left\| Q\big(W\,\mathrm{diag}(s)\big)\,\mathrm{diag}(s)^{-1}X - WX \right\|, \tag{10}$$

$$\begin{aligned} Q(w) &= \Delta \cdot \mathrm{round}(w/\Delta), \\ \Delta &= \max(|w|)/(2^{N-1}-1). \end{aligned} \tag{11}$$

No backprop or reconstruction calibration is required. Offline evaluation uses fake quantization (weights rounded to the 4-bit grid but stored/computed in FP16), which isolates the accuracy cost without packed kernels; on-disk size and latency change only at GGUF deployment (Section 3.16). Embedding rows stay FP16; only the seven linear projections are quantized.

3.14 Final Fine-Tuning

A QLoRA-style pass [47] trains fresh adapters on the frozen AWQ backbone $\hat{W}$,

$$y = \hat{W}x + B'A'x. \tag{12}$$

The original QLoRA pairs adapters with 4-bit NormalFloat (NF4)/bitsandbytes; here the same frozen-backbone recipe sits on the AWQ weights of Section 3.13. This is the final offline checkpoint. Reported parameter counts are backbone counts; adapter parameters are negligible relative to the stage-wise sizes reported in Section 5.

3.15 Evaluation Protocol and Metrics

Every stage is scored on each in-distribution corpus, the pooled validation set, and held-out BIPIA. Injection is the positive class:

$$\mathrm{Acc} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}, \tag{13}$$

$$\mathrm{Prec} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \tag{14}$$

$$\mathrm{Rec} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \tag{15}$$

$$\mathrm{F1} = \frac{2\cdot\mathrm{Prec}\cdot\mathrm{Rec}}{\mathrm{Prec}+\mathrm{Rec}}, \tag{16}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}, \tag{17}$$

$$J = \mathrm{Rec} - \mathrm{FPR}. \tag{18}$$

Precision, recall, and F1 are reported for the injection class. FPR is treated as a primary metric, since over-blocking benign input is the dominant failure mode in deployment. Youden's J is reported on BIPIA because, unlike F1, it cannot be inflated merely by shifting the decision threshold toward the positive class.

Reported BIPIA accuracy, recall, and FPR are consistent with a balanced set of 1500 injection and 1500 benign examples ($n = 3000$), for which $\mathrm{Acc} = \frac{1}{2}(1+J)$. A rise in BIPIA F1 accompanied by Acc ≈ 0.5 and

$J \approx 0$ is therefore not a gain in held-out skill: on a balanced set it means the score no longer ranks the two classes above chance, so no decision threshold can separate them. We therefore report ROC-AUC and the true-positive rate (TPR) at a fixed FPR alongside J, because J alone cannot distinguish a badly placed threshold from an absent ranking signal.

Inputs are truncated at 512 tokens, which cuts 31.5% of BIPIA positives but only 13.8% of negatives; the effect on held-out scores is analysed in Section 5.10.

Accuracy-invisible safety degradation is defined here as a material drop in BIPIA J (or BIPIA accuracy) without a commensurate drop in pooled-validation F1. A drop in BIPIA F1 is neither necessary nor sufficient for this definition: F1 can rise while J falls. This is the operational criterion used to answer RQ2. All offline training was run on the Kaggle T4 GPUs described in Section 4; deployment timing follows the browser wall-clock protocol of Section 3.16.

On-disk sizes are $|\Theta| \times b/2^{20}$ bytes (mebibytes, labeled MB). FP16 uses $b = 2$. Final 4-bit estimates keep embeddings in FP16 ($b = 2$) and linear weights at 4-bit ($b = 1/2$), matching fake-AWQ and the intended GGUF path.

3.16 On-Device Deployment

The checkpoint of Section 3.14 is exported to GGUF (`Q4_K_M`, 202 MB; the file as first shipped, whose vocabulary fault and corrected 205 MB re-export are described in Section 5.9) and served through llama.cpp compiled to WebAssembly (wllama) from a page on the local network over Hypertext Transfer Protocol Secure (HTTPS) with cross-origin isolation, so the WASM runtime can use threads. The handset runs no native application: it opens the page, downloads the GGUF and runs inference on-device. Latency and memory are read from JavaScript (JS) in the page, not from an Android or iOS process monitor; the page records `measureUserAgentSpecificMemory()`, which includes WASM linear memory, and the Chromium JS heap, which does not.

Classification follows the deployed path: the page tokenizes each document, maps the token ids through the vocabulary-pruning table, obtains the sequence embedding from llama.cpp (embedding mode, last-token pooling) and applies the two-class head in JS; the head is exported with the checkpoint and is not in the GGUF. A document is injection when the injection logit exceeds the benign logit. This wllama build accepts no explicit last-token pooling option, so pooling is taken from the GGUF metadata; the parity check below confirms it matches.

The build fixes the physical batch at 512 tokens, so a document must fit in 504 tokens. Of the 4732 validation and 3000 BIPIA documents, 101 and 984 are longer and are excluded. From the remainder the page classifies 400 validation and 400 BIPIA documents (label-stratified, seed 0) and compares each prediction with a native llama.cpp run of the same file (Secure Hash Algorithm 256 [SHA-256] digest recorded by the server). After ten documents a gate stops the run if agreement falls below 80%, which catches a wrong pooling or normalization setting. Rows logged more than once are counted once, using the first measurement. The file was the one first shipped. The thread count of this run was not logged reliably (the final page session reports two; an earlier smoke test showed four). The full-set comparison of Section 5.9 uses the same file with native llama.cpp on the Mac, without the length restriction. The comparison with a directly trained 0.5B model (Section 5.14) uses the same page, context (512) and thread count, with one change: because compute cost does not depend on the weight values, the 0.5B model is packed through the same export path with an untrained two-class head and is timed only; its predictions are never scored or quoted.

An earlier timing protocol generated one token for each of $N = 50$ short prompts and did not classify; it supplies the load times and host comparison of Section 5.13 and its latency is superseded.

The two Chromium hosts are (i) a Realme C25 (MediaTek Helio G70, 2× Cortex-A75 at 2.0 GHz plus 6× Cortex-A55 at 1.7 GHz; Mali-G52; 4 GB random-access memory [RAM]) and (ii) a MacBook with an M1 processor and 8 GB RAM, which ran only the earlier timing page. On iOS/WebKit the identical build fails at `WebAssembly.Memory` allocation (Section 5.13); every iOS browser must use WebKit, so this is a platform result, not a browser choice.

3.17 Experimental Scope

The full pipeline is run on Qwen2.5-1.5B-Instruct only. The sparsity sweep (Sections 3.6 and 3.7) is deliberately cheap—four sparsity levels, one recovery, and one benchmark per level—so that the remaining compute budget (Sections 3.8–3.14) can be spent following that single line to its conclusion. The contribution is therefore the sequential width–depth–vocabulary–precision pipeline and its measured behavior, not a cross-architecture compression bake-off. The reference models introduced in Section 3.11 are independently fine-tuned and are not themselves pruned, vocabulary-reduced, or quantized. All stages use seed 42; results are single-run point estimates, and no statistical-significance claim is made.

Language and obfuscation.

The held-out evaluation is not multilingual and does not test obfuscated attacks. BIPIA's source tasks are English-language datasets (news articles, e-mails, Python questions) [36], and an audit of the sampled data (script analysis plus automatic language identification on a random sample of 1500 documents of at least 40 characters per corpus) confirms how little else it contains. In BIPIA, 99.0% of the sampled documents are identified as English; no document is mostly non-Latin script; 7.8% of the 3000 contain a non-ASCII letter (9.7% of positives, 5.9% of negatives); 1.7% contain a base64-like run (2.4% and 1.0%); and four contain zero-width characters. The in-distribution corpora are also overwhelmingly English (roughly 97% of the sampled rows), but they carry a label-correlated minority: 0.9% of training documents (238 of 26,807) and 0.9% of validation documents (42 of 4732) are mostly non-Latin script, and 235 and 41 of them respectively are labelled injection (2.0% of injections against 0.02%–0.04% of benign documents). Non-Latin script is therefore a label cue in training and validation that BIPIA does not contain. Language identification on short or noisy text is unreliable, so the minority-language shares should be read as upper bounds. Multilingual prefixes and obfuscation also appear as synthetic training augmentation (Stage 1, Section 3.10), but they are never scored on held-out data, so nothing here measures how the guard behaves on non-English or obfuscated injections. The 512-token limit is a further restriction (Section 5.10).

4 Experimental Setup

This section reports the hardware environment, software stack, dataset statistics, and hyperparameter configurations used at every stage of the compression pipeline. All experiments were conducted on Kaggle-hosted infrastructure using two NVIDIA Tesla T4 GPUs (16 GB video RAM [VRAM] each, 32 GB total system RAM). Training at every stage used mixed-precision (FP16) arithmetic throughout. Browser GGUF timing (Section 5.13) uses a Realme C25 (Helio G70, 4 GB) and a MacBook M1 (8 GB), both in Chromium. Hyperparameter settings are collected in Table 3.

4.1 Software Environment

All experiments were implemented in Python using PyTorch as the primary deep-learning framework. Model loading, tokenization, and training orchestration used the Hugging Face `transformers` library, with parameter-efficient fine-tuning provided by the `peft` library. Dataset handling used the Hugging Face `datasets` library. Evaluation metrics were computed using `scikit-learn`. A fixed random seed of 42 was used at every stage for all framework-level random-number generators.

Table 3: Training, recovery, and AWQ hyperparameters on one page. The Sweep column is also used after each FFN-only round (Section 3.8). Training columns use cosine LR scheduling, FP16, and seed 42; the initial fine-tune uses AdamW (Hugging Face default) and gradient checkpointing. "All 7" denotes q, k, v, o, gate, up, and down. AWQ is fake quantization for offline accuracy only; embeddings stay FP16. Warmup is a step count in the Sweep column and a ratio elsewhere. Empty cells (—) are not applicable. KD is the recovery carried forward; four-stage is the ablation.

