NEUROPROBE: LLM-AUGMENTED REVERSE ENGINEERING OF DEEP NEURAL NETWORK BINARIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning compilers translate high-level models into optimized native binaries, yet these executables often leak structural patterns that expose proprietary architectures. We present *NeuroProbe*, a reverse-engineering pipeline that reconstructs operator semantics, kernel parameters, and network topology from compiled neural-network binaries under both clean and identifier-stripped conditions. Our solution ensembles a symbol-stripped XGBoost model trained on assembly features with a LoRA-tuned Qwen2.5-3B-Instruct model that outputs structured chain-of-thought rationale. We build and release a benchmark of 400 compiled binaries spanning three compiler backends (TVM, IREE, TensorRT) and three architecture families (CNN, Transformer, and Hybrid CNN-Transformer), covering 117 base models and 13,438 extracted kernel function records. NeuroProbe recovers operators with 99.7% F1 under clean compilation and 79.7% F1 under obfuscated setting, while architecture-family accuracy remains at 95% per binary in both settings. Beyond classification, NeuroProbe performs fine-grained hyperparameter estimation and topology reconstruction. We perform signal-ablation experiments to quantify the explicit contributions of symbol names versus assembly features, demonstrating that compiled transformer models remain substantially recoverable under heavy obfuscation despite prior assumptions of robustness.

1 INTRODUCTION

Commercial deployment of deep neural networks (DNNs) increasingly relies on deep learning compilers such as Apache TVM (Chen et al., 2018), IREE (Vanik et al., 2022), NVIDIA TensorRT, and Glow (Rotem et al., 2018) to translate high-level models into optimized, target-specific native binaries. Deep learning compilers aggressively fuse operators, rewrite tensor memory layouts, and strip high-level metadata. Under this assumption, an adversary who obtains the compiled artifact must exhaustively reconstruct the hidden operator graph topology and underlying architecture family before mounting any white-box attack.

Prior reverse-engineering literature has substantially challenged these security assumptions, primarily within the scope of Convolutional Neural Networks (CNNs) compiled for CPUs. Early methodologies relied on x86-specific side (Dasgupta et al., 2019) channels (Hua et al., 2018) or static library signatures (Zhang et al., 2023). Subsequent advances leveraged static control-flow recovery and AST lifting (Wu et al., 2022), while tools like BTD (Liu et al., 2023) pioneered dynamic symbolic execution over assembly instruction traces. More recently, hybrid frameworks such as NeuroDeX (Li et al., 2025) have combined trace analysis with large language models (LLMs) to recognize operator types across CPU architectures. In response, compiler defenses like dedicated obfuscation passes (Zhong et al., 2025) and fake-operator insertion have slashed CNN operator recovery rates under general decompilation to below 5%. However, existing frameworks suffer from two fundamental limitations: **(1)** Existing decompilers (BTD, DnD, NeuroDeX) are strictly single-compiler and CPU-centric, failing to parse closed-source GPU engine plans like TensorRT that lack reachable disassembly tables. **(2)** Modern Transformer models (Vaswani et al., 2017) such as BERT (Devlin et al., 2019) and ALBERT (Lan et al., 2020) possess strict data-flow dependencies and multi-head attention structures that resist standard fusion rules. Defenses like NeuroShield (Zhong et al., 2025) leave over 47% of Transformer operators exposed, yet state-of-the-art reverse-engineering pipelines achieve severely degraded recovery rates on these architectures due to complex tensor-reshaping signatures.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To bridge these gaps, recent works have explored domain-adapted LLMs (Chen et al., 2025a) and agentic intermediate-representation reasoners (Chen et al., 2025b) to interpret stripped execution binaries. However, existing contextual frameworks construct local topology by querying adjacent kernel function metadata. In practice, these methods implicitly over-rely on unstripped symbol names or ground-truth neighbor annotations during context construction (Brown et al., 2020). When deployed on real-world, fully stripped or obfuscated binaries, these models collapse because their contextual features leak evaluation labels during offline training. Another problem is that naïvely prompting fine-tuned LLMs leads to severe grounding errors. Language models frequently hallucinate mathematically or structurally impossible kernel signatures. Standard chain-of-thought (Wei et al., 2022) fine-tuning fails to enforce these strict low-level system invariants. We discuss the related work in more detail in Appendix C.

We address this gap in deep learning binary analysis by proposing *NeuroProbe*, a statistical-neural framework that recovers operator, macro-architecture, and full graph topologies from stripped, cross-compiler binaries spanning both CPU executables and TensorRT engine plans. To capture global execution structure without introducing label contamination, we introduce **Leak-Free Neighboring-Function Context Extraction**. For any target function, adjacent functions are queried using operator predictions drawn *exclusively* from a name-blind statistical prior rather than ground-truth metadata, enabling spatial topological context propagation across stripped binaries with strictly zero evaluation leakage under full obfuscation. Second, we propose **Structural-Constraint-Accumulating Chain-of-Thought (SCA-CoT)**, a specialized reasoning paradigm that conditions a LoRA-adapted Qwen2.5-3B-Instruct model on a four-block structured context (`FUNCTION`, `ASM/BYTES`, `PRIOR`, and `CONTEXT`). It forces the language model to systematically accumulate and verify low-level structural and memory-alignment constraints before committing to an operator schema. To prevent degenerate identity mapping onto statistical priors during training, we incorporate *Prior Dropout* combined with GRPO-lite policy tuning. Our primary contributions are:

- **NeuroProbe:** A unified statistical-neural system integrating leak-free statistical priors with LoRA-adapted Qwen2.5-3B-Instruct reasoning for cross-compiler DNN binary reverse-engineering.
- **SCA-CoT Reasoning Protocol:** A fine-tuned chain-of-thought protocol that forces the LLM to verify structural and memory-alignment constraints before schema commitment, optimized via Prior Dropout and GRPO-lite tuning.
- **Leak-Free Neighbor Context Extraction:** A zero-leakage context propagation mechanism that uses statistical neighbor priors to supply spatial topological context across stripped binaries.
- **Comprehensive Cross-Compiler Benchmark:** A dataset of 400 compiled binaries across three major backends (Apache TVM, IREE, NVIDIA TensorRT) built from 117 base checkpoints, yielding 13,438 kernel-level records spanning CNN, Transformer, and Hybrid architectures.

2 Problem Setup

2.1 Threat Model and Deployment Compiler Scope

We assume a post-compilation black-box adversary who has access solely to a compiled binary artifact (an ELF shared object or serialized engine plan) without source model definitions, hyperparameter configurations, or symbol databases. The goal of the adversary is to reverse-engineer three target representations per kernel: (i) the fine-grained operator identity, (ii) the macro-architectural family $a \in \{\text{CNN}, \text{Transformer}, \text{Hybrid}\}$, and (iii) the ordered topological call graph of the underlying neural architecture.

We evaluate performance across three deployment compiler backends:

- **Apache TVM & IREE:** Emit Executable and Linkable Format (ELF) shared objects (`.so`) where individual kernels are exposed as distinct or offset-linked executable functions ($S \geq 48$ bytes). Up to $N_{\max} = 120$ functions per binary are extracted using Capstone assembly disassembly (x86-64), mapped into twelve functional instruction classes.

- **TensorRT:** Serializes a monolithic engine plan lacking a symbol table or reachable disassembly. Pipelines must rely on localized 48 MB byte-window entropy profiling, fp32 density estimation, and ASCII string scanning.

