Data Engineering and Applications

Research Paper

Edge-Semantic Perceiver IO for Firmware Backdoor Detection Using Binary Function Call Graphs

Takács Ádám[1, *] and Juhász Imre[1]

[1] Faculty of Science and Informatics, University of Szeged, Szeged, 6720, Hungary
*Corresponding author: takacs.a@inf.u-szeged.hu

Abstract. Firmware backdoors are difficult to detect after compilation because malicious behavior may be confined to a small set of functions and remain hidden after extensive changes to instruction layout. This paper proposes ESPI-FBD, an edge-semantic Perceiver IO framework that analyzes binary function call graphs while avoiding quadratic interaction among all recovered functions. The method integrates architecture-normalized function descriptors, typed directional call-relation biases, salience-seeded latent vectors, latent self-attention, and separate output queries for firmware classification and suspicious-function ranking. A product-disjoint evaluation protocol is specified for a multi-architecture firmware corpus and covers detection, localization, cross-architecture transfer, robustness, calibration, latency, and memory. The framework is designed to preserve sparse malicious evidence in a bounded latent workspace and to return an analyst-oriented review queue in addition to an image-level decision. The numerical values currently reported are illustrative placeholders that define the intended evaluation and must be replaced with audited experimental outputs before submission.

Keywords: *Firmware Backdoor Detection, Binary Function Call Graph, Perceiver IO, Edge Semantics, Latent Attention, Cross-Architecture Generalization*

Received on 20 February 2022, Accepted on 31 May 2022, Published on 14 June 2022

Introduction

Network devices, industrial gateways, cameras, and embedded controllers are all managed by firmware at a high privilege level, and a hidden command channel can get around the host operating system's security. Semantic binary analysis seeks to identify program behaviour that can tolerate such surface changes; static signatures identify recognised byte patterns but deteriorate following recompilation, packing, or instruction replacement [1]. Without the need for source code, a binary function call graph illustrates the interprocedural structure by employing functions as nodes and resolved or inferred calls as directed edges [2]. Backdoors often only make minor changes to this structure, such as the initialisation procedure conditionally registering a covert service, a network handler calling a shell wrapper, or a parser reaching a concealed authentication bypass [3]. This link cannot be captured by traditional function-level classifiers, which are unable to independently award scores [4]. Firmware graphs are irregular, architecture-dependent, and frequently comprise thousands of functions, which makes whole-graph learning attractive [5].

Graph neural networks have been used for malware analysis, program classification, and vulnerability detection by aggregating neighbourhood data [6]. Because message transmission smoothes the representation at multiple layers, the dominating benign graph may dilute a small harmful path. Although dense self-attention necessitates quadratic pairwise interactions and is hence costly for large statically connected images [8], attention-based graph encoders assign adaptive weights to neighbours [7]. Although binary lifting minimises instruction set differences, it causes recovery issues at tail calls and indirect calls [9]. Architecture-specific traits may be suppressed by domain adaptation [10], but processor-dependent signs of questionable behaviour may also be eliminated by an excessively strong alignment. Another kind of interface is called Perceiver IO, which uses query-

driven decoding to produce output after reading an arbitrary input set via cross-attention [11]. Although the original formulation does not encode call direction, linkage semantics, disassembly confidence, or sparse backdoor evidence, the computation primarily depends on the latent breadth rather than on each pair of input functions.

Taken together, these limitations motivate a model that retains directed call semantics and sparse security evidence without applying dense pairwise attention to every recovered function. ESPI-FBD is an edge-semantic Perceiver IO model for firmware backdoor detection using binary function call graphs. It combines typed caller-callee relation biases, salience-aware latent initialization, architecture-normalized function tokens, and separate output queries for firmware classification and function localization. The engineering objective is to preserve the small number of graph relations that may reveal a concealed execution path while maintaining bounded inference memory for large firmware images. Unlike conventional graph encoders that repeatedly propagate information through all recovered neighborhoods, ESPI-FBD reads an irregular function set into a fixed latent workspace and performs most deep reasoning independently of graph size. Its output interface separates firmware-level screening from evidence ranking, allowing a scanner to return both an image-level probability and a limited queue of functions for reverse engineering. The proposed validation protocol covers baseline comparison, unseen-architecture transfer, ablation, graph corruption, calibration, latency, and memory. All numerical results in this draft are illustrative and must be replaced with audited measurements before submission. The model is intended as a filtering and prioritization component rather than an autonomous attribution system, because statistical classification cannot eliminate disassembly uncertainty or trigger-dependent behavior.