	Init	Sweep	Four-Stage (Ablation)				KD (Carried)		AWQ	QLoRA
Hyperparameter	**FT**	**Rec.**	**1**	**2**	**3**	**4**	**1**	**2**	**4-Bit**	**Rec.**
Objective	Val. F1	Val. F1	Recall	Prec.	F1	Acc+F1	KD+CE	KD+CE	fake-4b	Val. F1
Training data	Pooled	Pooled	Heavy aug	Clean	Doc-emb.	Clean	Pooled	Pooled	cal. 256	pooled
LoRA rank r	8	8	32	24	16	8	32	16	—	16
LoRA alpha	16	16	64	48	32	16	64	32	—	32
LoRA dropout	0.05	0.15	0.15	0.20	0.20	0.25	0.10	0.15	—	0.30
Target modules	All 7	All 7	All 7	All 7	All 7	All 7	All 7	All 7	All 7	All 7
Learning rate	2×10^{-4}	5×10^{-5}	1×10^{-4}	5×10^{-5}	3×10^{-5}	1×10^{-5}	5×10^{-5}	1×10^{-5}	—	1×10^{-4}
Warmup	None	300 st.	0.08	0.06	0.05	0.03	0.06	0.03	—	0.10
Epochs	3	8	12	12	10	8	15	10	—	5
Train batch	4	8	16	16	16	16	16	16	cal. 8	16
Grad. accum. (eff.)	4 (16)	1 (8)	2 (32)	2 (32)	2 (32)	2 (32)	2 (32)	2 (32)	—	2 (32)
Eval batch	8	16	32	32	32	32	32	32	—	16
Weight decay	0.0	0.01	0.03	0.06	0.05	0.08	0.03	0.05	—	0.10
Early stopping	—	3 ep.	3 ep.	4 ep.	3 ep.	3 ep.	4 ep.	3 ep.	—	—
Focal α/γ	—	—	0.85/2.5	0.55/1.5	0.70/2.0	0.60/1.0	—	—	—	—
Label smoothing	—	—	0.03	0.08	0.06	0.10	0.03	0.05	—	—
KD mix/T	—	—	—	—	—	—	0.70/0.30/3.0	0.30/0.70/2.0	—	—
Quant. bits/grp/α	—	—	—	—	—	—	—	—	4/128/0.5	—

4.2 Dataset Statistics

Table 4 summarizes the size and class composition of each data split used throughout the study. The pooled training and validation sets were produced by the group-aware deduplication procedure described in Section 3.3. The BIPIA-derived evaluation corpus was held out in full and never used for training or validation at any stage.

Table 4: Dataset split sizes used across all pipeline stages.

Split	Examples	Role	Used for
Train (pooled)	26,807	Training	Stages 1–9
Validation (pooled)	4732	Evaluation	All benchmark stages
xTRam1 subset	1521	Evaluation	Per-corpus benchmark
S-Labs subset	2270	Evaluation	Per-corpus benchmark
NeurAlchemy subset	941	Evaluation	Per-corpus benchmark
BIPIA (held-out)	3000	Evaluation	Held-out benchmark only

All input sequences were tokenized using the Qwen2.5 tokenizer with a maximum sequence length of 512 tokens; sequences exceeding this limit were truncated. Padding was applied dynamically per batch using `DataCollatorWithPadding`. After vocabulary pruning, the same tokenizer is retained and token identifiers are remapped to the reduced embedding index space in the collator, as described in Section 3.12.

4.3 Training and Quantization Hyperparameters

All LoRA, recovery, distillation, and AWQ settings are in Table 3. Adapters from the initial fine-tune are merged before Wanda calibration. Teacher logits for distillation were written over the full 26,807-example training set at batch size 32, so teacher and student were never co-resident on GPU. The four-stage procedure is the unused ablation; KD is carried forward.

4.4 Structured Pruning Sweep Configuration

The Wanda importance calibration used a fixed calibration set of 256 examples drawn randomly from the training corpus. Pruning was applied structurally to feed-forward intermediate neurons at four candidate sparsity levels: 60%, 70%, 80%, and 90%. Each pruned variant was recovered with a fresh LoRA fine-tune before benchmarking. Recovery hyperparameters are the Sweep column of Table 3.

4.5 FFN-Only Pruning Round Configuration

The two additional FFN-only pruning rounds described in Section 3.8 each removed a further 5% of the surviving feed-forward intermediate neurons, reducing the intermediate size from 1792 (post-80% sweep) to 1702 after round 1 and to 1617 after round 2. Each round used the column-wise importance score of Eq. (6) with a 256-example calibration set. Recovery fine-tuning after each round used the Sweep column of Table 3.

4.6 Layer Pruning Configuration

Layer importance was scored using the residual-stream criterion of Eq. (7) over the same 256-example calibration set. Fig. 2 reports the per-layer importance scores and the removal decision for all 28 layers of the model entering this stage. The 15 layers with the lowest importance scores were removed, leaving a 13-layer model.

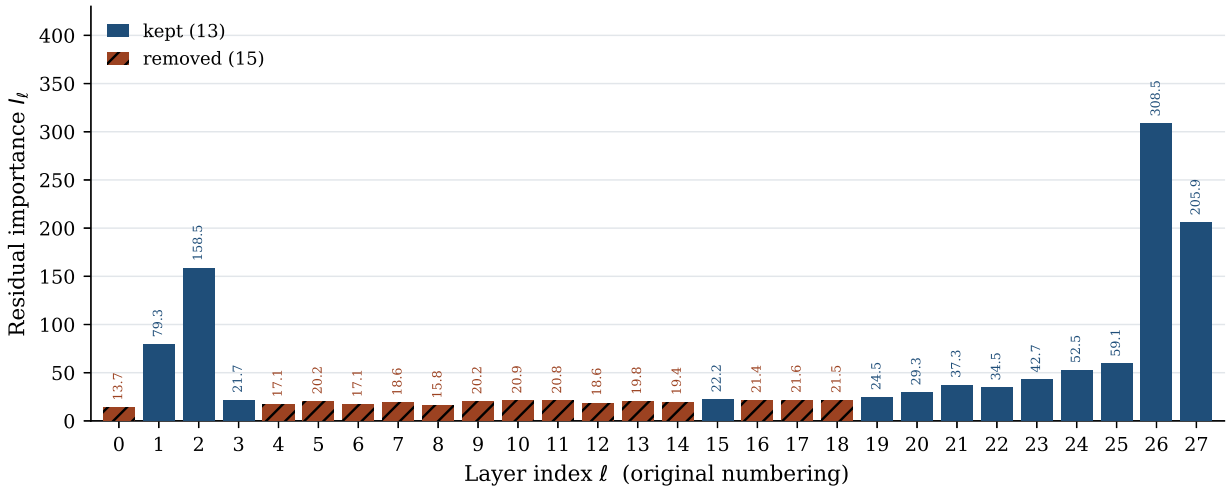


Figure 2: Per-layer residual-stream importance I_ℓ (Eq. (7)) and removal decisions. The model entering this stage had 28 layers, FFN intermediate size 1617, and 12 attention heads. Hatched bars are the 15 lowest-scoring layers and are removed; solid bars are the 13 layers that are kept and then renumbered contiguously. Layer 0 ranks lowest, contrary to the early/late resistance prior of Ratsapa et al. [17]. Scores on the bars are rounded to one decimal place.

The resulting compressed architecture contained 401,874,944 parameters, with a theoretical half-precision size of 766.52 MB. A uniform 4-bit packing of every parameter would occupy 191.63 MB; under the embedding-FP16/linear 4-bit integer (INT4) convention of Section 3.15 the footprint would be larger,

because the still-unpruned 151,936-row embedding table would remain in half precision. The hidden size (1536), FFN intermediate size (1617), and number of attention heads (12) were unchanged by layer pruning.

4.7 Offline Preparation Cost

Wall-clock times were not logged for the original runs, so Table 5 reports costs measured on one Tesla T4 when the stages were re-run for Section 5.11, and bounds the full pipeline with them. One epoch of recovery training over the 26,807 training examples took 96 min for the 1.5B model (distillation, batch 16) and 39 min for the 80%-pruned 619M model (sweep recovery, batch 8); peak GPU memory was 6.1 GB. Writing the teacher logits took 31 min, and the AWQ and Wanda calibration passes about 10 s each. Every later stage trains a model no larger than 619M parameters, so 39 min is an upper bound on its epoch. At the maximum epoch counts of Table 3, the carried path costs at most 46 T4-hours; the unselected sparsities and the four-stage ablation add at most 58 more. Early stopping makes the real cost lower: the distillation re-run stopped at epoch 6 of 15.

Table 5: Offline preparation cost on one Tesla T4. "Measured" rows are timed runs; "bound" rows multiply the maximum epochs of Table 3 by the measured epoch time of a model at least as large. The initial fine-tune is assumed to cost one 1.5B epoch per epoch.

Stage (Carried Path)	Basis	Epochs (Max)	T4-Hours
Teacher-logit generation (26,807 examples)	Measured	—	0.5
Initial LoRA fine-tune, 1.5B	Bound	3	4.8
80% sweep recovery, 619M	Bound	8	5.2
Two FFN-only rounds with recovery	Bound	16	10.4
Distillation recovery (two sub-stages)	Bound	25	16.3
Vocabulary-pruning recovery	Bound	8	5.2
AWQ calibration and QLoRA	Bound	5	3.3
Carried path, total	Bound		46
Unselected sparsities and four-stage ablation	Bound		58

5 Results

Fig. 3 is the pipeline in three panels. Each stage's per-corpus results are tabulated in its own subsection.

5.1 Initial Fine-Tuning of the Base Model

The uncompressed LoRA-tuned 1.5B classifier is the reference (Table 6). Pooled-validation F1 is 0.9888 at FPR 0.0078, so LoRA transfers the decoder to this task. Held-out BIPIA is conservative: F1 0.2298 and $J = 0.0974$, from low recall and low FPR. Later BIPIA-F1 increases must be read against that operating point: raising recall and FPR together can improve F1 while destroying J.