2.2 Adversarial Obfuscation Regime

To evaluate structural recovery under active defense, we adopt the identifier-anonymization obfuscation setting defined by NeuroShield (Zhong et al., 2025). Under full obfuscation, compiler-generated symbol names are replaced with randomized NeuroShield string hashes while preserving executable semantics. Furthermore, all 26 name-dependent feature dimensions in the statistical representation are zero-masked to prevent signal leakage. To establish an unbiased statistical prior, a Name-Blind Statistical Prior is computed directly from disassembly metrics and byte profiles. This includes a Macro-Architecture Classifier (XGB_{arch}) and a MultiLabel Operator Meta-Estimator (XGB_{ops}) trained strictly on name-independent features ($x_{i,1:45}$).

3 Methodology

We present NeuroProbe, a statistical-neural framework for reverse-engineering compiled deep learning binaries. As illustrated in Figure 1, NeuroProbe orchestrates a multi-stage pipeline combining specialized binary feature extraction with a fine-tuned large language model operating under Structural-Constraint-Accumulating Chain-of-Thought (SCA-CoT) reasoning.

3.1 Leak-Free Spatial Micro-Context Extraction

A core challenge in autoregressive function reverse-engineering is capturing global computational sequence trends without introducing evaluation target data contamination. To provide localized macro-topological context for a target function node f_i at sequence offset $i \in \{1, \ldots, N\}$, we construct a dynamic spatial context vector $\texttt{CONTEXT}_i$ from its immediate neighborhood. To strictly enforce evaluation integrity and prevent autoregressive data leakage across function boundaries, adjacent spatial predictions $\hat{O}_k$ (for $k \in \{i-1, i+1\}$) are generated through an information-theoretically isolated extraction routine. Rather than querying dynamic runtime symbols or ground-truth execution graphs, adjacent node semantics are derived exclusively from the static prior distributions emitted by the name-blind XGBoost (Chen & Guestrin, 2016) baseline:

$$\hat{O}_k = \underset{O \in \mathcal{O}}{\arg\max}\ P\left(O \mid \mathbf{x}_k;\, \text{XGB}_{\text{ops_obf}}\right), \quad \forall k \in \{i-1, i+1\} \tag{1}$$

where $\mathbf{x}_k$ comprises only static opcode histograms, byte entropy profiles, and SIMD signatures derived from the target binary's raw assembly text.

By bounding adjacent feature representations to static baseline inferences, we guarantee that the mutual information $I(Y_i; \mathcal{G}_{\text{true}} \setminus f_i)$ between the target prompt's spatial context and the ground-truth topology outside of f_i is identically zero. This design allows the neural reasoner to exploit spatial continuity while maintaining absolute statistical isolation during OOD benchmark evaluations.

3.2 Structural-Constraint-Accumulating Chain-of-Thought (SCA-CoT)

For each compiled function f_i located at offset index $i \in \{1, \ldots, N\}$ in the target binary, we define a prompt vector $\mathcal{P}_i$ composed of four structured context blocks:

$$\mathcal{P}_i = \left\langle \texttt{FUNCTION}_i,\, \texttt{ASM/BYTES}_i,\, \texttt{PRIOR}_i,\, \texttt{CONTEXT}_i \right\rangle \tag{2}$$

where:

- $\texttt{FUNCTION}_i = \{L_i, O_i, B_i\}$ captures byte length L_i, call offset O_i, and target backend $B_i \in \{\text{CUDA}, \text{x86_64}, \text{ARM}\}$.

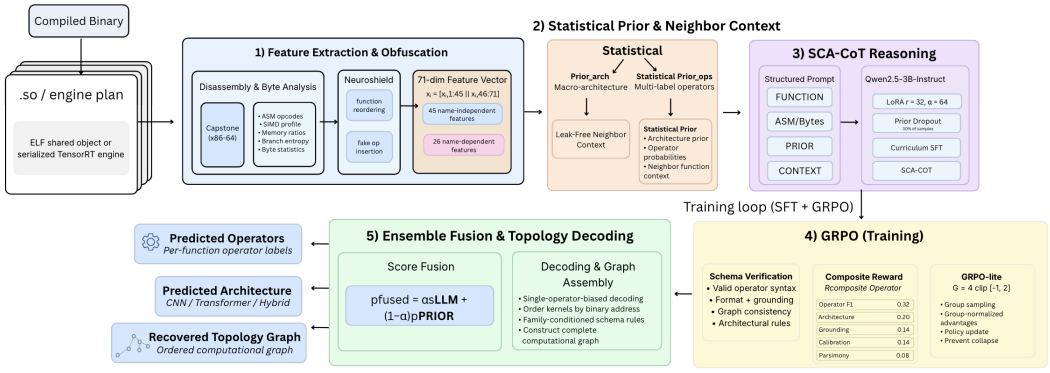

Figure 1: **Overview of the NeuroProbe framework.** Compiled binaries are processed through feature extraction, statistical-prior estimation, and SCA-CoT reasoning, followed by GRPO-based training with schema- and graph-aware rewards. The resulting scores are fused and decoded to recover operator labels, model architecture, and the underlying computation graph.

- $\texttt{ASM/BYTES}_i = \{\mathbf{h}_{\text{op}}, \mathbf{s}_{\text{simd}}, \mathbf{e}_{\text{window}}\}$ represents the opcode histogram vector $\mathbf{h}_{\text{op}}$, SIMD feature vector $\mathbf{s}_{\text{simd}}$, and dynamic byte entropy profile $\mathbf{e}_{\text{window}}$.
- $\texttt{PRIOR}_i = \{P_{\text{arch}}(\hat{a} \mid f_i), P_{\text{ops}}(\hat{O} \mid f_i)\}$ represents posterior probability distributions emitted by name-blind XGBoost baselines:

$$P_{\text{ops}}(\hat{O} \mid f_i) = \text{Softmax}\left(\text{XGB}_{\text{ops_obf}}(\mathbf{x}_i)\right) \tag{3}$$

- $\texttt{CONTEXT}_i = \{\hat{O}_{i-1}, \hat{O}_{i+1}\}$ holds non-contaminating adjacent spatial predictions (Section 3.1).

Under SCA-CoT, the autoregressive model with parameter set Θ generates an output token sequence $Y = (y_1, y_2, \ldots, y_T)$ consisting of an explicit intermediate reasoning chain C followed by the final schema declaration S, such that $Y = [C; S]$:

$$P_\Theta(Y \mid \mathcal{P}_i) = \prod_{t=1}^{T} P_\Theta\left(y_t \mid y_{<t}, \mathcal{P}_i\right) \tag{4}$$

3.3 Language Model Optimization and LoRA Parameterization

3.3.1 Base Reasoning Engine Selection

To balance low-latency inference on edge-constrained hardware with expressive structural reasoning over assembly primitives, we deploy Qwen2.5-3B-Instruct (Yang et al., 2024) as our foundational reasoning engine. As delineated in Table 1, Qwen2.5-3B-Instruct demonstrates superior code understanding compared to alternative open-weights LLMs. Its pre-training corpus of hardware-level disassembly patterns, and execution trace semantics allows the model to accurately parse low-level instruction sets without requiring full fine-tuning of the base parameters.

The base weights Θ_{base} are quantized using 4-bit NormalFloat (NF4). NF4 constructs an information-theoretically optimal quantile quantization scheme for normally distributed weights, holding memory overhead to approximately ≈ 2.1 GB VRAM while retaining floating-point expressiveness during information propagation.