Section 2 reviews binary firmware representation and graph-based security learning. Section 3 presents the input representation, latent reasoning mechanism, query decoder, and optimization objective. Section 4 describes the evaluation protocol and intended analyses. Section 5 summarizes the contributions, limitations, and future research directions.

Related Work

Binary Firmware Representation and Backdoor Analysis

Unpacking, identifying the executable, identifying the architecture, disassembling, and recovering call connections are the first steps in static firmware analysis. Although reachable code is preserved, indirect targets may be missed by recursive traversal [12]. More bytes are recovered via linear sweep, which can also interpret the data as instructions [13]. In order to facilitate compiler comparison, intermediate representations standardise register names and instruction formats [14]. Local instruction semantics are summarised by function embeddings [15], and externally observable features like process creation, credential access, socket handling, etc. are exposed by imported APIs. [16] Until a suspect routine occurs normally, these representations remain incomplete. Only then are the caller, guard condition, and privileged callee taken into account. The relational context is provided by a binary function call graph, although systematic noise is introduced due to unresolved edges and library code [17].

String indications, sensitive APIs, control-flow abnormalities, taint routes, or learnt byte features are typically combined in backdoor detection techniques. API listings disregard the execution environment; dynamic analysis sees actual behaviour but is environment-dependent and lacks triggers; and string evidence is inexpensive but simple to conceal. Consequently, the engineering requirement is a ranking system that links a firmware-level decision to a limited number of operations and requires verification rather than a replacement for reverse engineering.

Graph Learning and Latent Attention for Binary Programs

Graph convolution is effective for sparse call graphs and uses shared aggregation to spread local context [18]. Graph attention highlights security-sensitive edges and gives neighbours data-dependent weights [19]. Although deep propagation still blends benign and malicious regions, graph isomorphism networks improve structural discrimination [20]. Global interactions and structural encodings are introduced via transformer-style graph models [21]. Position schemes for molecular or social networks do not explicitly display caller-callee

relationships, import linkages, or recovery confidence, and their memory usage rises quickly with the number of functions.

Piplai et al. [22] By cross-attending input tokens into a small workspace, latent-array architectures isolate the input size from the primary reasoning depth. This feature allows firmware to process both a large router image with the same latent size and a small bootloader. Selection is the remaining piece: consistent cross-attention will distribute capacity to common runtime functions, and atypical call motifs, network-to-command pathways, or infrequent privilege switches provide valuable information. By including query-conditioned evidence decoding, risk-seeded latents, and typed edge bias into the input reading, ESPI-FBD tackles the issue. The principal distinction is therefore not the use of latent attention alone, but the joint integration of confidence-aware directed relations, risk-seeded latent allocation, and query-based function localization within a single firmware-screening architecture.

Proposed Edge-Semantic Perceiver IO Model

Binary Function Call Graph Encoding

The pipeline starts after firmware unpacking and executable deduplication. For each executable, the disassembler recovers functions, direct calls, import stubs, and confidence scores for inferred indirect targets. Library signatures collapse known runtime routines into typed external nodes rather than removing them, because calls to process, file, credential, and networking services carry security meaning. Every function token concatenates a 128-dimensional normalized instruction embedding, twelve control-flow statistics, eight string and constant indicators, sixteen imported-capability flags, log-scaled function size, in-degree, out-degree, and an architecture code. Architecture-specific mnemonics are mapped to operation families before embedding, while operands are categorized as registers, immediate values, memory references, or symbols. This design reduces vocabulary fragmentation without assuming that different instruction sets have identical semantics.

For reproducibility, each variable-length instruction sequence is encoded by the same operation-family encoder and pooled into a 128-dimensional function vector. Missing strings and imported capabilities are represented by explicit absence indicators rather than silently omitted fields, and all scalar statistics are normalized using training-partition parameters only.