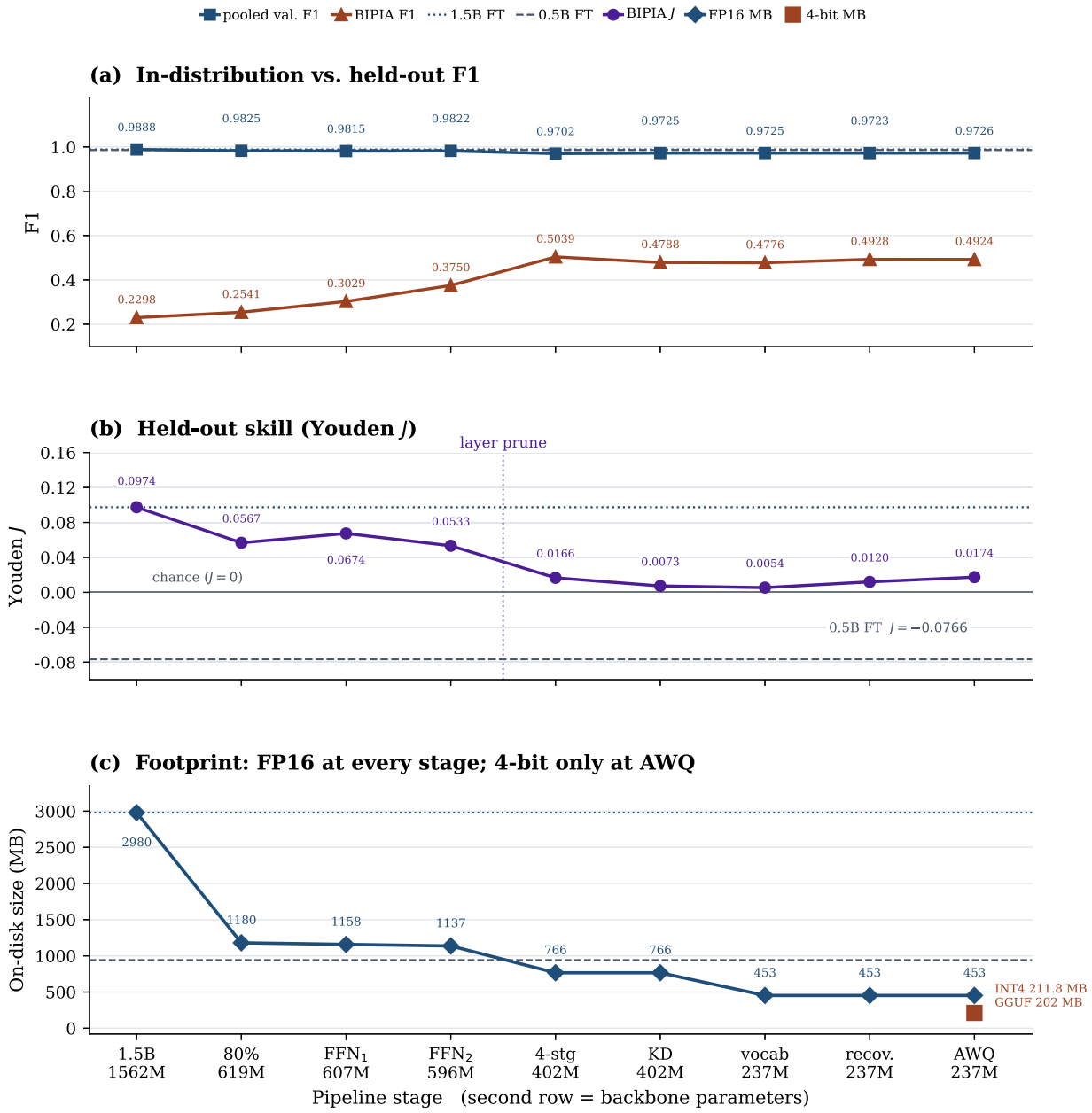


Figure 3: Stage-wise trajectory of the compressed classifier. (**a**) *In-distribution vs. held-out F1.* Pooled-validation F1 stays near the uncompressed 1.5B ceiling (dotted) and just below the fine-tuned 0.5B reference (dashed); held-out BIPIA F1 rises toward 0.5. (**b**) *Held-out skill (Youden J).* J collapses at layer pruning (vertical dotted marker) and does not recover through vocabulary reduction or fake-AWQ. Chance is $J = 0$; the 0.5B reference is $J = -0.0766$. (**c**) *Footprint: FP16 at every stage; 4-bit only at AWQ.* FP16 on-disk size is plotted at each tick; the theoretical embedding-FP16/linear-INT4 estimate (211.8 MB) and the packed `Q4_K_M` GGUF (202 MB as first shipped) appear only at the final AWQ stage. The 4-stg and KD ticks are two recoveries of the same 13-layer backbone, not successive compressions; KD is the checkpoint carried forward. The second x-axis row is backbone parameter count.

Table 6: Benchmark results for the LoRA-fine-tuned Qwen2.5-1.5B-Instruct baseline prior to any compression. Precision, recall, and F1 are for the injection class. J = Recall − FPR is reported for the held-out BIPIA corpus.

Dataset	Accuracy	Precision	Recall	F1	FPR	J	n
xTRam1	0.9961	0.9935	0.9935	0.9935	0.0028	—	1521
S-Labs	0.9925	0.9893	0.9941	0.9917	0.0088	—	2270
NeurAlchemy	0.9756	0.9874	0.9717	0.9795	0.0187	—	941
pooled_val	0.9903	0.9898	0.9878	0.9888	0.0078	—	4732
BIPIA	0.5487	0.7829	0.1347	0.2298	0.0373	0.0974	3000

5.2 Structured Pruning Sweep

Table 7 reports the four recovered widths. Pooled F1 stays above 0.978 at every level. 70% is best on pooled F1 (0.9859). The 80% checkpoint has 618.9M parameters, 115.6M fewer than the 70% checkpoint (734.5M) and 220.5 MB smaller, at a pooled-F1 cost of 0.0036. 90% is the first variant below 0.98 and is worse on every in-distribution corpus than 80%. Under Section 3.7, 80% is the parent of later stages: the most compact model with pooled F1 > 0.98.

Table 7: Results of the initial structured pruning sweep after recovery fine-tuning. F1 is reported for each validation corpus individually, for the pooled validation set, and for the held-out BIPIA corpus as a sweep-stage measurement only; the canonical BIPIA evaluation of the carried-forward 80% checkpoint is Table 8.

Sparsity	Params (M)	FP16 (MB)	xTRam1 F1	S-Labs F1	NeurAlc. F1	Pooled F1	BIPIA F1
60%	850.1	1621.4	0.9893	0.9854	0.9778	0.9842	0.3078
70%	734.5	1400.9	0.9883	0.9883	0.9794	0.9859	0.5126
80%	618.9	1180.4	0.9851	0.9854	0.9741	0.9823	0.5504
90%	503.3	959.9	0.9777	0.9830	0.9706	0.9784	0.4456

Sweep-stage BIPIA F1 is non-monotonic and is not used for selection. It peaks at 0.5504 for the 80% variant; the canonical re-evaluation in Section 5.3 is 0.2541. In-distribution F1 is stable across the two evaluations (0.9823 vs. 0.9825).

5.3 Selected 80% Checkpoint Carried Forward

The canonical 80% checkpoint (Table 8) is the parent of all later stages: 618.9M parameters, FFN width 1,792. Pooled F1 is 0.9825 (cost 0.0063 vs. the uncompressed model) after a 60.4% parameter cut. Held-out BIPIA F1 is 0.2541 and $J = 0.0567$: vs. the baseline, F1 rises while J falls and FPR nearly triples. This 0.2541—not the sweep-stage 0.5504—is the BIPIA number at the 80% tick of Fig. 3.

Table 8: Canonical benchmark results for the selected 80% structured-pruned checkpoint after recovery fine-tuning. This is the parent of the FFN-only pruning rounds. *J* is reported on BIPIA only.

Dataset	Accuracy	Precision	Recall	F1	FPR	*J*	n
xTRam1	0.9901	0.9767	0.9914	0.9840	0.0104	—	1521
S-Labs	0.9850	0.9815	0.9853	0.9834	0.0152	—	2270
NeurAlchemy	0.9756	0.9910	0.9682	0.9794	0.0133	—	941
pooled_val	0.9848	0.9829	0.9820	0.9825	0.0131	—	4732
BIPIA	0.5283	0.6071	0.1607	0.2541	0.1040	0.0567	3000

5.4 Additional FFN-Only Pruning Iterations

Two further 5% FFN-width cuts (no extra head pruning) barely move pooled F1 (0.9815 then 0.9822). BIPIA F1 rises (0.2541→0.3029→0.3750) while *J* stays flat then falls (0.0567, 0.0674, 0.0533), because recall and FPR rise together. The rounds train only on clean pooled data, so the shift is not confounded by document-embedded recovery. That is a mild accuracy-invisible change: in-distribution F1 is flat, held-out *J* does not improve. In-distribution and BIPIA grids are Tables 9 and 10.

Table 9: In-distribution results of the two additional FFN-only pruning iterations applied after the selected 80% structured-pruned checkpoint.

Checkpoint	FFN width	Params (M)	FP16 (MB)	xTRam1 F1	S-Labs F1	NeurAlc. F1	Pooled F1
80% recovered	1792	618.9	1180.4	0.9840	0.9834	0.9794	0.9825
FFN iter 1	1702	607.3	1158.3	0.9819	0.9839	0.9767	0.9815
FFN iter 2	1617	596.3	1137.3	0.9872	0.9825	0.9777	0.9822

Table 10: Held-out BIPIA performance across the recovered 80% checkpoint and the two additional FFN-only pruning iterations. *J* = Recall − FPR.

Checkpoint	Accuracy	Precision	Recall	F1	FPR	*J*
Recovered 80% checkpoint	0.5283	0.6071	0.1607	0.2541	0.1040	0.0567
FFN iter 1	0.5337	0.5996	0.2027	0.3029	0.1353	0.0674
FFN iter 2	0.5267	0.5518	0.2840	0.3750	0.2307	0.0533

5.5 Layer Pruning and Post-Pruning Recovery

Residual-stream pruning removes 15 of 28 layers. Two recoveries are compared on that fixed 13-layer backbone (Tables 11 and 12). Four-stage focal loss restores pooled F1 to 0.9702, and BIPIA F1 rises to 0.5039, but *J* = 0.0166 is near chance: recall and FPR both sit near 0.5. Stages 1 and 3 also use BIPIA-like wrappers, so that F1 jump cannot be attributed to depth cut alone.

Table 11: Architecture and headline benchmark results for the two post-layer-pruning recovery variants evaluated on the same 13-layer compressed architecture. BIPIA *J* is Youden's index.

Recovery	Layers	Params (M)	FP16 (MB)	Pooled F1	Pooled FPR	BIPIA F1	BIPIA *J*
Four-stage	13	401.9	766.5	0.9702	0.0265	0.5039	0.0166
Distillation	13	401.9	766.5	0.9725	0.0209	0.4788	0.0073

Table 12: Detailed benchmark metrics for the two post-layer-pruning recovery variants. *J* is reported on BIPIA only.