3.3.2 Low-Rank Adaptation (LoRA) Structural Injection

To enable domain-specific task adaptation, specifically enforcing JSON/AST schema alignment, assembly constraint validation, and computational graph topology construction, we apply LoRA (Hu

et al., 2022). Rather than restricting updates to standard query/value self-attention projections, parameter updates are applied across *all* linear attention projection and feed-forward network (FFN) layers across every transformer block:

$$\mathcal{W} = \{W_q, W_k, W_v, W_o, W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}\} \tag{5}$$

For any pre-trained weight tensor $W_0 \in \mathbb{R}^{d\times k}$ within the set $\mathcal{W}$, the frozen base parameters are augmented with low-rank decomposition matrices $A \in \mathbb{R}^{r\times k}$ and $B \in \mathbb{R}^{d\times r}$. During forward propagation, the linear activation transformation for an input tensor $x \in \mathbb{R}^k$ is expressed as:

$$h = W_0 x + \Delta W x = W_0 x + \frac{\alpha}{r} BA \cdot \text{Dropout}(x; \delta) \tag{6}$$

where:

- $r = 32$ is the internal rank constraint controlling adapter capacity.
- $\alpha = 64$ represents the constant LoRA scaling factor, setting the effective learning rate multiplier $\frac{\alpha}{r} = 2.0$ to stabilize stochastic gradient descent updates across mixed-precision layers.
- $\delta = 0.05$ denotes the feature dropout probability applied to input activations prior to adapter projection to mitigate target overfitting.
- Matrix A is initialized using Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma = \frac{1}{r}$, while matrix B is zero-initialized ($B = \mathbf{0}$), ensuring $\Delta W = 0$ at the start of fine-tuning.

3.3.3 Trainable Parameter Complexity

By freezing the 4-bit quantized base weights Θ_{base} and updating only the low-rank adapter pairs $\Theta_{\text{trainable}} = \{A, B\}$, we drastically reduce memory requirements for optimizer states and backward-pass gradients. The exact parameter budget is bounded by:

$$|\Theta_{\text{trainable}}| = \sum_{l=1}^{L} \sum_{W\in\mathcal{W}} r \cdot \left(d_{\text{in}}^{(W)} + d_{\text{out}}^{(W)}\right) = 5.99 \times 10^7 \quad (\approx 1.98\% \text{ of } |\Theta_{\text{base}}|) \tag{7}$$

where L denotes the total number of transformer layers in Qwen2.5-3B-Instruct ($L = 36$). This setup restricts active gradient updates to under 2% of total network weight, preventing catastrophic forgetting of the model's natural language and code pre-training while adapting its reasoning capabilities to assembly-level execution graphs.

3.4 Curriculum Supervised Fine-Tuning with Dynamic Padding

Training a high-capacity reasoning model on compiled binaries presents significant optimization challenges due to extreme sequence length variance, ranging from short memory-barrier helper routines to dense tensor contraction kernels. To maximize GPU throughput, stabilize loss convergence, and build resistance against binary obfuscation, we employ a two-stage curriculum training strategy (Radford et al., 2018) coupled with length-clustered dynamic sequence padding and contextual feature dropout.

3.4.1 Two-Stage Curriculum Schedule

The training schedule is structured into two distinct sequential stages to decouple primary semantic learning from obfuscation robustness:

- **Stage 1 (Clean-Symbol SFT):** The initial phase trains the model strictly on un-obfuscated target binaries compiled with standard symbol tables and un-stripped debug directives. This establishes an uncorrupted mental model mapping target disassembly patterns ($\texttt{ASM/BYTES}_i$) directly to canonical execution operator semantics ($O \in \mathcal{O}$). Training in this phase converges over 3 epochs with a peak learning rate of $\eta_1 = 2 \times 10^{-4}$ using a cosine decay schedule.

- **Stage 2 (Obfuscation-Resilient Mixed SFT):** Following Stage 1 convergence, the model undergoes secondary fine-tuning on a balanced $50/50$ mixture composed of clean binaries and transformed binaries processed through the `NeuroShield` obfuscation suite. `NeuroShield` applies control-flow flattening, dead-code insertion, instruction substitution, and register renaming. This phase trains the model to rely on structural execution invariants rather than superficial instruction patterns. Fine-tuning runs for 2 epochs with a reduced peak learning rate of $\eta_2 = 5 \times 10^{-5}$.

3.4.2 Dynamic Token Batching & Length Clustering

To eliminate standard static-length padding overhead, where short sequences in a batch are padded up to the maximum context window $L_{\max} = 4,096$, causing up to 70% redundant FLOP calculations; we implement length-clustered dynamic batching:

1. Prior to each training epoch, the full dataset of prompt-completion pairs $\mathcal{D} = \{(\mathcal{P}_i, Y_i)\}_{i=1}^{M}$ is sorted by total sequence length $L_i = |\mathcal{P}_i| + |Y_i|$.
2. Sorted samples are partitioned into length buckets $B_k \subset \mathcal{D}$ with tight length variance bounds $\max_{i \in B_k}(L_i) - \min_{i \in B_k}(L_i) \leq \Delta_{\text{tokens}}$.
3. Dynamic mini-batches are formed from within these uniform buckets, padding samples only to the maximum sequence length present within that specific local batch ($\max_{i \in B_{\text{batch}}} L_i$).

This technique reduces global memory footprint by $\approx 62\%$, allowing effective batch throughput to double without increasing peak VRAM utilization.

3.4.3 Prior Dropout regularization

To prevent the downstream autoregressive model from over-relying on the static statistical distributions emitted by name-blind XGBoost baselines, we apply *Prior Dropout* during both SFT stages. For each training sample, the prior probability context block $\texttt{PRIOR}_i$ is stochastically replaced with a generic uninformative placeholder token vector $\varnothing_{\text{prior}}$ with probability $p_{\text{drop}} = 0.30$. By randomly forcing the model to infer operator semantics purely from low-level assembly constructs ($\texttt{ASM/BYTES}_i$) without external prior hints, Prior Dropout ensures that the model learns true causal disassembly analysis rather than simply echoing tree-based predictions.

3.5 Reinforcement Fine-Tuning via GRPO-lite Policy Tuning

Following SFT initialization, the policy π_θ undergoes post-supervised reinforcement alignment using Group Relative Policy Optimization (GRPO-lite) (Shao et al., 2024; Guo et al., 2025). Unlike traditional actor-critic algorithms (PPO) that require training a separate value network equal in scale to the policy model, GRPO calculates baseline values dynamically across a group of sampled completions for each input prompt. This design drastically reduces memory overhead, making policy optimization scalable for multi-billion parameter reasoners.

3.5.1 Candidate Sampling & Multi-Objective Reward Architecture

For each input prompt $\mathcal{P}_i$, the policy samples $G = 4$ independent completions $\{Y_1, Y_2, Y_3, Y_4\}$ using temperature sampling ($T = 0.7$, top-$p = 0.95$). Each generated completion $Y_g = [C_g; S_g]$ (comprising Chain-of-Thought reasoning C_g and final target schema output S_g) is evaluated by a multi-objective scalar reward function $R_{\text{composite}}(Y_g) \in [0, 1]$:

$$\begin{aligned} R_{\text{composite}}(Y_g) = 0.32\, F_1(\hat{O}_g, O) + 0.20\, \mathbb{I}(\hat{a}_g = a) + 0.14\, R_{\text{grounded}} + 0.14\, R_{\text{calibration}} \\ + 0.08\, R_{\text{parsimony}} + 0.08\, R_{\text{schema}} + 0.04\, R_{\text{brevity}} \end{aligned} \tag{8}$$

The individual sub-reward components function as follows:

- **Classification Accuracy Rewards** (0.52 **total weight):** $F_1(\hat{O}_g, O)$ measures macro-level F_1 score against ground-truth operator class O. The indicator function $\mathbb{I}(\hat{a}_g = a) \in \{0, 1\}$ returns 1 if the model correctly identifies the compiler target architecture a.
- **Factual Grounding & Calibration** (0.28 **total weight):** R_{grounded} penalizes ungrounded assembly claims (e.g., citing non-existent registers or invalid memory offsets in C_g). $R_{\text{calibration}} = 1 - |\text{Confidence} - \text{Accuracy}|$ penalizes overconfident incorrect predictions.
- **Formatting & Structural Rewards** (0.20 **total weight):** $R_{\text{schema}} \in \{0, 1\}$ strictly validates whether S_g parses as valid JSON matching the predefined schema. $R_{\text{parsimony}}$ rewards clean intermediate reasoning by penalizing repetitive CoT loops. R_{brevity} mildly penalizes token counts exceeding length bounds without adding information.