Let the recovered binary function call graph contain N function nodes and a directed edge set $\mathcal{E}$. The initial representation of each function combines semantic features, structural descriptors, and a learned architecture condition. The projection is defined as

$$\mathbf{x}_i = \mathbf{W}_f[\mathbf{h}_i \| \mathbf{s}_i \| \mathbf{a}_i] + \mathbf{b}_f, i = 1, \dots, N \quad \text{Eq.(1)}$$

Here, $\mathbf{h}_i$ contains normalized instruction and capability features, $\mathbf{s}_i$ contains graph statistics, and $\mathbf{a}_i$ denotes the processor and compiler condition. The projected token $\mathbf{x}_i$ is consumed by the edge-semantic input-attention module. Removing the architecture condition forces the projection to encode incompatible operand distributions and calling conventions in the same coordinates. This omission is therefore expected to increase false alarms when the model processes firmware from an unseen processor family.

Each recovered call is assigned a relation type covering direct internal calls, inferred indirect calls, imported calls, tail calls, callback registrations, and unresolved candidate edges. Call direction is preserved through separate caller and callee roles. A confidence value down-weights uncertain edges without deleting them, because indirect dispatch may contain the behavior required to identify a covert command path. The edge representation is calculated as

$$\mathbf{r}_{ij} = \mathbf{W}_t \mathbf{e}_{\mathrm{type}(i,j)} + \mathbf{W}_d \mathbf{e}_{\mathrm{dir}(i,j)} + \mathbf{w}_c \log(\varepsilon + c_{ij}) \quad \text{Eq.(2)}$$

Relation labels are assigned by a fixed precedence rule: verified imported and direct internal calls take priority, followed by tail calls, callbacks, inferred indirect calls, and unresolved candidates. Each edge receives one primary type, while recovery confidence is clipped to the interval from zero to one before transformation.

The vector $\mathbf{r}_{ij}$ becomes an attention bias rather than the input of a separate message-passing layer. Its type component distinguishes, for example, an internal helper call from an imported process-execution call. Its direction component prevents the model from treating a caller and its callee symmetrically, while c_{ij} represents the confidence assigned during call-graph recovery. The resulting relation vector is passed directly to the latent

reader. Without this relation bias, functions with similar local instructions but different invocation contexts become difficult to separate, increasing missed detections for authorization-bypass and hidden-service patterns.

Risk salience is estimated from capability rarity, caller-callee novelty, privilege transition, network reachability, and disassembly uncertainty. The score is not treated as a rule-based backdoor decision. Instead, it controls how much latent-reading capacity is allocated to functions that could otherwise be overwhelmed by initialization routines, library wrappers, and other high-frequency benign structures:

$$\rho_i = \sigma(\mathbf{w}_r^\top [\mathbf{u}_i \| \mathbf{q}_i \| \mathbf{p}_i \| \mathbf{n}_i] + b_r) \qquad \text{Eq.(3)}$$

Each salience component is standardized within the training partition before concatenation. Capability rarity and neighborhood novelty are computed from training-corpus frequencies, whereas recovery quality and privilege semantics are derived from the disassembler and capability taxonomy. The projection parameters are learned jointly with the detector.

The components $\mathbf{u}_i, \mathbf{q}_i, \mathbf{p}_i$, and $\mathbf{n}_i$ encode capability rarity, recovery quality, privilege semantics, and neighborhood novelty, respectively. The bounded score ρ_i seeds a subset of latent vectors and subsequently regularizes function localization. Its bounded form allows the neural model to overturn a misleading heuristic score. The output is consumed by the initialization operator and the localization query. If salience estimation is removed, attention mass tends to concentrate on benign functions with high graph degrees.

The complete processing path in Figure 1 connects firmware unpacking and executable selection with disassembly, typed call recovery, token construction, latent reasoning, firmware scoring, and ranked-function generation. The edge-confidence channel shows how uncertain indirect targets remain available to the detector without receiving the same weight as verified direct calls. The workflow therefore preserves potentially important call relations while limiting the influence of graph-recovery noise.