Checkpoint	Dataset	Acc.	Prec.	Rec.	F1	FPR	*J*	n
Four-stage focal-loss	xTRam1	0.9809	0.9504	0.9892	0.9694	0.0227	—	1521
Four-stage focal-loss	S-Labs	0.9727	0.9652	0.9746	0.9698	0.0289	—	2270
Four-stage focal-loss	NeurAlchemy	0.9660	0.9802	0.9629	0.9715	0.0293	—	941
Four-stage focal-loss	pooled_val	0.9740	0.9658	0.9747	0.9702	0.0265	—	4732
Four-stage focal-loss	BIPIA	0.5083	0.5085	0.4993	0.5039	0.4827	0.0166	3000
Offline distillation	xTRam1	0.9849	0.9623	0.9892	0.9756	0.0170	—	1521
Offline distillation	S-Labs	0.9744	0.9689	0.9746	0.9717	0.0257	—	2270
Offline distillation	NeurAlchemy	0.9660	0.9890	0.9541	0.9712	0.0160	—	941
Offline distillation	pooled_val	0.9761	0.9727	0.9722	0.9725	0.0209	—	4732
Offline distillation	BIPIA	0.5037	0.5041	0.4560	0.4788	0.4487	0.0073	3000

Distillation, trained only on clean pooled data, reaches pooled F1 0.9725 and $J = 0.0073$. It is the more conservative of the two on FPR. The BIPIA-F1 gap is not a skill gap—both *J* values are near zero. Distillation is carried forward, on pooled F1/FPR and on the absence of BIPIA-family augmentation, not on BIPIA F1. Four-stage remains the ablation.

5.6 Intra-Family Benchmarking against Qwen2.5 Reference Models

Table 13 compares the distilled 13-layer model with independently fine-tuned 1.5B and 0.5B models from the same family.

Table 13: Intra-family comparison between the uncompressed Qwen2.5-1.5B-Instruct baseline, the fine-tuned Qwen2.5-0.5B-Instruct baseline, and the compressed model after layer pruning and offline distillation recovery. BIPIA *J* for the 0.5B baseline is implied by its accuracy on the balanced BIPIA set ($J = 2 \cdot \text{Acc} - 1$). All models are Qwen2.5 variants.

Model	Params (M)	FP16 (MB)	Pooled F1	BIPIA Acc.	BIPIA F1	BIPIA *J*
1.5B-Instruct (FT)	1562.2	2979.63	0.9888	0.5487	0.2298	0.0974
0.5B-Instruct (FT)	494.0	942.29	0.9863	0.4617	0.3072	−0.0766
Compressed (distilled)	401.9	766.52	0.9725	0.5037	0.4788	0.0073

In-distribution, compressed pooled F1 0.9725 trails uncompressed 1.5B (0.9888) by 0.0163 and native 0.5B (0.9863) by 0.0138. At this budget, training 0.5B directly is the stronger in-distribution option and needs no compression pipeline; Section 5.14 compares what each costs to deploy.

Held-out ranking depends on the metric. Uncompressed 1.5B has the worst BIPIA F1 and the best J (0.0974). Native 0.5B is below chance on accuracy ($J = -0.0766$). The compressed model would look like a large held-out win on F1 alone; on J it is the closest of the three to a coin flip. Only 1.5B has a clear, if modest, held-out edge. Vocabulary pruning later moves the compressed line well below the 0.5B budget.

5.7 Vocabulary Pruning

Unused embedding rows are dropped (151,936→44,871 tokens), cutting the model to 237.4M parameters (453.24 MB FP16). The tokenizer is not rebuilt (Section 3.12). Pooled F1 is unchanged at the raw prune (0.9725) and 0.9723 after recovery. BIPIA J stays near chance. There is no new Section 3.15 degradation event. More than 70% of the embedding table can be removed at no in-distribution cost; held-out behavior remains in the near-chance regime set by layer pruning. The full grid is Table 14.

Table 14: Benchmark results before and after vocabulary pruning, and after the subsequent light recovery fine-tune. All three checkpoints share the same 13-layer compressed transformer backbone; the only structural change is the reduction of the embedding table from 151,936 to 44,871 token rows. J is reported on BIPIA only.

Checkpoint	Dataset	Acc.	Prec.	Rec.	F1	FPR	J	n
Pre-pruning distilled	pooled_val	0.9761	0.9727	0.9722	0.9725	0.0209	—	4732
	BIPIA	0.5037	0.5041	0.4560	0.4788	0.4487	0.0073	3000
Vocab pruned	xTRam1	0.9849	0.9623	0.9892	0.9756	0.0170	—	1521
	S-Labs	0.9744	0.9689	0.9746	0.9717	0.0257	—	2270
	NeurAlchemy	0.9660	0.9890	0.9541	0.9712	0.0160	—	941
	pooled_val	0.9761	0.9727	0.9722	0.9725	0.0209	—	4732
	BIPIA	0.5027	0.5029	0.4547	0.4776	0.4493	0.0054	3000
Vocab pruned + recovery	xTRam1	0.9822	0.9544	0.9892	0.9715	0.0208	—	1521
	S-Labs	0.9740	0.9653	0.9775	0.9713	0.0289	—	2270
	NeurAlchemy	0.9702	0.9891	0.9611	0.9749	0.0160	—	941
	pooled_val	0.9759	0.9691	0.9757	0.9723	0.0239	—	4732
	BIPIA	0.5060	0.5063	0.4800	0.4928	0.4680	0.0120	3000

5.8 Post-Training Quantization and Final Fine-Tuning

Fake-AWQ plus QLoRA recovery gives Table 15. Pooled F1 is 0.9726 vs. 0.9723 pre-AWQ; BIPIA J moves from 0.0120 to 0.0174. Neither change is material.

This is the stage at which Rath and Maliakkal [4] would predict quantization-induced alignment collapse. It does not appear. Held-out J has already collapsed before AWQ, and every post-pruning checkpoint we re-measured sits at ROC-AUC ≈ 0.50 (Section 5.10). This pipeline contains no unpruned-but-quantized control of its own; Section 5.11 applies AWQ to the unpruned baseline and finds no loss of ranking ability there either.

Table 15: Benchmark results for the vocabulary-pruned recovery checkpoint (pre-AWQ) and the final fake-quantized AWQ recovery checkpoint. The theoretical INT4 size of 211.8 MB is the embedding-FP16/linear-INT4 deployment estimate.

Checkpoint	Dataset	Acc.	Prec.	Rec.	F1	FPR	*J*	n
Vocab + recovery (pre-AWQ)	pooled_val	0.9759	0.9691	0.9757	0.9723	0.0239	—	4732
	BIPIA	0.5060	0.5063	0.4800	0.4928	0.4680	0.0120	3000
Final model (AWQ + recovery)	xTRam1	0.9855	0.9644	0.9892	0.9766	0.0161	—	1521
	S-Labs	0.9731	0.9634	0.9775	0.9704	0.0305	—	2270
	NeurAlchemy	0.9671	0.9890	0.9558	0.9721	0.0160	—	941
	pooled_val	0.9761	0.9704	0.9747	0.9726	0.0228	—	4732
	BIPIA	0.5087	0.5093	0.4767	0.4924	0.4593	0.0174	3000

The final backbone is 237.4M parameters (453.24 MB FP16); the corrected export of Section 5.9 adds 4,037 vocabulary rows and has 243.6M. The 211.8 MB figure is a theoretical embedding-FP16/linear-INT4 estimate matching fake-AWQ accounting (embeddings stay FP16 offline; Table 3). The first-shipped `Q4_K_M` file is 202 MB (205 MB corrected), smaller than that estimate, so llama.cpp does *not* keep the full embedding table in FP16: `Q4_K_M` applies mixed *k*-quantization across tensors, including embeddings. Table 15 is therefore fake-AWQ with FP16 embeddings; Section 5.9 scores the packed file itself. Uniform 4-bit packing of every parameter would be 113.3 MB and is not used.

Fig. 3 collects the same story: pooled F1 stays near the 1.5B ceiling, BIPIA *J* collapses at layer pruning and does not recover, and 4-bit size appears only at the last tick. Device timing is Section 5.13.

5.9 Packed GGUF against Fake-AWQ on the Full Sets

The offline figures above come from the fake-quantized checkpoint. To check that GGUF packaging does not diverge from them, we scored packed `Q4_K_M` files on the complete pooled-validation (4732) and BIPIA (3000) sets with native llama.cpp on the M1 Mac, through the deployed path (tokenize, vocabulary map, last-token embedding, two-class head) with a 4096-token context, so only one validation document was truncated. Table 16 sets the results beside Table 15.

Table 16: Packed `Q4_K_M` GGUF files (native llama.cpp, M1 Mac, full sets) against the fake-AWQ checkpoint of Table 15, same rows and labels. *Shipped*: the file first exported, tokenized by its own (truncated) vocabulary. *Shipped, HF ids*: the same file, with the token ids taken from the training tokenizer and nothing else changed. *Re-export*: corrected vocabulary, built from the vocabulary-pruned recovery checkpoint that precedes AWQ, not from the fake-AWQ checkpoint. Fake-AWQ ROC-AUC and TPR@1%FPR are the range over the two final-checkpoint rows of Section 5.10. "—": not computed. Brackets: 95% percentile bootstrap, 1000 resamples. Threshold: the classifier's own (argmax).