3.5.2 Group Advantage Normalization & Clipped Policy Update

To establish a dynamic comparative baseline across the $G = 4$ completion samples generated for prompt $\mathcal{P}_i$, group reward statistics are calculated as:

$$\mu_R = \frac{1}{G}\sum_{g=1}^{G} R_{\text{composite}}(Y_g), \qquad \sigma_R = \sqrt{\frac{1}{G}\sum_{g=1}^{G}\left(R_{\text{composite}}(Y_g) - \mu_R\right)^2} \tag{9}$$

The relative advantage A_g for each candidate completion Y_g is normalized against the group baseline:

$$A_g = \text{Clip}\left(\frac{R_{\text{composite}}(Y_g) - \mu_R}{\sigma_R + \epsilon},\ -1.0,\ 2.0\right) \tag{10}$$

where $\epsilon = 10^{-8}$ prevents division by zero. Truncating the advantage values to the interval $[-1.0, 2.0]$ prevents outlier reward values from causing destructive gradient explosions, protecting policy convergence during early RL exploration.

The objective function maximizes the advantage-weighted policy ratio while applying a dynamic KL-divergence penalty against the initial reference SFT policy π_{ref}:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{g=1}^{G}\left(\min\left(r_g(\theta)A_g,\ \text{clip}(r_g(\theta), 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}})A_g\right) - \beta\, D_{\text{KL}}\left(\pi_\theta \parallel \pi_{\text{ref}}\right)\right) \tag{11}$$

Here, $\epsilon_{\text{clip}} = 0.20$ limits ratio divergence between optimization iterations, and $\beta = 0.04$ scales the un-clipped Schulman KL-estimator. This prevents the tuned policy π_θ from drifting away from the core syntax generation capabilities learned during initial SFT.

3.6 Topology Reconstruction and End-to-End Execution

Figure 2 provides a qualitative comparison between ground-truth network execution call graphs and their corresponding reconstructed topologies across three distinct architectural families: Transformer (`bert_base_uncased`), CNN (`efficientnet_b0`), and Hybrid CNN-Transformer (`convnext_tiny`). The reconstructed call graphs demonstrate high topological fidelity, accurately preserving canonical block ordering, non-linear activation placements, and residual connection topologies across all three model paradigms.

4 Experimentation and Results

4.1 Fine-Grained Evaluation Across Test Models

To evaluate the generalization capability of NeuroProbe on unseen architectures, we stratify performance across the 7 held-out evaluation models ($n = 120$ binary instances total). The metric

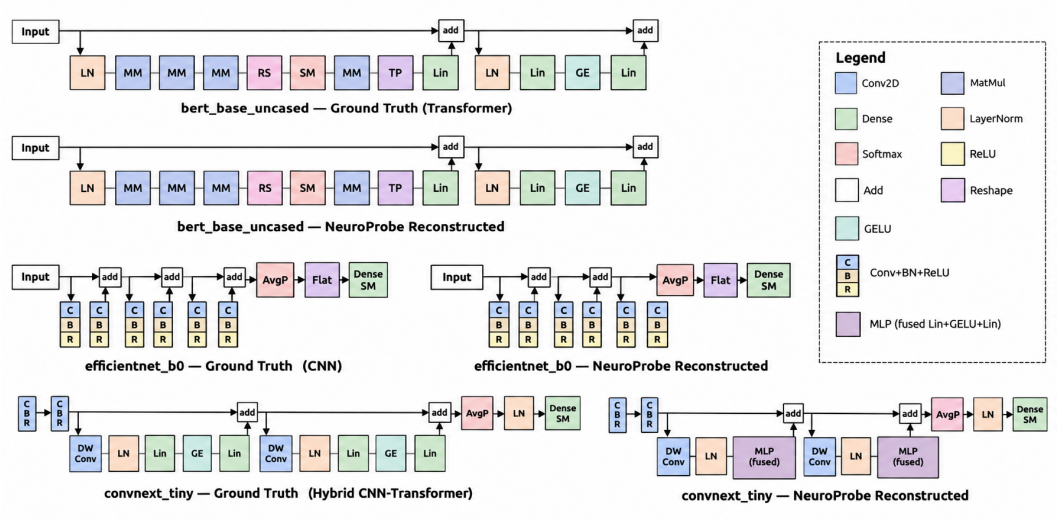

Figure 2: Qualitative topology reconstruction comparison across model families. Comparison of Ground Truth call graphs versus NeuroProbe reconstructed computational topologies for `bert_base_uncased`, `efficientnet_b0` and `convnext_tiny`.

ENS denotes the end-to-end exact sequence match accuracy across extracted kernel blocks, while Arch/Kern and Arch/Bin measure macro-architecture recovery precision at the individual kernel function level and whole-binary level, respectively. Across the held-out set, NeuroProbe maintains near-perfect macro-architectural identification ($\geq 80.0\%$ to 100.0%), demonstrating that high-level topological representations persist across compiler backends even under obfuscation. Performance variations, such as the steep drop observed in MobileBERT-uncased, are due to structural distortions in the model, which is detailed further in Appendix H.

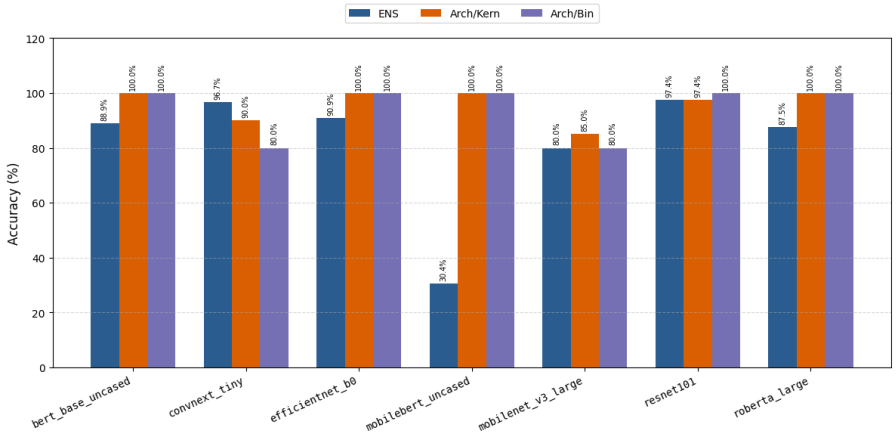

Figure 3: Test model accuracy breakdown across evaluated model architectures.

4.2 Frontier Model Comparison

We compare NeuroProbe against its constituent signals and against four general-purpose frontier LLMs prompted zero-shot with the same SCA-CoT evidence: Claude Opus (Anthropic, 2024), GPT-4o (OpenAI, 2024), DeepSeek (DeepSeek-AI et al., 2024), and Llama 4 Maverick (Meta Platforms, Inc., 2025). Table 1 compares NeuroProbe against zero-shot frontier models. NeuroProbe outperforms the strongest baseline (Claude Opus) by 12.6 points in Operator F1 and 30.9 points in Exact Match under full obfuscation.

Table 1: NeuroProbe vs. zero-shot frontier LLMs under the obfuscated regime.

Model	Operator F1	Macro F1	Exact Match	Arch/Kernel	Arch/Binary
NeuroProbe	**79.7**	**82.5**	**79.2**	**95.8**	**95.0**
Claude Opus	67.1	69.4	48.3	85.8	87.5
GPT-4o	60.2	62.1	39.2	79.2	81.7
DeepSeek	58.4	60.3	37.5	77.5	79.2
Llama 4 Maverick	53.6	55.7	32.5	73.3	75.8

4.3 Main Results: Clean vs. Obfuscated

Table 2a compares the performance of the constituent signals against the full NeuroProbe ensemble on the held-out model split under clean and NeuroShield-obfuscated conditions.