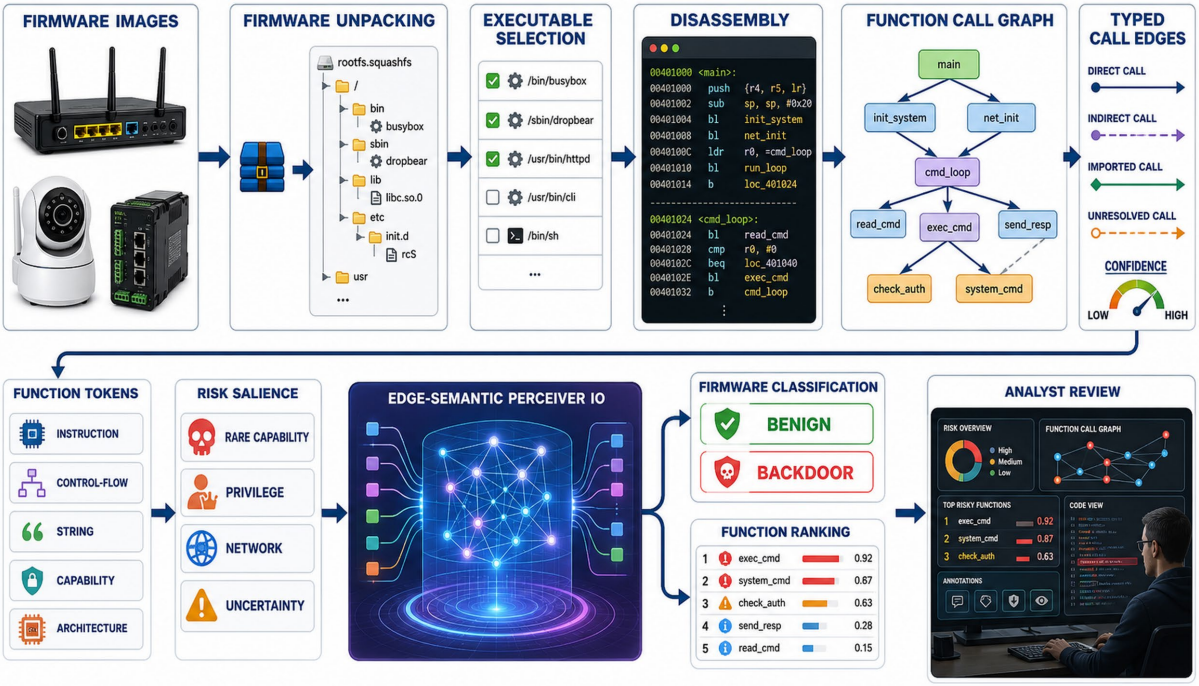

Figure 1. End-to-end workflow from firmware extraction and binary function call graph recovery to ESPI-FBD classification and analyst-oriented function ranking.

Edge-Semantic Perceiver IO

A fixed array of M latent vectors provide bounded computation for graphs containing widely different numbers of functions. The first $M/4$ latent vectors are initialized from salience-weighted function summaries, while the remaining vectors are learned global prototypes. This division gives the model immediate access to firmware-specific risk evidence while maintaining stable representation capacity across samples. The initialization is expressed as

$$\mathbf{z}_m^0 = \mathbf{z}_m^{\text{learn}} + \mathbb{I}[m \le M/4] \sum_{i=1}^{N} \alpha_{mi}\mathbf{x}_i, \alpha_{mi} = \frac{\exp(\rho_i/\tau_m)}{\sum_{j=1}^{N} \exp(\rho_j/\tau_m)} \quad \text{Eq.(4)}$$

The temperature τ_m varies among the seeded latent vectors. Low-temperature vectors focus sharply on rare highsalience functions, whereas higher-temperature vectors collect broader contextual evidence. The initialized array $\mathbf{Z}^0$ is passed to input cross-attention. Removing risk-aware initialization does not alter asymptotic complexity, but it delays the acquisition of sparse malicious evidence and destabilizes localization when backdoor functions occupy less than one percent of the graph.

Standard Perceiver input attention compares latent queries with function keys. ESPI-FBD extends this operation with an aggregated incident-edge term and a graph-distance embedding. For latent vector m and function i, the read score is

$$A_{mi} = \frac{(\mathbf{W}_q\mathbf{z}_m)^\top(\mathbf{W}_k\mathbf{x}_i)}{\sqrt{d}} + \boldsymbol{\beta}_m^\top \sum_{j:(i,j)\in\mathcal{E}} \mathbf{r}_{ij} + \boldsymbol{\gamma}_m^\top \mathbf{d}_i \quad \text{Eq.(5)}$$

The edge term introduces typed outgoing-call behavior into the read operation, while $\mathbf{d}_i$ encodes graph distance from network-entry, authentication, and process-control anchors. The resulting attention weights transfer value projections of $\mathbf{x}_i$ into the latent workspace. This mechanism avoids explicit interactions between every pair of functions. Its inputreading complexity is $O(NMd)$, while the complexity of L latent-processing layers is $O(LM^2d)$. If the edge term is removed, the reader approaches a bag of independent function descriptors and becomes less sensitive to a benignlooking parser that invokes a privileged routine through an anomalous call path.