Set	Artifact	Acc.	Prec.	Rec.	F1	FPR	ROC-AUC	TPR@1%FPR
Pooled val. (4732)	Fake-AWQ	0.9761	0.9704	0.9747	0.9726	0.0228	—	—
	Shipped GGUF	0.9527	0.9217	0.9737	0.9470	0.0635	0.9873 [0.984, 0.990]	0.846
	Shipped, HF ids	0.9751	0.9713	0.9713	0.9713	0.0220	—	—
	Re-export	0.9746	0.9645	0.9776	0.9710	0.0276	—	—
BIPIA (3000)	Fake-AWQ	0.5087	0.5093	0.4767	0.4924	0.4593	0.499	0.002–0.009
	Shipped GGUF	0.5040	0.5039	0.5220	0.5128	0.5140	0.5255 [0.504, 0.547]	0.0073
	Shipped, HF ids	0.5057	0.5082	0.3513	0.4155	0.3400	—	—
	Re-export	0.5120	0.5128	0.4807	0.4962	0.4567	—	—

On BIPIA every artifact is, like the fake-AWQ checkpoint, not a usable detector: Youden's J is 0.008 for the shipped file (fake-AWQ 0.0174), 0.011 with the training tokenizer's ids and 0.024 for the re-export, and the shipped file's ROC-AUC of 0.526 is only just above chance, its interval excluding 0.5 by 0.004. In-distribution the shipped file was worse than the checkpoint it came from. Its recall is unchanged (0.9737 against 0.9747) but the false-positive rate rose from 0.023 to 0.063, which took precision from 0.970 to 0.922 and F1 from 0.9726 to 0.9470: it retained 0.9470/0.9888 = 95.8% of the uncompressed F1, not 98.4%.

The cause was the file's tokenizer, not the 4-bit weights. The GGUF holds only the first 44,871 vocabulary entries (Section 3.12), so a word whose original token id is at or above that number is split into single letters. In both corpora every row containing such a token was tokenized to a different length by the GGUF than by the training tokenizer, and no row without one was. Feeding the same file the training tokenizer's ids, with the vocabulary map, head and scoring path unchanged, restores validation F1 to 0.9713 and the false-positive rate to 0.0220, the level of the fake-AWQ checkpoint; the packing itself therefore costs about 0.001 F1. The browser and llama-server tokenize with the file's own vocabulary, so the shipped file was fed misspelled text at deployment while the fake-AWQ numbers were not.

The corrected export rebuilds the vocabulary (Section 3.12) and is built from the vocabulary-pruned recovery checkpoint, which precedes AWQ and the final fine-tuning; it is therefore not the fake-AWQ model, and we make no row-level comparison with it. Its token ids equal the training tokenizer's on every row up to the first token the pruning data never contained; a few rows differ after that point, mostly beyond position 512. It scores validation F1 0.9710 (false-positive rate 0.0276) and retains 0.9710/0.9888 = 98.2% of the uncompressed F1. The on-phone classification check of Section 5.13 used the shipped file, and parity with native llama.cpp is a same-file comparison, so it is unaffected; its accuracy figures are those of the shipped file.

5.10 Held-Out Discrimination across Every Checkpoint

Every metric in Tables 6–15 is taken at the classifier's own threshold, so it does not say whether the model still *ranks* an injection above a benign document. This subsection reports threshold-free measures on the same BIPIA split for every checkpoint: ROC-AUC is the probability that a random injection outscores a random benign document, and TPR@1%FPR is injection recall at a 1% false-positive rate, the regime a deployed guard has to operate in.

Table 17 gives both. The uncompressed 1.5B baseline reaches ROC-AUC 0.590: weak, but above chance. Every checkpoint that has passed through structured pruning sits between 0.499 and 0.509, with TPR@1%FPR of 0.000–0.009 against the baseline's 0.054. At a tolerable false-positive rate the compressed models recover essentially no injections and the baseline about one in twenty; the ranking ability is absent, not relocated.

Table 17: Threshold-free held-out discrimination on BIPIA (n = 3000, 1500 positives/1500 negatives) for every checkpoint in the pipeline. The baseline row reproduces Table 6; every other row is the same split and the same preprocessing. ΔAUC is against the uncompressed baseline.

Checkpoint	ROC-AUC	TPR@1%FPR	ΔAUC
Uncompressed 1.5B baseline	0.5900	0.0540	—
4-stage focal-loss recovery	0.5083	0.0087	−0.082
Layer-15 recovery	0.4995	0.0100	−0.090
Vocabulary-pruned recovery	0.5092	0.0000	−0.081
Fake-AWQ + QLoRA (INT4)	0.5079	0.0047	−0.082
Final AWQ + QLoRA	0.4998	0.0020	−0.090
Final fine-tuned checkpoint	0.4988	0.0087	−0.091

Table 18 bounds how much of that 0.590 is injection signal. BIPIA documents are far longer than anything the guard was trained on (training medians 9, 15 and 33 tokens against BIPIA's 277), and the 512-token limit truncates 31.5% of positives but only 13.8% of negatives.

Table 18: Length and truncation controls on the uncompressed 1.5B baseline. "Length only" scores each example by its raw token count and runs no model. "Length-matched" bins by document-length deciles and samples equal positives and negatives from each bin, holding n at 2182.

Subset	n	pos/neg	ROC-AUC	TPR@1%FPR
All BIPIA	3000	1500/1500	0.5900	0.0540
Examples ≤512 tokens	2321	—	0.6386	—
Examples ≤256 tokens	1357	—	0.7339	—
Examples >512 tokens	679	—	0.5595	—
Length-matched subset	2182	1091/1091	0.6395	—
Length only (no model)	3000	1500/1500	0.6585	—
Length only, matched set	2182	1091/1091	0.5007	—

Two things follow. The baseline's advantage is largely a length heuristic: scoring each example by its raw token count, with no model, reaches ROC-AUC 0.659 and beats the fine-tuned guard. That heuristic does not explain the residual signal, though: on the length-matched subset the token-count control falls to 0.501 while the model still reaches 0.640. Truncation also works against detection (0.734 for documents of at most 256 tokens, 0.560 for truncated ones).

The honest reading is that the baseline never solved this task: compression removes a small amount of real ranking ability on a task that was already weakly tractable, and it removes all of it.

5.11 Isolating Pruning, Recovery and Quantization

Every checkpoint in Table 17 after the baseline has been structurally pruned, so the pipeline alone cannot say which step removes the held-out signal, and once BIPIA is at chance a further loss from quantization would be invisible. We therefore applied each step on its own to the uncompressed baseline, with that stage's hyperparameters (Table 3), seed 42, and the same splits and scoring. Table 19 gives the results; the baseline row reproduces Table 17 to within 0.001.

Table 19: Each compression step applied alone to the uncompressed 1.5B baseline. BIPIA n = 3000 (1500/1500); pooled validation n = 4732. Recovery is the first distillation sub-stage with the baseline as teacher (early-stopped at epoch 6, best epoch 2). Pruning is the 80% Wanda FFN cut of Section 3.6; its recovery uses the Sweep column of Table 3 and stopped at epoch 5 of 8 on the session time limit. One seed.

Step Applied to the Baseline	Params (M)	ROC-AUC	TPR@1%FPR	J	FPR	Pooled F1
None (uncompressed)	1543.7	0.5900	0.0540	0.0967	0.0373	0.9888
Recovery training (KD) only	1543.7	0.5954	0.0580	0.1153	0.0380	0.9890
AWQ only	1543.7	0.6133	0.0260	0.0927	0.0747	0.9883
AWQ + QLoRA	1543.7	0.6329	0.0353	0.1333	0.0913	0.9878
80% FFN pruning only	618.9	0.5128	0.0067	−0.0113	0.0747	0.6226
80% FFN pruning + recovery	618.9	0.5448	0.0100	0.0307	0.2147	0.9759

Recovery training alone leaves BIPIA unchanged (ROC-AUC 0.595). AWQ alone and AWQ followed by QLoRA do not lower ROC-AUC either (0.613 and 0.633), so QLoRA is not masking a quantization loss; they do halve TPR at 1% FPR and double the BIPIA FPR at the default threshold, so 4-bit weights change the top of the score distribution without removing the ranking. FFN pruning removes it: ROC-AUC 0.513, with in-distribution F1 also down to 0.623. Recovery then restores in-distribution F1 to 0.976 but held-out ROC-AUC only to 0.545, with TPR at 1% FPR of 0.010. The held-out loss is caused by structured pruning and is not repaired by recovery training, which reproduces, under control, the accuracy-invisible pattern of the full pipeline.

5.12 A Second Backbone Family

To test whether the weak held-out boundary belongs to Qwen2.5, we fine-tuned Llama-3.2-1B-Instruct (1.24B parameters; a different tokenizer and pre-training corpus) with the recipe of Section 3.5, on the same corpora and splits, seed 43, in bfloat16 because FP16 overflowed its loss. This run fine-tunes only; the compression pipeline was not repeated on it, and it is one seed, so it bears on the backbone and not on the seed question. Table 20 compares the two backbones.

Table 20: Uncompressed, fine-tuned backbones on the same held-out BIPIA split (n = 3000) and pooled validation. Qwen row: seed 42, Tables 17 and 6; Llama row: seed 43.

Backbone	Pooled F1	ROC-AUC	TPR@1%FPR	J	F1 (BIPIA)
Qwen2.5-1.5B	0.9888	0.5900	0.0540	0.0974	0.2298
Llama-3.2-1B	0.9893	0.5370	0.0427	0.0753	0.3123

In-distribution quality is the same (0.9893 against 0.9888), so the Llama run is not under-trained, and it is no better on held-out data: ROC-AUC 0.537, J = 0.075. Qwen is ahead on three of the four held-out measures; Llama is ahead only on thresholded F1, which again disagrees with ROC-AUC and J about which model is better. The weak held-out boundary is therefore not specific to one decoder family. The run does not show that pruning would collapse Llama in the same way, because that was not tested.

5.13 On-Device Performance

Table 21 reports the on-handset classification run of Section 3.16: 400 validation and 400 BIPIA documents of at most 504 tokens, classified in Chromium on the Realme C25 and compared with a native llama.cpp run of the same `Q4_K_M` file.

Table 21: On-handset classification of the first-shipped `Q4_K_M` GGUF (truncated vocabulary, Section 5.9) on the Realme C25, against a native llama.cpp run of the same file on the same 400 documents per corpus (documents of at most 504 tokens; label-stratified sample, seed 0). Disagreements: all within 0.01 of the decision boundary. Margin: injection logit minus benign logit for a document. Agree: fraction of documents with the same predicted label on phone and reference. Margin r: Pearson correlation between the phone and reference margins over the 400 documents. Max $|\Delta|$: largest absolute difference between the phone and reference margins.