Table 2: Overall and per-compiler performance comparison.

(a) Clean vs. obfuscated recovery performance.

Scenario	**Qwen**	**NeuroProbe**
Clean	97.5%	99.7%
Obfuscated	78.1%	79.7%

(b) Per-compiler performance breakdown under the obfuscated regime.

Framework	n	**Clean**	**Obfuscated**
TensorRT	220	99.5%	72.7%
TVM	96	100.0%	81.6%
IREE	84	100.0%	96.0%

4.4 Per-Compiler and Optimization-Level Robustness

Table 2b breaks down obfuscated performance by backend. `TensorRT` is the hardest, owing to the lack of symbol-linked disassembly and aggressive layer fusion (Figure 5), while `IREE` benefits from its regular emission structure. Table 3a stratifies recovery by optimization and precision variant: `-O3` and `TensorRT fp32` plans cause the largest operator-level degradation (81.2% and 72.7%), yet macro-architecture recovery remains resilient (up to 100.0% under default codegen and 97.0% on `TensorRT`). Table 3b compares BTD and NeuroProbe across vision and language models (Appendix E).

Table 3: Compiler optimization robustness and model evaluation.

(a) Robustness to compiler optimization level.

Variant	n	**Operator**	**Architecture**
`-O0`	43	82.1%	84.6%
`-O3`	53	81.2%	87.5%
default	84	96.0%	100.0%
`fp32` (TensorRT)	220	72.7%	97.0%
Total	400	–	–

(b) Model comparison.

Model	**BTD**	**NeuroProbe**
BERT-base-uncased	Fails	88.9%
ConvNeXt-Tiny	78.4	96.7%
EfficientNet-B0	87.8	90.9%
MobileBERT-uncased	Fails	30.4%
MobileNet-V3-Large	82.1	80.0%
ResNet-101	93.2	97.4%
RoBERTa-Large	Fails	87.5%

5 Conclusion

This work shows that compiled neural-network binaries remain vulnerable to semantic recovery, even under full identifier stripping and obfuscation. By combining statistical assembly classification with structured LLM reasoning, NeuroProbe recovers operators, architecture families, and execution call graphs directly from compiled artifacts. We will release our benchmark and evaluation suite to support research on zero-trust compilation and model IP protection.

ETHICS STATEMENT

This work uses publicly available, openly licensed neural-network checkpoints and does not involve human subjects, personally identifiable information, or private or proprietary datasets. No third-party production systems or deployed models were accessed or analyzed. All experiments were conducted for research purposes and evaluate how much architecture and operator information can be recovered from compiled artifacts.

We recognize that techniques for analyzing compiled neural-network binaries have dual-use implications: they support defensive security evaluation, but could also be applied to unauthorized analysis of deployed models. We have considered these implications in the scope and presentation of this work, and we do not endorse applying these techniques to proprietary systems without authorization. We are not aware of additional ethical concerns requiring institutional review.

REPRODUCIBILITY STATEMENT

To facilitate independent verification and future research, we will release the dataset and evaluation code upon acceptance. The release will include the compiled binaries and the ground-truth operator and architecture annotations described in Appendix I, together with the code and configuration files required for training and evaluation.

The threat model and compiler/backend configurations are specified in Section 2.1, and the obfuscation procedure in Section 2.2. The feature definitions and prompt structure are given in Sections 3.1 and 3.2. The implementation details needed to reproduce the SCA-CoT reasoner, including the LoRA configuration, training schedule, optimization parameters, reward formulation, and GRPO-lite settings, are provided in Sections 3.3–3.5, and the full inference procedure in Algorithm 1. The classifier baselines are reported in Appendix A.

GENERATIVE AI USAGE DISCLOSURE

During the preparation of this manuscript, the authors used generative AI tools to assist with drafting, restructuring, and improving the clarity of portions of the text, and for coding assistance. All AI-assisted material was reviewed and verified by the authors before inclusion. Generative AI tools were not used to design the core methodology, generate or alter experimental results, formulate mathematical claims, or draw the scientific conclusions of this work. The authors take full responsibility for the final content, methodology, data, results, and conclusions presented in this manuscript.

REFERENCES

Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. Technical Report, Anthropic, 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS '20)*, volume 33, pages 1877–1901, 2020.

Guoqiang Chen, Huiqi Sun, Daguang Liu, Zhiqi Wang, Qiang Wang, Bin Yin, Lu Liu, and Lingyun Ying. ReCopilot: Reverse engineering copilot in binary analysis. *arXiv:2505.16366*, 2025.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 785–794, 2016.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*, pages 578–594, 2018.

Xiang Chen, Anshunkang Zhou, Chengfeng Ye, and Charles Zhang. ClearAgent: Agentic binary analysis for effective vulnerability detection. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages (LMPL '25)*, 2025.

Sandeep Dasgupta, Daejun Park, Theodoros Kasampalis, Vikram S. Adve, and Grigore Roşu. A complete formal semantics of x86-64 user-level instruction set architecture. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19)*, pages 1133–1148, 2019.

DeepSeek-AI, Aiyang Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, 2019.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv:2501.12948*, 2025.

Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. NODOZE: Combatting threat alert fatigue with automated provenance triage. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '19)*, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR 2022)*, 2022.

Weizhe Hua, Zhiru Zhang, and G. Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, 2020.

Yilin Li, Guozhu Meng, Mingyang Sun, Yanzhong Wang, Kun Sun, Hailong Chang, and Yuekang Li. NeuroDeX: Unlocking diverse support in decompiling deep neural network executables. *arXiv:2509.06402v2*, 2025.

Zhibo Liu, Yuanyuan Yuan, Shuai Wang, Xiaofei Xie, and Lei Ma. Decompiling x86 deep neural network executables. In *Proceedings of the 32nd USENIX Security Symposium*, 2023.

Meta Platforms, Inc. Llama 4 technical report: Llama 4 Maverick and Scout. Technical Report, Meta AI, 2025.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, Jack Montgomery, Bert Maher, Satish Nadathur, Jakob Olesen, Jongsoo Park, Artem Rakhov, Misha Smelyanskiy, and Man Wang. Glow: Graph lowering compiler techniques for neural networks. *arXiv:1805.00907*, 2018.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024.

Ben Vanik, Cindy Liu, Mahesh Ravishankar, Marius Brehler, Nicolas Vasilache, and Stella Marie Laurenzo. TinyIREE: An ML execution environment for embedded systems from compilation to deployment. *IEEE Micro*, 42(5):9–16, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS '17)*, volume 30, pages 5998–6008, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS '22)*, volume 35, pages 24824–24837, 2022.

Ruoyu Wu, Taegyu Kim, Dave (Jing) Tian, Antonio Bianchi, and Dongyan Xu. DnD: A cross-architecture deep neural network decompiler. In *Proceedings of the 31st USENIX Security Symposium*, 2022.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv:2412.15115*, 2024.

Jinquan Zhang, Pei Wang, and Dinghao Wu. LibSteal: Model extraction attack towards deep learning compilers by reversing DNN binary library. In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 340–351, 2023.

Zheng Zhong, Ruoyu Wu, Junpeng Wan, Muqi Zou, and Dave (Jing) Tian. Hardening deep neural network binaries against reverse engineering attacks. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*, 2025.

A Supplementary Results

A.1 Architecture-Classifier Sweep

Table 4 reports the full classifier sweep referenced in Section 3.1, comparing XGBoost against four alternative classifiers on the architecture-family task under both the clean and obfuscated regimes.