Incident-edge vectors are aggregated with confidence-weighted degree normalization so that the magnitude of the bias does not grow solely because a function has many recovered neighbors. The same normalization is used during training and inference.

The input-reading output is normalized through a residual connection:

$$\mathbf{Z}^1 = \text{LayerNorm}(\mathbf{Z}^0 + \text{softmax}(\mathbf{A})\mathbf{V}_x) \quad \text{Eq.(6)}$$

The representation $\mathbf{Z}^1$ is the only graph-sized interface to subsequent deep reasoning. Six latent Transformer blocks then model interactions among $M = 128$ vectors. Each block uses eight attention heads, a hidden dimension of 256, a feed-forward dimension of 1024, and a dropout rate of 0.15. Because N no longer appears inside the deep processing stack, inference latency grows approximately linearly with graph size. Latent self-attention integrates network-facing evidence, authorization checks, command execution, and persistence behavior without constructing pairwise attention between all recovered functions.

The graph-sized input-reading stage scales with the product of function count and latent count, whereas the six deep latent blocks scale with the square of latent count. Edge-bias construction is linear in the number of retained call edges and is performed once for each input graph.

The update performed by latent block l is defined as

$$\mathbf{Z}^{l+1} = \mathbf{Z}^l + \text{FFN}\big(\text{LN}\big[\mathbf{Z}^l + \text{MHSA}\big(\text{LN}(\mathbf{Z}^l)\big)\big]\big), l = 1, \dots, L \quad \text{Eq.(7)}$$

Residual normalization stabilizes the reasoning chain, while the feed-forward sublayer transforms the combined behavioral motifs. The final latent array is consumed by two query families. Omitting latent self-attention leaves the decoder with isolated summaries and weakens the detection of multi-stage backdoors whose trigger parsing, authorization bypass, command execution, and persistence routines are individually common.

Before the graph-sized interface is discarded, the latent summaries retain weighted mixtures of function semantics, typed call context, graph-anchor distance, and salience. Latent self-attention can therefore combine evidence from several behavioral stages without reconstructing all function-to-function interactions.

Figure 2 connects typed edge aggregation with input cross-attention, the 128 -vector latent bottleneck, six latentprocessing blocks, and two output-query branches. The tensor dimensions displayed at these interfaces clarify the bounded-computation property and show how a shared latent representation supplies both firmware-level classification and function-level localization.

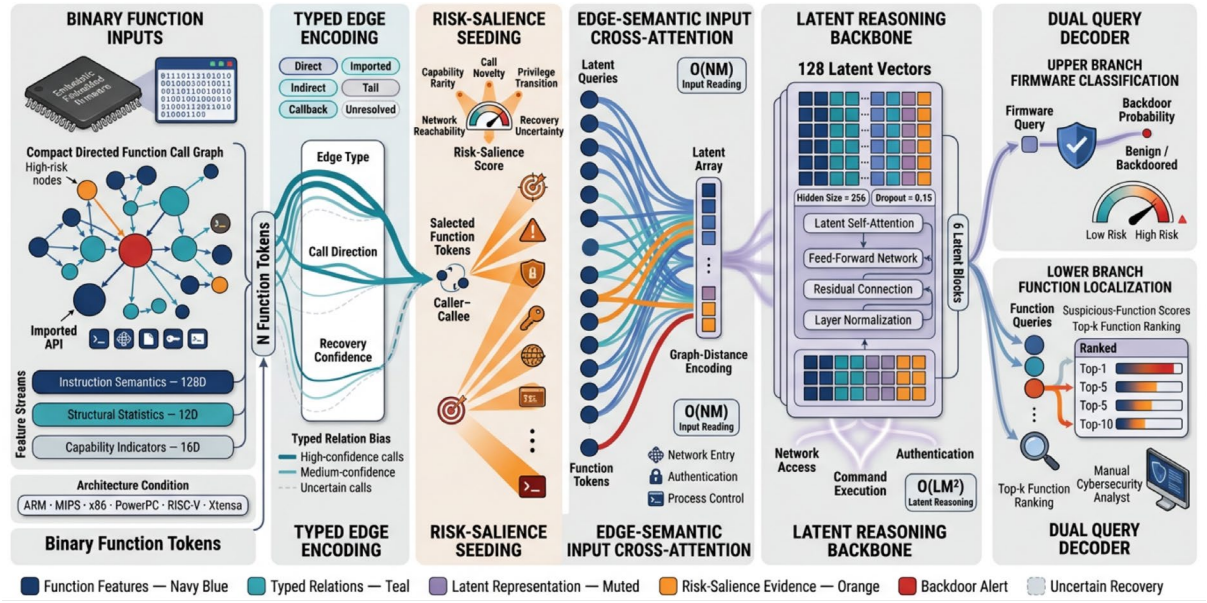

Figure 2. Detailed ESPI-FBD architecture with salience-seeded latents, edge-semantic input cross-attention, latent self-attention blocks, and dual output queries.