Corpus	n	Agree	Disagree	Margin r	Max $\lvert\Delta\rvert$	AUC Phone	AUC Ref.	Acc. Phone	Acc. Ref.
Validation	400	0.9925	3	0.9988	0.106	0.9912	0.9913	0.9525	0.9550
BIPIA	400	0.9875	5	0.9972	0.089	0.5901	0.5900	0.5425	0.5450

The phone reproduces the reference prediction on 397 of 400 validation documents (99.25%) and 395 of 400 BIPIA documents (98.75%). All eight disagreements fall within 0.01 of zero margin (injection minus benign logit), as floating-point differences between the WASM and native kernels would produce; the largest margin difference is 0.11. ROC-AUC on the phone is 0.9912 on validation and 0.5901 on BIPIA, against 0.9913 and 0.5900 for the reference run on the same rows.

These BIPIA figures describe only the shorter documents the build can embed; with 400 documents per corpus the accuracy intervals are about ±5 points, so this is a parity check and not a new estimate of BIPIA performance. The same file scored on the full sets is compared with the fake-AWQ checkpoint in Section 5.9.

Table 22 gives the on-phone time for the whole path (tokenization, vocabulary mapping, embedding, head). Latency grows with length, from a median of 0.12 s for validation prompts of at most 32 tokens to about 3 s for documents of 129–504 tokens; the BIPIA median is 2.9 s (95th percentile 5.9 s) and the validation median 0.15 s (1.6 s). Three of the 800 rows took more than 10 s, the longest 41 s; the cause was not identified and they are included in every figure.

Table 22: On-phone classification latency (Realme C25, Chromium, wllama) by input length in tokens; milliseconds, whole path. Each row counted once, using its first measurement.

Corpus	Tokens	*n*	Median	p95
Validation	≤32	300	118	271
Validation	33–128	77	582	1178
Validation	129–504	23	2932	5684
Validation	all	400	148	1628
BIPIA	33–128	46	1025	1273
BIPIA	129–504	354	3153	5920
BIPIA	all	400	2942	5856

Browser-reported memory including WASM was 0.91–1.18 GB across sessions on this 4 GB phone; the JS heap alone stayed below 224 MB. The 196 MB figure of the earlier protocol is a JS-heap lower bound that omits the WASM memory in which the model resides.

The earlier protocol (one generated token for each of 50 short prompts, no classification) gave the load times of Table 23: 18.7 s on the Realme C25 and 0.62 s for the same page on the M1 Mac. Its latency columns are superseded by Table 22; the Mac figures are CPU-only WASM wall-clock times, not native llama.cpp and not process resident set size (RSS). iOS/WebKit refuses allocation for the 202 MB file, which does not contradict the Android result.

Table 23: Earlier timing protocol: in-browser wllama time for one generated token and Chromium heap, first-shipped `Q4_K_M` GGUF (202 MB). N = 50 random short test prompts; eight threads; isolated context. This protocol did not classify; its latency is superseded by Table 22. Heap is a Chromium JS-heap lower bound that excludes WASM memory, not process RSS. The Realme C25 is a MediaTek Helio G70 (MT6769V/CB) with 4 GB RAM.

Host	Chip/RAM	Load	Mean	Med.	p95	Inj.	Ben.	Heap
Realme C25 (browser)	Helio G70, 4 GB	18.7 s	2332 ms	2115 ms	4325 ms	2795 ms	1868 ms	196/966 MB
Mac (browser)	M1, 8 GB	0.62 s	127 ms	115 ms	256 ms	154 ms	99 ms	203/4192 MB

5.14 The Pipeline against a Directly Trained 0.5B Model and PIGuard

The intra-family comparison (Table 13) shows that a fine-tuned, never-compressed Qwen2.5-0.5B has the higher pooled F1 (0.9863 against 0.9725). Whether the pipeline earns its complexity therefore depends on what it buys at deployment. Table 24 compares the corrected pipeline file with a 0.5B model packed through the same export path and with PIGuard [48], an encoder-based guard released with a benchmark for over-defense (NotInject); PIGuard is evaluated as released, without fine-tuning on our corpora, so its validation figures are zero-shot.

Table 24: Pipeline file (corrected re-export) against a Qwen2.5-0.5B packed the same way and PIGuard. Mac: native llama.cpp (4096 tokens) or PyTorch (PIGuard), M1, mean over the full sets. Phone: Realme C25, Chromium (wllama), 20 documents of at most 504 tokens (10 validation, 10 BIPIA, seed 0, identical token counts for both models), context 512, two threads, run back to back. The 0.5B file has an *untrained* head: it is timed, never scored, and its F1 is that of the fine-tuned floating-point checkpoint (Table 13), not of the packed file. Memory: `measureUserAgentSpecificMemory` after load and at the end of the run. "n/m": not measured.

	Pipeline	**0.5B**	**PIGuard**
Parameters (M)	243.6	494.0	184.4
File (MB)	205.4 (`Q4_K_M`)	397.8 (`Q4_K_M`)	703 (FP32)
Pooled-validation F1	0.9710 (packed)	0.9863 (FP)	0.7824
Validation FPR	0.0276	n/m	0.0825
BIPIA J	0.024 (packed)	−0.0766 (FP)	0.1867
NotInject FPR (all benign)	n/m	n/m	0.115
Mac mean latency, validation (ms)	44.0	62.1	n/m
Mac mean latency, BIPIA (ms)	208.0	328.8	n/m
Phone model load (s)	3.6	11.2	n/m
Phone memory after load/end (MB)	920/1217	1566/2288	n/m
Phone median latency, 20 docs (s)	15.7	13.2	n/m
Phone median, ≤32 tokens (s)	1.50	1.25	n/m

What the pipeline buys is footprint. The file is 1.9 times smaller, loads 3.1 times faster on the phone, and the browser reports 41% less memory after load and 47% less at the end of the run. On the Mac it is 1.4 to 1.6 times faster. What it does not buy is phone latency: the 0.5B model was faster on all 20 documents, with a median ratio of 0.845 (10th–90th percentile 0.83–0.86), so the pipeline file is 18% slower there. Both files are mixed layouts under the `Q4_K_M` label. In the pipeline file the 13 feed-forward down-projections of width 1617, which is not a multiple of the 32-wide quantization block, stayed in 16-bit floating point (61.6 of 205 MB); in the 0.5B file most weights are `Q5_0` (165 MB) or `Q8_0` (139 MB). We did not test whether the 16-bit tensors explain the phone result, for example by padding the layers to a quantizable width, and name it as the likely cause only.

In accuracy the pipeline loses to the 0.5B model in-distribution, and wins nothing held out: neither is a usable BIPIA detector. Against PIGuard, the pipeline file is the better in-distribution classifier on our validation data (F1 0.971 against 0.782, false-positive rate 0.028 against 0.083), but PIGuard is the better held-out ranker (BIPIA ROC-AUC 0.791 against the shipped file's 0.526; at its own threshold it flags 71% of BIPIA negatives, $J = 0.187$). PIGuard's input limit truncated 479 of the 3000 BIPIA documents and 65 validation documents. On the Mac its medians are 55 ms (validation) and 137 ms (BIPIA) against 29 and 197 ms for the first-shipped file, measured in an earlier session. We did not run NotInject on our models, so over-blocking is compared on the validation false-positive rate only, and we did not run PIGuard on the phone.

Phone speed.

The phone was far slower than in the first run. For inputs of at most 32 tokens the median was 1.5 s here against 0.12 s in the earlier on-phone run (Table 22), and for 129–512 tokens about 46 s against 3 s. The ratios between the two models above come from one session and are stable (the 0.5B model was faster on every document), but the absolute times are not comparable with the earlier table. A single-thread JavaScript probe run at the start and end of each session gave 0.30 and 0.25 GFLOP/s for the pipeline file and 0.37 and 0.26 for the 0.5B file, so the CPU was in a similar and slow state in both runs. We ruled out the network (latency excludes download), the page code, a background tab and the thread count (four threads gave a speed-up of only 1.3), but not the phone's thermal or power state, and we do not know why it is slower. The earlier run itself shows such a state: repeated measurements of its rows, made while it was degraded, have block medians of 15 to 37 s, the same order as here; Table 22 uses each row's first measurement, taken earlier. The 20-document sample is a comparison of two models under identical conditions, not a latency benchmark.

6 Discussion

The pipeline does one thing reliably and fails at another. Width, depth, vocabulary and 4-bit reductions take a 1.5B decoder from 1562M parameters to a 202–205 MB GGUF classifier whose pooled-validation injection F1 is 0.973 as the fake-AWQ checkpoint and 0.971 as a correctly exported packed file (0.947 for the file first shipped, whose vocabulary was truncated). The same artifact is not a usable detector of held-out, document-embedded injections: BIPIA accuracy stays near 0.50 and Youden's J falls from 0.097 to about 0.01. The apparent held-out F1 gain is a more liberal decision boundary, not new skill.

6.1 RQ1: What Compression Does to In-Distribution and Held-Out Accuracy

Relative to the uncompressed LoRA-tuned 1.5B checkpoint, the fake-AWQ model retains 0.9726/0.9888 = 98.4% of pooled-validation F1 after an 84.8% parameter cut; the file first shipped retains 95.8% and the corrected re-export 98.2% (Section 5.9). Most of the cost is paid at layer pruning (0.9822 → 0.9702–0.9725); the other stages each move F1 by less than 0.01.

Held-out BIPIA cannot be summarized by F1. The uncompressed model is conservative (recall 0.1347, FPR 0.0373, J = 0.0974). After compression recall and FPR both approach 0.5, F1 rises toward 0.49 and J collapses; on a balanced set that is the arithmetic of a near-coin-flip, not transfer. After four-stage recovery the BIPIA family is no longer unseen, because Stages 1 and 3 train on document-embedded wrappers, yet distillation, which does not, is no better (J = 0.0073). The sweep-stage F1 peak of 0.5504 was not reproduced on the carried-forward checkpoint (0.2541) and is not a finding.

On in-distribution grounds the intra-family comparison undercuts the pipeline: the never-pruned Qwen2.5-0.5B has pooled F1 0.9863 at 494M parameters, above the 401.9M distilled student (0.9725), and is simpler to train. The compressed line is smaller (243.6M; 205 MB packed against 398 MB for a 0.5B file) and loads and runs in less memory (Section 5.14). That is a deployment argument, not a detection-quality one, and it holds for storage, download and memory, not for latency on the phone, where the 0.5B file was faster. The pipeline is the better choice only when file size or resident memory is the binding constraint and a lower in-distribution F1 is acceptable; otherwise train the 0.5B model directly.