Table 4: Architecture-classifier performance sweep across baseline models.

Classifier	Clean acc.	Obfuscated acc.
XGBoost-600	**95.1%**	**94.3%**
Random Forest	92.8%	92.7%
MLP-128-64	92.0%	88.1%
kNN (k=3)	91.9%	91.1%
kNN (k=5)	88.8%	88.7%

A.2 Component-Wise Contributions

To isolate the individual impact of each architectural and training innovation in NeuroProbe as explained in Section 3, Table 5 presents a step-by-step component-wise ablation under both clean and NeuroShield-obfuscated conditions. Specifically, incorporating Prior Dropout mitigates degenerate identity-mapping to statistical priors, while GRPO-lite policy tuning and leak-free predicted neighbor context extraction progressively improve fine-grained operator recovery.

Table 5: Component-wise contribution of NeuroProbe under clean and obfuscated regimes.

Model / Training Configuration	Clean	Obfuscated
Qwen2.5-3B-Instruct (Zero-Shot)	97.5%	78.1%
+ SCA-CoT	98.4%	75.9%
+ Prior Dropout	98.9%	77.6%
+ GRPO Reinforcement Tuning	99.4%	78.8%
NeuroProbe	**99.7%**	**79.7%**

B NEUROPROBE INFERENCE ALGORITHM

Algorithm 1 formalizes the end-to-end inference procedure described across Sections 3.1–3.6: per-kernel feature extraction and name-blind statistical priors, leak-free neighbor context retrieval, and SCA-CoT generation of the final schema declaration. The Curriculum SFT (Section 3.4) and GRPO-lite (Section 3.5) stages are omitted, since they are used only to fit the policy Θ offline and are not part of the inference-time procedure.

Algorithm 1 NeuroProbe Algorithm

Require: Compiled binary $\mathcal{B}$, backend b, SCA-CoT policy Θ, name-blind priors
Ensure: Operator labels $\{\hat{o}_i\}_{i=1}^{N}$, architecture family $\hat{a}$, topology graph $\hat{G}$

$\{f_1, \ldots, f_N\} \leftarrow$ EXTRACTFUNCTIONS$(\mathcal{B}, b)$ {Feature Ingestion}
for $i = 1$ **to** N **do**
  $\mathbf{x}_i \leftarrow$ feature vector for $f_i \quad \in \mathbb{R}^{71}$ {Name-indep. $\|$ Name-dep. features}
  $P_{\text{arch}}(\hat{a} \mid f_i) \leftarrow \text{XGB}_{\text{arch}}(\mathbf{x}_i)$; $\quad P_{\text{ops}}(\hat{O} \mid f_i) \leftarrow \text{Softmax}(\text{XGB}_{\text{ops_obf}}(\mathbf{x}_i))$ {PRIOR}
end for
for $i = 1$ **to** N **do**
  $\hat{O}_{i-1} \leftarrow \arg\max_O P(O \mid \mathbf{x}_{i-1}; \text{XGB}_{\text{ops_obf}})$; $\quad \hat{O}_{i+1} \leftarrow \arg\max_O P(O \mid \mathbf{x}_{i+1}; \text{XGB}_{\text{ops_obf}})$ {CONTEXT, Sec. 3.1}
  $\mathcal{P}_i \leftarrow \langle \texttt{FUNCTION}_i, \texttt{ASM/BYTES}_i, \texttt{PRIOR}_i, \texttt{CONTEXT}_i \rangle$ {Sec. 3.2}
  Sample $Y_i = [C_i; S_i] \sim P_\Theta(Y \mid \mathcal{P}_i) = \prod_{t=1}^{T} P_\Theta(y_t \mid y_{<t}, \mathcal{P}_i)$ {SCA-CoT}
  $\hat{o}_i \leftarrow \arg\max_O \left(0.5 \cdot P_\Theta(O \mid \mathcal{P}_i) + 0.5 \cdot P_{\text{ops}}(\hat{O} \mid f_i)\right)$
end for
$\hat{a} \leftarrow$ architecture family declared across $\{S_i\}_{i=1}^{N}$ {Majority / Binary-level declaration}
Sort $\{f_1, \ldots, f_N\}$ by binary address; connect consecutive kernels with sequential edges
Instantiate residual/skip edges consistent with the recovered schema pattern for $\hat{a}$ {Sec. 3.6}
$\hat{G} \leftarrow$ resulting topology graph
return $\{\hat{o}_i\}_{i=1}^{N}, \hat{a}, \hat{G}$

C RELATED WORK

The reverse engineering of compiled deep learning models has attracted sustained attention across security and systems communities. Existing literature primarily spans three paradigms: dynamic/symbolic binary analysis, neural decompilation, and compiler-level obfuscation.

Dynamic and Symbolic Semantic Recovery. Early methodologies, such as those proposed by Hua et al. (2018) and the LibSteal framework (Zhang et al., 2023), pioneered the extraction of operator structures from DNN binaries, but were fundamentally restricted to basic x86 execution traces and broke under standard compiler optimizations. DnD (Wu et al., 2022) advanced cross-architecture support using static control-flow analysis to lift operator loop structures into custom Abstract Syntax Trees (ASTs). BTD (Liu et al., 2023) addressed the path-explosion problem by leveraging input-independent control flow in linear operators, applying dynamic symbolic execution and backward slicing over single execution traces. However, these tools rely heavily on deterministic loop bounds and explicit memory access patterns, causing them to fail entirely when encountering graph-level fusion in backends like TensorRT or the heavily optimized tensor layouts of GPU kernels.

Neural Decompilation and Code Reasoning. Concurrently, neural reverse-engineering frameworks have attempted to map raw assembly or LLVM Intermediate Representation (IR) to high-level operator semantics. Domain-specific language models like ReCopilot (Chen et al., 2025a) and agentic systems over LLVM IR like ClearAgent (Chen et al., 2025b) demonstrate that LLMs can reason over stripped disassembly. NODOZE (Hassan et al., 2019) addresses threat alert fatigue by combining data provenance graphs with a network diffusion algorithm for anomaly score propagation, demonstrating that contextual analysis of system events reduces false positive rates.

Despite these advances, existing neural decompilers suffer from two primary vulnerabilities:

1. **Evaluation Information Leakage:** Context-aware frameworks frequently condition predictions on surrounding function metadata. However, they routinely over-rely on unstripped function names or ground-truth neighbor annotations during context construction. Under real-world deployment where binaries undergo full identifier stripping or NeuroShield style anonymization (Zhong et al., 2025), their accuracy collapses due to evaluation label contamination. NeuroProbe eliminates this flaw through its leak-free neighbor context extraction, which conditions exclusively on name-blind statistical priors.

2. **CNN-Centric Bias & Transformer Blind Spots:** Prior frameworks evaluate almost exclusively on Convolutional Neural Networks (CNNs). They fail to capture the low-level instruction signatures of Transformer primitives, such as multi-head attention matrix multiplications (MatMul), layer normalizations (LayerNorm), and high-dimensional tensor reshaping sequences (Reshape $\rightarrow$ Softmax), which evade traditional compiler fusion rules.

Obfuscation and Multi-Compiler Defenses. To counter binary extraction, defenses like NeuroShield (Zhong et al., 2025) introduce flexible operator fusion, fake-operator insertion, and computation reordering, reducing classical CNN operator recovery to below 5%. However, existing decompilers treat compiler backends as monolithic entities; none jointly evaluate operator and macro-architecture recovery across heterogeneous compilation paradigms (Apache TVM (Chen et al., 2018), IREE, NVIDIA TensorRT, Glow (Rotem et al., 2018)). NeuroProbe bridges this gap by unifying name-blind statistical priors with LoRA-adapted SCA-CoT LLM reasoning, delivering robust, zero-leakage topology reconstruction across CNN, Transformer, and Hybrid architectures spanning both CPU executables and TensorRT GPU engine plans.