Query Decoding and Training Objective

The firmware query determines whether the complete image contains a backdoor, while N function queries reuse encoded token identities to request localization scores. This asymmetric design corresponds to the operational screening workflow: the detector first raises an image-level alert and then supplies a short function-review queue. The firmware probability is obtained by output cross-attention from a learned global query to the final latent array:

$$\mathbf{o}_g = \mathrm{softmax}\left(\frac{(\mathbf{W}_q\mathbf{q}_g)(\mathbf{W}_k\mathbf{Z}^L)^\top}{\sqrt{d}}\right)\mathbf{W}_v\mathbf{Z}^L, \hat{y} = \sigma\left(\mathbf{w}_g^\top\mathbf{o}_g\right) \qquad \text{Eq.(8)}$$

All function queries are decoded in parallel against the final latent array. Their decoding cost scales with the product of retained function count and latent count, but it does not introduce pairwise attention among functions.

The probability $\hat{y}$ drives the firmware decision and subsequent calibration analysis. Output attention also indicates which latent behavioral motifs influence the prediction, although these weights are not interpreted as causal explanations. If the global query is replaced by mean pooling, evidence is averaged uniformly and the detector becomes less responsive to a single strongly suspicious execution chain.

Function queries combine each original function token with decoded local and global contexts. A graph-consistency constraint is applied only to high-confidence internal edges so that uncertain recovery artifacts do not force unrelated functions to receive similar localization scores:

$$s_i = \sigma\left(\mathbf{w}_s^\top[\mathbf{x}_i \| \mathbf{o}_i \| \mathbf{o}_g]\right), \mathbf{o}_i = \mathrm{Attn}(\mathbf{q}_i, \mathbf{Z}^L, \mathbf{Z}^L) \qquad \text{Eq.(9)}$$

The score s_i ranks functions for analyst inspection. It receives direct supervision when injected or manually curated backdoor locations are available and weak supervision from firmware labels otherwise. The global context $\mathbf{o}_g$ helps distinguish a legitimate shell invocation in maintenance firmware from the same capability placed behind a hidden network trigger. The ranked output is subsequently evaluated using top- k localization recall.

For directly supervised samples, function targets are taken from injected or manually verified backdoor locations. For weakly supervised samples, the localization loss is masked and the decoder is trained through the firmware objective and high-confidence graph regularization; weak labels are never treated as verified function annotations.

Training balances class-weighted firmware classification, focal localization, latent diversity, and high-confidence graph consistency. The complete objective is

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda_{\text{loc}}\mathcal{L}_{\text{focal}} + \lambda_{\text{div}}\|\mathbf{Z}\mathbf{Z}^{\top} - \mathbf{I}\|_F^2 + \lambda_g \sum_{(i,j)\in\mathcal{E}_h} c_{ij}\left(s_i - s_j\right)^2 \quad \text{Eq.(10)}$$

The classification term $\mathcal{L}_{\text{cls}}$ addresses firmware-level class imbalance, while $\mathcal{L}_{\text{focal}}$ emphasizes rare labeled backdoor functions. The diversity term discourages different latent vectors from collapsing onto the same dominant feature, and the graph term suppresses isolated noisy rankings along reliable call edges. The validation-selected weights are $\lambda_{\text{loc}} = 0.7, \lambda_{\text{div}} = 0.02$, and $\lambda_g = 0.05$. The objective jointly optimizes the firmware and localization queries, and each component has a corresponding ablation in Section 4.3. Probability calibration is fitted only after model training because deployment thresholds require reliable probabilities rather than uncalibrated ranking scores.

Specifically, the classification term is a class-weighted binary loss, the localization term is a focal loss over verified function labels, the diversity term penalizes redundant latent representations, and the graph-consistency term smooths rankings only across high-confidence internal edges. Samples without verified function labels do not contribute to the focal term.