6.2 RQ2: Where Safety-Relevant Behavior Moves, and Where It Does Not

RQ2 asked whether the pipeline produces the accuracy-invisible degradation described by Rath and Maliakkal [4], and at which stage. Under the definition of Section 3.15 (a material drop in BIPIA J without a commensurate drop in pooled F1) the answer is yes, and the cause is structured pruning. J is already down to

0.0533 after the FFN-only rounds while pooled F1 is flat (0.9888 → 0.9822); it falls to 0.0166 (four-stage) and 0.0073 (distillation) at layer pruning and stays in the 0.005–0.017 band through vocabulary pruning and fake-AWQ. Quantization adds no further collapse: BIPIA F1 changes by −0.0004. This is narrower than a rebuttal of Rath et al., who studied generative alignment in chat models; here the discriminative loss had already happened. Applied alone to the unpruned baseline (Section 5.11), recovery training and AWQ leave ROC-AUC at or above 0.590, while FFN pruning alone drops it to 0.513 and recovery after pruning reaches only 0.545. Structured pruning is the cause, recovery does not repair it, and quantization only lowers detection at the strictest false-positive rate.

An F1-only readout would have reversed the conclusion, because BIPIA F1 rises through the stages in which J falls. Reporting FPR, J and ROC-AUC on a balanced held-out set is what exposes the degradation, which is the methodological point for guardrail papers that compress and then quote a single metric [5].

The uncompressed 1.5B guard is not an adequate indirect-injection detector either. Its ROC-AUC is 0.590, matching on length lifts that only to 0.640, and a raw token count reaches 0.659; a second backbone is no better (Section 5.12). Every post-pruning checkpoint sits at ROC-AUC ≈0.50, so compression removed a small amount of genuine ranking ability on a task that was already weakly tractable, and no operating point remains to be moved.

6.3 RQ3: Browser Latency on a 4 GB Phone and on a Mac

The Realme C25 classifies sampled documents in Chromium and its predictions match a native llama.cpp run of the same file on 99.0% of 800 documents, every disagreement within 0.01 of the decision boundary. The median is 0.15 s for validation prompts and 2.9 s for BIPIA documents of up to 504 tokens (95th percentile 5.9 s). Browser-reported memory including WASM is 0.9–1.2 GB on a 4 GB phone, several times the 196 MB JS-heap figure of the earlier protocol. The 512-token physical batch of this build excludes documents longer than 504 tokens, about a third of BIPIA. iOS/WebKit cannot allocate the module, so an iPhone path would have to be native, while an Android path can be this page. The in-distribution over-blocking of the first-shipped file came from its vocabulary and not from its speed (Section 5.9). Against a 0.5B file packed the same way, the pipeline file loads three times faster and uses less memory but is 18% slower on this phone, for a reason we did not establish (Section 5.14). There is no native application and no cloud-judge arm.

6.4 What the Findings Do and Do Not License

The study licenses four claims. (1) The artifact classified on the phone in the on-device run is the first-shipped `Q4_K_M` GGUF, which scores below the fake-AWQ checkpoint in-distribution (F1 0.947 against 0.973) because of its truncated vocabulary; a corrected file scores 0.971. (2) The compression sequence destroys the already-small held-out edge on document-embedded injections, invisibly if only F1 is reported. (3) That loss is caused by structured pruning: applied alone to the uncompressed model, pruning removes it and recovery does not restore it, while recovery training or AWQ alone do not cause it. (4) The first-shipped file classifies in Chromium on a 4 GB Helio G70 phone at 0.15 to 2.9 s depending on length, and fails to initialize under iOS/WebKit.

It does not license the claims that compression regularizes a guard into a better OOD detector; that pruning a 1.5B model beats training the native 0.5B model on this task (it gives a smaller file and less memory, not higher F1 or a faster phone); that quantizing a security classifier is safety-neutral in general; that three seconds per document suits every assistant; or that the results hold against adaptive attackers or for non-English text (Sections 3.2 and 3.17).

6.5 Limitations

Language and obfuscation. No held-out evaluation covers non-English or obfuscated text. BIPIA is 99.0% English by language identification and contains no mostly non-Latin documents, only 1.7% base64-like runs and four zero-width cases (Section 3.17). Multilingual and obfuscated examples exist only as synthetic training augmentation, and non-Latin script is a label cue in training and validation (235 of 238 and 41 of 42 such documents are injections), so the guard may have learned that cue, and our results cannot say how it behaves on multilingual or encoded attacks.

Attacker and task. Only non-adaptive, document-embedded injections are tested (Section 3.2); there are no adaptive attacks, no agent-level benchmark such as InjecAgent, and no hosted-LLM judge for comparison.

Held-out set. BIPIA's negatives are model-generated, so class differences include realism and length; the documents are far longer than the training data, and the 512-token limit truncates positives more often than negatives.

Scale of the evidence. One compression trajectory on one backbone and one seed per run; a second backbone (Llama-3.2-1B) was fine-tuned but not compressed. The single-step controls of Section 5.11 re-implement each stage rather than reuse the original checkpoints, which were not kept; the pruned-and-recovered control stopped at 5 of 8 epochs with validation F1 still rising, and the second distillation sub-stage was not run. The corrected GGUF is built from the checkpoint that precedes AWQ and was compared with fake-AWQ on aggregate metrics only. The headline pooled-validation F1 of 0.9726 is not tied to one checkpoint: when every saved checkpoint was re-scored, 0.9726 matched the offline-distillation checkpoint (0.9725) and not the vocabulary-pruned AWQ checkpoint (0.9618), the held-out BIPIA F1 of 0.4924 was closest to the latter (0.5012), and the Youden's J of 0.0174 was closest to the fake-AWQ INT4 checkpoint (0.0187). No single checkpoint reproduces all three figures, and we did not resolve why; the conclusions rest on the stage-wise tables and not on any one of these numbers.

Comparison with other models. The 0.5B model in the deployment comparison has an untrained head, so it is timed and never scored; a trained 0.5B file was not packed or scored, so what 4-bit packing costs its F1 is unknown. NotInject over-blocking was measured for PIGuard only, PIGuard was not run on the phone, and the phone comparison uses 20 documents in one session on a phone whose speed was far below the earlier run for a reason we did not establish.

Deployment. The phone run covers one Android device, one browser, documents of at most 504 tokens (about two thirds of BIPIA), and one run with a few unexplained stalls over 10 s; iOS/WebKit does not run the file at all.

7 Conclusion

This paper compressed a Qwen2.5-1.5B-Instruct prompt-injection classifier through structured feed-forward pruning, residual-stream layer pruning, vocabulary reduction and 4-bit packing, then served the packed GGUF in a browser. In-distribution F1 stays high (0.9726 for the fake-AWQ checkpoint, 0.9710 for a correctly packed file, 0.9888 uncompressed); the 0.9470 of the file first shipped was an export fault in its vocabulary, found and corrected. Held-out BIPIA performance does not: J falls from 0.0974 to roughly 0.01 while F1 rises toward 0.5, and ROC-AUC falls from 0.590 to 0.500, a loss of ranking ability and not an operating-point shift. Applying each step alone to the uncompressed model locates the cause: FFN pruning drops ROC-AUC to 0.513, and recovery restores in-distribution F1 but not the held-out signal (0.545), while recovery training or AWQ alone leave it at 0.595 and 0.613. A never-pruned 0.5B model has higher in-distribution F1 but is below chance on BIPIA ($J = -0.0766$). The pipeline's measured advantage over it is

footprint: a 1.9 times smaller file, 3.1 times faster load and 41%–47% less browser memory on the phone, and 1.4 to 1.6 times lower Mac latency; on the phone it was 18% slower, and its F1 is lower.

The uncompressed baseline was itself a weak held-out detector (J = 0.0974, ROC-AUC 0.590), no better than a one-feature length heuristic (ROC-AUC 0.659), and the BIPIA documents (median 277 tokens) are far longer than the training corpora (medians 9, 15 and 33). Compression therefore removed a small amount of genuine signal and did not create a new failure; the pipeline deserves neither credit nor blame for the baseline's limited skill.

The first-shipped file runs in Chromium on a Realme C25 (Helio G70, 4 GB) at a median 0.15 s (validation) to 2.9 s (BIPIA) for documents of up to 504 tokens, matching a native run of the same file on 99.0% of rows. Longer documents cannot be classified by this WASM build, iOS/WebKit does not allocate the module, and there is no cloud-judge arm.

The contribution is a stage-wise measurement, not a new detector or a finished product, and its limits are set out in Section 6.5. Useful next steps are adaptive attacks, a multilingual and obfuscated held-out set, repeating the pipeline on a second backbone and more seeds, a native build with process-level memory measurement, and a hosted-LLM judge on the same prompts.

Acknowledgement: The authors thank the maintainers of the xTRam1, S-Labs, NeurAlchemy and BIPIA corpora for making these datasets publicly available.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: S. M. Nihal Ahmed: conceptualization, methodology, software, experiments, formal analysis, validation, visualization, resources, and original draft preparation. Afrim Hossen Khan: data curation and review of the compression experiments. Sabikun Nahar Sinthia: benchmark evaluation and manuscript review. Md. Rashedul Alam: supervision, project administration, and critical review of the manuscript. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The three in-distribution training/benchmark corpora (xTRam1, S-Labs, NeurAlchemy) and the held-out BIPIA-derived corpus used in this study are third-party, publicly available datasets; their sources and repository identifiers are listed in Table 1. No new data were collected for this study. The trained model checkpoints and processed data splits are not hosted in a public repository at the time of submission but can be made available by the corresponding author upon reasonable request. The code for the compression pipeline and the in-browser serving page is available at https://github.com/nihal4/Accuracy-Invisible-Safety-Degradation-in-a-Compressed-On-Device-Prompt-Injection-Classifier.

Ethics Approval: This study analysed only publicly available, third-party datasets and involved no human participants, no animal subjects, and no collection of new personal data. Ethics approval was therefore not required.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Milani A, Franzoni V, Florindi E. Indirect prompt injection in large language models. Neural Comput Appl. 2026;38(13):530. doi:10.1007/s00521-026-12266-x.
2. Dikici S, Bilgin TT. Small language models: a systematic review of computational trade-offs, privacy advantages and deployment in intelligent systems. Expert Syst. 2026;43(8):e70331. doi:10.1111/exsy.70331.
3. Zhu X, Li J, Liu Y, Ma C, Wang W. A survey on model compression for large language models. Trans Assoc Comput Linguist. 2024;12(2):1556–77. doi:10.1162/tacl_a_00704.
4. Rath PK, Maliakkal R. Quantization undoes alignment: bias emergence in compressed LLMs across models and precision levels. In: Proceedings of the 2026 IEEE Cloud Summit; 2026 Jun 25–26; Washington, DC, USA. p. 57–63.