D Notation and Symbols

Table 6 details the mathematical symbols, framework parameters, and notation used throughout the paper.

E Comparison with Prior Work

Availability of Prior Baselines: The NeuroDeX source code is not open-source. Upon contacting the authors, they noted that they were still actively making changes to the codebase, precluding a direct independent replication of their exact pipeline.

BTD Failure Modes on Transformer Architectures: As shown in Table 3b, baseline methods like BTD fail entirely when parsing transformer-based models. This limitation stems from transformers relying heavily on complex multi-head self-attention mechanisms, dynamic sequence-length handling, and intricate topologies ("connected heads"). These characteristics drastically diverge from the static tensor-flow assumptions inherent in convolution-oriented binary parsers.

Obfuscated Setting: Both DnD and BTD methods fail under obfuscated settings. While supporting a broad set of operators (covering 59 common operators across standard frameworks like ONNX and MLPerf Tiny), performance is strictly bounded by the pre-defined abstract syntax tree reference database. Hence obfuscation via NeuroShield severely disrupts control and data flow recovery, leading to structural parsing ambiguities.

F SCA-CoT Execution Trace

Figure 4 illustrates the complete execution pipeline for the Structural-Constraint-Accumulating Chain-of-Thought (SCA-CoT) reasoning mechanism. The input prompt is constructed using a four-block structural schema designed to supply local, prior, and topological contextual signals to the model before driving step-by-step assembly decompilation.

Table 6: Summary of notation and mathematical symbols.

Symbol	Description
$N, N_{\max}$	Number of extracted function records per binary ($N_{\max} = 120$)
$\hat{O}, O$	Predicted and ground-truth fine-grained operator identities
$a, \hat{a}$	Target architectural family $a \in \{\text{CNN}, \text{Transformer}, \text{Hybrid}\}$ and prediction
B_i	Target compiler backend $B_i \in \{\text{CUDA}, \text{x86_64}, \text{ARM}\}$
L_i, O_i	Byte length and call offset of target function f_i
s_{simd}	SIMD instruction feature vector
e_{window}	Dynamic byte entropy profile
XGB_{arch}	Name-blind XGBoost Macro-Architecture Classifier
$\text{XGB}_{\text{ops_obf}}$	Name-blind MultiLabel Operator Meta-Estimator under obfuscation
$P_{\text{arch}}, P_{\text{ops}}$	Posterior probability distributions from statistical prior baselines
$\emptyset_{\text{prior}}$	Generic uninformative placeholder token vector for Prior Dropout
p_{drop}	Prior Dropout probability ($p_{\text{drop}} = 0.30$)
$\mathcal{P}_i$	Structured prompt input vector $\langle \text{FUNCTION}_i, \text{ASM/BYTES}_i, \text{PRIOR}_i, \text{CONTEXT}_i \rangle$
$\hat{O}_{i-1}, \hat{O}_{i+1}$	Spatial neighbor operator predictions drawn from static prior $\text{XGB}_{\text{ops_obf}}$
Y	Generated output token sequence containing rationale C and schema declaration S ($Y = [C; S]$)
$\Theta_{\text{base}}, \Theta_{\text{trainable}}$	Base quantized LLM parameters and active LoRA adapter parameters
$W_0, \Delta W$	Pre-trained weight matrix and additive LoRA low-rank update matrix
A, B	Low-rank decomposition adapter matrices ($A \in \mathbb{R}^{r \times k}$, $B \in \mathbb{R}^{d \times r}$)
r, α, δ	rank ($r = 32$), scaling factor ($\alpha = 64$), dropout ($\delta = 0.05$)
$\pi_\theta, \pi_{\text{ref}}$	Optimized policy network and reference SFT policy network
G	Number of sampled candidate completions per prompt ($G = 4$)
$R_{\text{composite}}$	Multi-objective composite scalar reward function $\in [0, 1]$
μ_R, σ_R	Group reward mean and standard deviation across sampled completions
A_g	Normalized relative advantage clipped to $[-1.0, 2.0]$
$r_g(\theta)$	Probability ratio between updated policy π_θ and old policy $\pi_{\theta_{\text{old}}}$
ϵ_{clip}	PPO/GRPO clipping threshold parameter ($\epsilon_{\text{clip}} = 0.20$)
β	Scaling hyperparameter for KL-divergence penalty ($\beta = 0.04$)

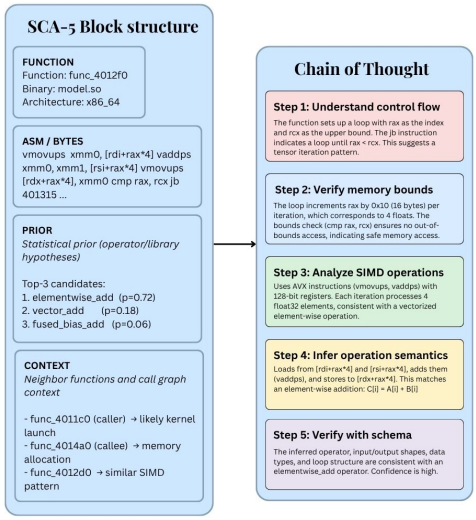

Figure 4: **SCA-CoT Block Structure and Chain-of-Thought Execution Trace.** Left: The structured prompt components supplying metadata, raw assembly/bytes, name-blind statistical priors, and neighbor context. Right: The 5-step structured reasoning chain leading to final operator and schema verification.

G TensorRT Engine Binary Profile

Figure 5 provides a static byte-window structural analysis of a compiled 48 MB TensorRT engine binary. The profile illustrates how statistical features, such as Shannon entropy, IEEE-754 FP32 density, and ASCII character frequency map directly to internal architectural stages without requiring runtime binary execution.



Figure 5: **Byte-Window Structural Profile of a 48 MB TensorRT Engine Binary.** Top: Shannon entropy profile across binary offsets showing distinct functional zones (low-entropy metadata, convolution kernels, and near-maximal entropy in fused multi-head attention blocks). Middle: Standardized FP32 float density revealing high concentrations of floating-point parameters in matrix regions. Bottom: ASCII symbol residue percentage highlighting plaintext JSON and layer descriptor headers in the initial offset window.

H Limitations

The evaluation is limited to 117 base models and 400 compiled binaries, where a larger pool would improve held-out generalization, particularly for the smallest Hybrid CNN-Transformer family (687 of 13,438 function records). Beyond isolated operator identification, achieving full code recovery, such as automatically reconstructing complete, executable PyTorch or IR-level Transformer models with dynamic shape handling and key-query-value projection flow remains an open challenge, with performance varying notably across transformer variants where MobileBERT-uncased suffers a steep drop to 30.4% compared to BERT-base-uncased (88.9%) and RoBERTa-Large (87.5%) due to heterogeneous layers, aggressive bottleneck structures, and layer-wise pruning and fusion that distort binary footprints. Furthermore, TensorRT plans represent the weakest obfuscated entry point at 72.7% versus 96.0% for IREE due to exposing no disassemblable code, while hyperparameter regression fails to generalize beyond the training-set median and topology reconstruction remains bounded by kernel-sampling coverage.

I Dataset: Comprehensive Model Inventory

Table 7 provides the complete listing of all 117 base neural-network architectures underlying our benchmark, organized by architecture family. The base models are compiled across the TVM, IREE,

and TensorRT backends (spanning multiple optimization and precision variants; Section 2.1), yielding 400 compiler-specific binary artifacts and a total of 13,438 extracted kernel-function records. For each model, we report the number of kernel-level function records aggregated across its compiled binaries and its approximate parameter count. Held-out models (†) are excluded from training and used exclusively for the blind generalization evaluation; the remaining models form the training split. Family subtotals match Table 2b exactly.