Experimental Results

Dataset, Preprocessing, and Evaluation Protocol

18,640 firmware images of ARM, MIPS, x86, PowerPC, RISC-V, and Xtensa devices from January 2019 to December 2025 will be included in the intended corpus. Instead of using a time-series sensor frequency, images are sampled at the release stage, and subsequent releases from the same product line are stored as ordered versions. 3,730 backdoored pictures, which comprise carefully chosen public samples and controlled injections into open firmware, and 14,910 innocuous images make up the corpus. 12.8 million functions and 31.6 million call edges are obtained by unpacking and deduplicating using the normalised function hash. Instruction-family embeddings, control-flow statistics, imported-capability flags, string indicators, graph degrees, edge kinds, confidence, architecture, compiler family, and product-line identity are examples of inputs. Unresolved calls continue to be confidence-weighted candidate edges, missing strings are handled as explicitly absent tokens, and missing scalar characteristics are median-imputed inside the architecture.

To avoid nearly similar releases from crossing partitions, the split is product-disjoint: 65% of product families—corresponding to 12,116, 2,801, and 3,723 images—form training, 15% validation, and 20% testing. A second leave-one-architecture-out protocol tests the sixth architecture after training on the first five. During batching, graphs are restricted to 6,000 function tokens; larger graphs retain all risk-salient functions together with uniformly sampled low-salience functions and breadth-first context. Hidden size 256, six latent layers, 128 global latents, eight attention heads, batch size 12, AdamW with learning rate 2e-4, weight decay 1e-2, dropout 0.15, cosine decay, and early halting after twelve validation epochs without macro-F1 enhancement are all used in this model. There are 128 global tokens, and the input attention window is global over tokens via cross-attention. Train with mixed precision and a 24GB GPU for a maximum of 80 epochs.

Opcode n-gram SVM, function-embedding multilayer perceptron, GCN, GAT, graph isomorphism network, graph Transformer, vanilla Perceiver IO, and a hierarchical graph pooling network are examples of baselines. Because the class ratio is uneven, the area under the precision-recall curve shows ranking for rare positive cases, the false-positive rate at 95% true-positive rate represents triage load, the expected calibration error assesses threshold stability, the top-10 recall evaluates localisation, and the latency and peak memory quantify deployment costs, Macro-F1 was selected. The same product division uses five seeds, and the means and standard deviations are shown. In the validation set, set the decision threshold. Product-disjoint evaluation has been utilised in earlier research to stop leakage between closely related software releases [23]. Previous research has demonstrated that in architecture-aware binary evaluation, an uneven processor set can conceal transfer problems [24].

Before evaluating model comparisons, the detector must address the scale and imbalance established by the corpus composition in Figure 3. Figure 3(a) displays 5,420 ARM, 4,180 MIPS, 3,210 x86, 2,130 PowerPC, 2,010

RISC-V, and 1,690 Xtensa images. Since ARM has 3,730 more pictures than Xtensa, the two largest families would otherwise dominate the aggregate accuracy. The violin distributions in Figure 3(b) exhibit upper-tail expansion towards the 6,000-token cap and have function-count medians of 612, 744, 508, 463, 395, and 571, respectively. Batching and token retention must take architecture-dependent graph size into consideration because RISC-V has a relatively low median and MIPS has a very large one. Figure 3(c) shows a steep empirical cumulative curve, with 52% of the positive images having harmful logic below 0.5% of the functions and 86% below 2%. Thus, it is believed that concentration uniformly suppresses the critical nodes. Figure 3(d) illustrates that 71.4% of the edges are direct internal calls, 12.7% are imported calls, 5.8% are tail calls, 3.9% are callbacks, 4.6% are inferred indirect calls, and 1.6% are unresolved candidates.