5. Dong Y, Mu R, Zhang Y, Sun S, Zhang T, Wu C, et al. Safeguarding large language models: a survey. Artif Intell Rev. 2025;58(12):382. doi:10.1007/s10462-025-11389-2.
6. Alzahrani A. PromptGuard: a structured framework for injection resilient language models. Sci Rep. 2026;16(1):1277. doi:10.1038/s41598-025-31086-y.
7. Deng Z, Guo Y, Han C, Ma W, Xiong J, Wen S, et al. AI agents under threat: a survey of key security challenges and future pathways. ACM Comput Surv. 2025;57(7):1–36. doi:10.1145/3716628.
8. He F, Zhu T, Ye D, Liu B, Zhou W, Yu PS. The emerged security and privacy of LLM agent: a survey with case studies. ACM Comput Surv. 2025;58(6):1–36. doi:10.1145/3773080.
9. Geng T, Xu Z, Qu Y, Wong WE. Prompt injection attacks on large language models: a survey of attack methods, root causes, and defense strategies. Comput Mater Contin. 2026;87(1):1–10.
10. Gulyamov S, Gulyamov S, Rodionov A, Khursanov R, Mekhmonov K, Babaev D, et al. Prompt injection attacks in large language models and AI agent systems: a comprehensive review of vulnerabilities, attack vectors, and defense mechanisms. Information. 2026;17(1):54. doi:10.3390/info17010054.
11. Mathew ES. Enhancing security in large language models: a comprehensive review of prompt injection attacks and defenses. J Artif Intell. 2025;7(1):347–63.
12. Derner E, Batistič K, Zahálka J, Babuška R. A security risk taxonomy for prompt-based interaction with large language models. IEEE Access. 2024;12:126176–87.
13. Yao Y, Duan J, Xu K, Cai Y, Sun Z, Zhang Y. A survey on large language model (LLM) security and privacy: the good, the bad, and the ugly. High-Confid Comput. 2024;4(2):100211. doi:10.1016/j.hcc.2024.100211.
14. Lan Q, Kaul A, Jones S. Prompt injection detection in LLM integrated applications. Int J Netw Dyn Intell. 2025;4(2):100013. doi:10.53941/ijndi.2025.100013.
15. Jamal S, Wimmer H, Sarker IH. An improved transformer-based model for detecting phishing, spam and ham emails: a large language model approach. Secur Priv. 2024;7(5):e402. doi:10.1002/spy2.402.
16. Gu J, Yuan J, Cai J, Zhou X, Fan L. La-LoRA: parameter-efficient fine-tuning with layer-wise adaptive low-rank adaptation. Neural Netw. 2026;194:108095.
17. Ratsapa P, Thonglek K, Chantrapornchai C, Ichikawa K. Automated pruning framework for large language models using combinatorial optimization. AI. 2025;6(5):96.
18. Lin J, Tang J, Tang H, Yang S, Xiao G, Han S. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. GetMobile Mob Comput Commun. 2025;28(4):12–7. doi:10.1145/3714983.3714987.
19. Das BC, Amini MH, Wu Y. Security and privacy challenges of large language models: a survey. ACM Comput Surv. 2025;57(6):1–39. doi:10.1145/3712001.
20. Qu G, Chen Q, Wei W, Lin Z, Chen X, Huang K. Mobile edge intelligence for large language models: a contemporary survey. IEEE Commun Surv Tutor. 2025;27(6):3820–60.
21. Friha O, Ferrag MA, Kantarci B, Cakmak B, Ozgun A, Ghoualmi-Zine N. LLM-based edge intelligence: a comprehensive survey on architectures, applications, security and trustworthiness. IEEE Open J Commun Soc. 2024;5(8):5799–856. doi:10.1109/ojcoms.2024.3456549.
22. Jiang S, Zhou X, Zhang M, Xu C, Liao G, Chen J, et al. Edge large language models: a comprehensive survey. CCF Trans Pervasive Comput Interact. 2026;8(2):181–210. doi:10.1007/s42486-025-00227-7.
23. Wang R, Gao Z, Zhang L, Yue S, Gao Z. Empowering large language models to edge intelligence: a survey of edge efficient LLMs and techniques. Comput Sci Rev. 2025;57(2):100755. doi:10.1016/j.cosrev.2025.100755.
24. Chen H, Deng W, Yang S, Xu J, Jiang Z, Ngai ECH, et al. Toward edge general intelligence via large language models: opportunities and challenges. IEEE Netw. 2025;39(5):263–71.
25. Zheng Y, Chen Y, Qian B, Shi X, Shu Y, Chen J. A review on edge large language models: design, execution, and applications. ACM Comput Surv. 2025;57(8):1–35. doi:10.1145/3719664.
26. Zhao S, Jia M, Guo Z, Gan L, Xu X, Wu X, et al. A survey of recent backdoor attacks and defenses in large language models. IEEE Trans Inf Forensics Secur. 2025.
27. Wan A, Wallace E, Shen S, Klein D. Poisoning language models during instruction tuning. In: Proceedings of the 40th International Conference on Machine Learning (ICML); 2023 Jul 23–29; Honolulu, HI, USA. p. 35413–25.

28. Xu J, Ma MD, Wang F, Xiao C, Chen M. Instructions as backdoors: backdoor vulnerabilities of instruction tuning for large language models. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL); 2024 Jun 16–21; Mexico City, Mexico. p. 3111–26.
29. Qi F, Li M, Chen Y, Zhang Z, Liu Z, Wang Y, et al. Hidden killer: invisible textual backdoor attacks with syntactic trigger. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP). Association for Computational Linguistics; 2021 Aug 1–6; Online. p. 443–53.
30. Hong S, Panaitescu-Liess MA, Kaya Y, Dumitraş T. Qu-ANTI-zation: exploiting quantization artifacts for achieving adversarial outcomes. In: Proceedings of the 35th International Conference on Neural Information Processing Systems; 2021 Dec 6–14; Online. p. 9303–16.
31. Egashira K, Vero M, Staab R, He J, Vechev M. Exploiting LLM quantization. In: Proceedings of the 38th International Conference on Neural Information Processing Systems; 2024 Dec 10–15; Vancouver, BC, Canada. p. 41709–32.
32. Wang S, Zhu T, Ye D, Zhou W. When machine unlearning meets retrieval-augmented generation (RAG): keep secret or forget knowledge? IEEE Trans Dependable Secure Comput. 2026;23(2):2030–45. doi:10.1109/tdsc.2025.3620832.
33. Ye D, Zhu T, Wang S, Liu B, Zhang LY, Zhou W, et al. Data-free model-related attacks: unleashing the potential of generative AI. In: Proceedings of the 34th USENIX Security Symposium (USENIX Security 2025); 2025 Aug 13–15; Seattle, WA, USA. p. 1709–27.
34. Qwen Team. Qwen2.5-1.5B-InStruct (model card). Hugging Face; 2024 [cited 2026 Oct 2]. Available from: https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct.
35. Warner B, Chaffin A, Clavié B, Weller O, Hallström O, Taghadouini S, et al. Smarter, Better, Faster, Longer: a modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers); 2025 Jul 27–Aug 1; Vienna, Austria. p. 2526–47.
36. Yi J, Xie Y, Zhu B, Kiciman E, Sun G, Xie X, et al. Benchmarking and defending against indirect prompt injection attacks on large language models. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '25); 2025 Aug 3–7; Toronto, ON, Canada. p. 1809–20.
37. Zhan Q, Liang Z, Ying Z, Kang D. InjecAgent: benchmarking indirect prompt injections in tool-integrated large language model agents. In: Findings of the Association for Computational Linguistics: ACL 2024; 2024 Aug 11–16; Bangkok, Thailand. p. 10471–506.
38. Li H, Dong Q, Tang Z, Wang C, Zhang X, Huang H, et al. Synthetic data (Almost) from scratch: generalized instruction tuning for language models. Trans Mach Learn Res. 2025
39. S-Labs. prompt-injection-dataset. Hugging Face; 2026 [cited 2026 Oct 2]. Available from: https://huggingface.co/datasets/S-Labs/prompt-injection-dataset.
40. NeurAlchemy. Prompt Injection and Jailbreak Detection Dataset. Hugging Face; 2026 [cited 2026 Oct 2]. Available from: https://huggingface.co/datasets/neuralchemy/Prompt-injection-dataset.
41. Alamsabi M, Tchuindjang M, Brohi S. Embedding-based detection of indirect prompt injection attacks in large language models using semantic context analysis. Algorithms. 2026;19(1):92. doi:10.3390/a19010092.
42. Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al. LoRA: low-rank adaptation of large language models. In: Proceedings of the Tenth International Conference on Learning Representations (ICLR 2022); 2022 Apr 25–29; Online.
43. Sun M, Liu Z, Bair A, Kolter JZ. A simple and effective pruning approach for large language models. In: Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024); 2024 May 7–11; Vienna, Austria.
44. Conneau A, Khandelwal K, Goyal N, Chaudhary V, Wenzek G, Guzmán F, et al. Unsupervised cross-lingual representation learning at scale. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics; 2020 Jul 5–10; Online. p. 8440–51.

45. Lin TY, Goyal P, Girshick R, He K, Dollár P. Focal loss for dense object detection. IEEE Trans Pattern Anal Mach Intell. 2020;42(2):318–27. doi:10.1109/tpami.2018.2858826.
46. Gou J, Yu B, Maybank SJ, Tao D. Knowledge distillation: a survey. Int J Comput Vis. 2021;129(6):1789–819. doi:10.1007/s11263-021-01453-z.
47. Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L. QLORA: efficient finetuning of quantized LLMs. In: Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23); 2023 Dec 10-16; New Orleans, LA, USA. Red Hook, NY, USA: Curran Associates Inc.; 2023. p. 10088–115.
48. Li H, Liu X, Zhang N, Xiao C. PIGuard: prompt injection guardrail via mitigating overdefense for free. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers); 2025 Jul 27–Aug 1; Vienna, Austria. p. 30420–37.