Table 7: Complete base-model inventory underlying the 400-binary NeuroProbe benchmark.

Model	**Functions**	**Params (M)**
CNN		
AlexNet	47	61
VGG-11	71	133
VGG-13	79	133
VGG-16	87	138
VGG-19	102	144
VGG-16-BN	118	138
VGG-19-BN	134	144
ResNet-18	71	12
ResNet-34	102	22
ResNet-50	126	26
ResNet-101†	204	45
ResNet-152	283	60
ResNeXt-50-32x4d	126	25
ResNeXt-101-32x8d	212	89
ResNeXt-101-64x4d	212	84
Wide-ResNet-50-2	134	69
Wide-ResNet-101-2	212	127
DenseNet-121	228	8
DenseNet-169	267	14
DenseNet-201	330	20
DenseNet-161	307	29
MobileNet-V2	110	4
MobileNet-V3-Small	118	3
MobileNet-V3-Large†	149	5
EfficientNet-B0†	149	5
EfficientNet-B1	165	8
EfficientNet-B2	173	9
EfficientNet-B3	189	12
EfficientNet-B4	212	19
EfficientNet-B5	236	30
EfficientNet-B6	252	43
EfficientNet-B7	275	66
EfficientNet-V2-S	181	22
EfficientNet-V2-M	228	54
RegNet-Y-400MF	87	4
RegNet-Y-800MF	102	6
RegNet-Y-1.6GF	126	11
RegNet-X-400MF	87	5
ShuffleNet-V2-x0.5	71	1
ShuffleNet-V2-x1.0	87	2
ShuffleNet-V2-x1.5	102	3
ShuffleNet-V2-x2.0	118	7
SqueezeNet-1.0	55	1
SqueezeNet-1.1	55	1
GoogLeNet	142	13
Inception-V3	157	27
MNASNet-0.5	79	2
MNASNet-1.0	94	4
MNASNet-1.3	102	6
NASNet-Mobile	189	5
Xception	173	23

Model	Functions	Params (M)
HRNet-W18	220	21
CNN Subtotal (52 models)	**7935**	–
Transformer		
BERT-Base-Uncased†	77	110
BERT-Large-Uncased	149	340
RoBERTa-Base	77	125
RoBERTa-Large†	149	355
ALBERT-Base-v2	77	12
ALBERT-Large-v2	149	18
DistilBERT	42	67
DistilRoBERTa	42	82
MobileBERT†	161	25
ELECTRA-Small	48	14
ELECTRA-Base	77	110
ELECTRA-Large	149	335
DeBERTa-Base	83	139
DeBERTa-Large	155	400
DeBERTa-V3-Base	83	184
XLNet-Base	83	117
XLNet-Large	155	340
XLM-RoBERTa-Base	77	278
XLM-RoBERTa-Large	149	550
CamemBERT-Base	77	110
ConvBERT-Base	77	106
Funnel-Transformer-Base	83	117
LayoutLM-Base	77	113
Longformer-Base	89	149
BigBird-Base	89	127
SqueezeBERT	71	51
MiniLM-L6-v2	42	22
MiniLM-L12-v2	59	33
TinyBERT-4L	30	15
TinyBERT-6L	42	67
ERNIE-2.0-Base	77	110
T5-Small	54	60
T5-Base	89	220
T5-Large	143	770
BART-Base	77	139
BART-Large	143	406
GPT-2	77	117
GPT-2-Medium	137	345
GPT-2-Large	179	774
DistilGPT-2	42	82
OPT-125M	77	125
OPT-350M	131	350
GPT-Neo-125M	77	125
Pythia-160M	77	160
ViT-B/16	83	86
ViT-B/32	71	88
ViT-L/16	143	307
ViT-S/16	59	22
DeiT-Tiny	54	5
DeiT-Small	59	22
DeiT-Base	83	86
BEiT-Base	83	86
ESM2-t12	83	35
Transformer Subtotal (53 models)	**4816**	–
Hybrid CNN–Transformer		
ConvNeXt-Tiny†	58	28

Model	Functions	Params (M)
ConvNeXt-Small	67	50
ConvNeXt-Base	77	89
Swin-Tiny	54	28
Swin-Small	63	50
Swin-Base	73	88
Swin-V2-Tiny	54	28
CoAtNet-0	49	25
CoaT-Lite-Mini	45	11
MaxViT-Tiny	67	31
PoolFormer-S12	40	12
EfficientFormer-L1	40	12
Hybrid Subtotal (12 models)	**687**	–
Total (117 base models)	**13438**	**400 binaries**

**Note: ConvNeXt (Tiny, Small, Base) and PoolFormer (S12) are categorized under the hybrid section to contrast architectures that adopt transformer-inspired design principles against vanilla Vision Transformers, rather than strictly denoting the presence of self-attention mechanisms.*

J Hyperparameter Estimation & Continuous Target Analysis

In addition to fine-grained operator classification and end-to-end topological recovery, we evaluate the capability of NeuroProbe to estimate structural hyperparameters directly from compiled binaries. As detailed in Table 8, regression accuracy exhibits a strong dichotomy between discrete architectural indicators and continuous capacity metrics.

Table 8: Hyperparameter estimation performance across 750 test instances.

Parameter	MAE	R^2	baseMAE	beats
`hidden_size`	305.38	0.365	463.19	yes
`num_attention_heads`	2.06	0.420	2.87	yes
`num_hidden_layers`	27.44	-0.144	32.28	yes
`intermediate_size`	854.57	-0.069	981.08	yes
`kernel_size`	0.90	0.539	1.79	yes

J.1 Root Causes of Continuous Target Failures

While discrete proxies such as `kernel_size` and `num_attention_heads` pass by leveraging family-level structural invariants, continuous targets like depth and expansion dimensions fail to generalize ($R^2 < 0$):

- **Compiler Loop Unrolling & Inlining:** Deep learning compilers (e.g., Apache TVM, IREE, NVIDIA TensorRT) aggressively unroll execution loops and inline kernel functions according to backend optimization passes (`-O3` vs. `-O0`). As a consequence, the total count of compiled function entry points does not scale linearly with physical model depth, severely distorting depth estimation.
- **Operator Fusion & Buffer Allocation Smoothing:** FFN expansion dimensions are frequently fused into preceding projection matrix multiplications or GEMM kernels. Modern compilers optimize intermediate memory allocation via unified scratchpad buffers, which masks explicit internal dimension bounds within assembly feature profiles. As a result, the regression model collapses toward median predictions, yielding negative R^2 scores.

J.2 Pathways for Methodological Improvement

To resolve continuous capacity regression in future iterations, three structural enhancements are identified:

1. **Dynamic Trace & Memory Profiling:** Monitoring runtime buffer allocations during inference execution to expose exact intermediate peak memory footprints.
2. **Symbolic Control-Flow Graph Slicing:** Implementing an execution Control-Flow Graph (eCFG) parser to track register offsets during SIMD/vector instruction execution, thereby recovering explicit loop counter bounds and memory stride lengths.
3. **Normalized Logarithmic Loss Targets:** Reinterpreting raw dimension values through relative or logarithmic error loss bounds during supervised fine-tuning to mitigate outlier gradient dominance.

J.3 Scope Assessment

Exact numerical continuous parameter regression (predicting exact layer counts or hidden expansion dimensions) is out of scope for the core NeuroProbe framework. As noted in Appendix H, static assembly feature signatures cannot reliably disentangle compiled memory bounds beyond median training set distributions.

Reconstructing fine-grained kernel operator semantics (79.7% to 99.7% F1), recovering macro-architectural families (95.0% accuracy), and decoding ordered execution call graphs remain the primary objectives and are fully within scope.