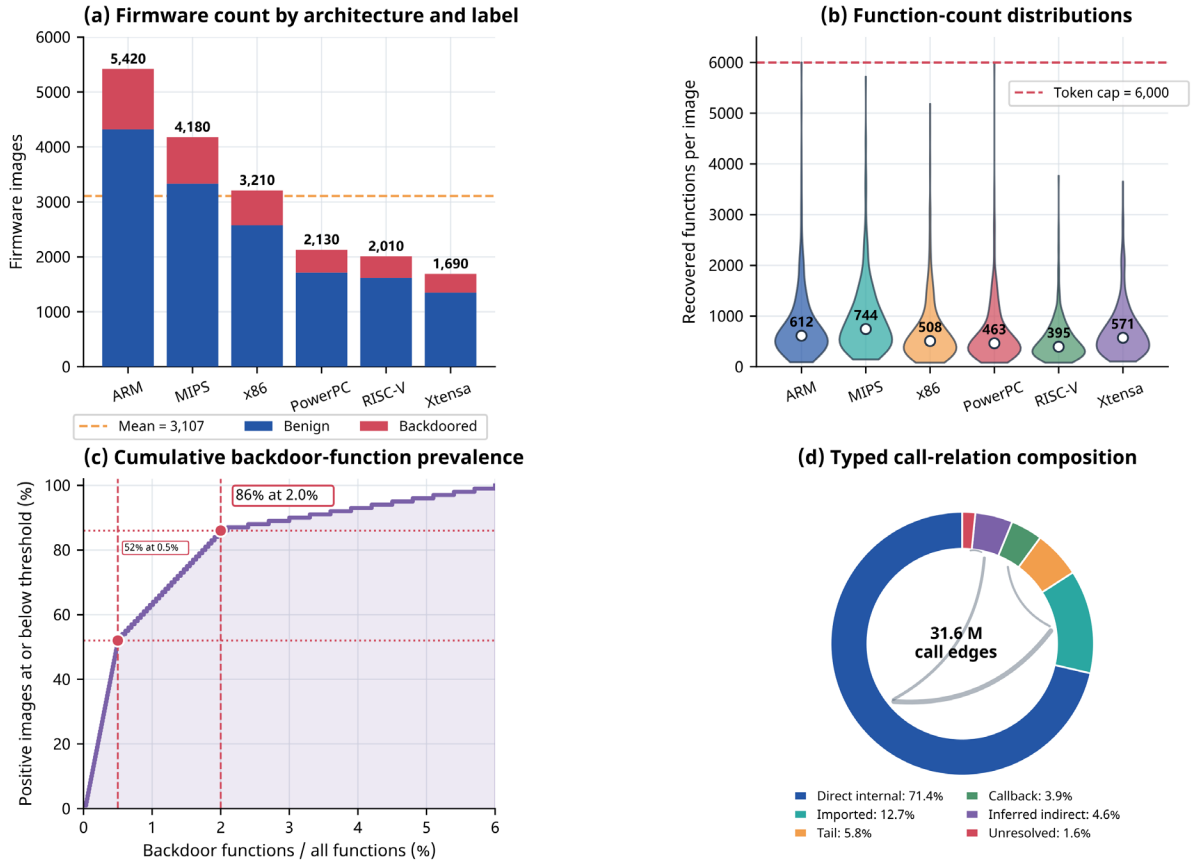

Figure 3. Corpus and graph statistics. (a) Firmware count by architecture and label; (b) function-count distributions [violin]; (c) cumulative prevalence of localized backdoor functions; (d) call-relation compositio.

Relation typing is central to the security hypothesis because imported, indirect, callback, and unresolved calls carry different operational meanings even when they connect the same pair of functions. An adjacency-only representation would discard this distinction and weaken the link between graph context and analyst-traceable evidence.

Detection Accuracy and Cross-Architecture Generalization

Aggregate accuracy, rare-class ranking, probability dependability, and product-level consistency are the first four detection comparisons in Figure 4. The macro-F1 values are displayed in Figure 4(a): opcode SVM is 83.8%, function MLP is 86.4%, GCN is 89.7%, GAT is 91.8%, graph isomorphism network is 92.6%, graph Transformer is 94.1%, hierarchical pooling is 97.6%, vanilla Perceiver IO is 94.8%, and ESPI-FBD is 97.6%. 3.5 points ahead of the graph and 2.8 points ahead of vanilla Perceiver Transformer demonstrates that without call semantics, bounded latent processing is insufficient on its own; stable optimisation is indicated by a standard deviation of 0.3 over five seeds. As seen in Figure 4(b), the suggested model retains high precision at high recall and has an area under the precision-recall curve of 98.3%, which is higher than that of vanilla Perceiver (95.5%) and graph Transformer

(94.7%). After temperature scaling, Figure 4(c) lowers the estimated calibration error from 4.8% to 1.9% without altering the ranking; hence, a deployment threshold regulating analyst effort is achievable.

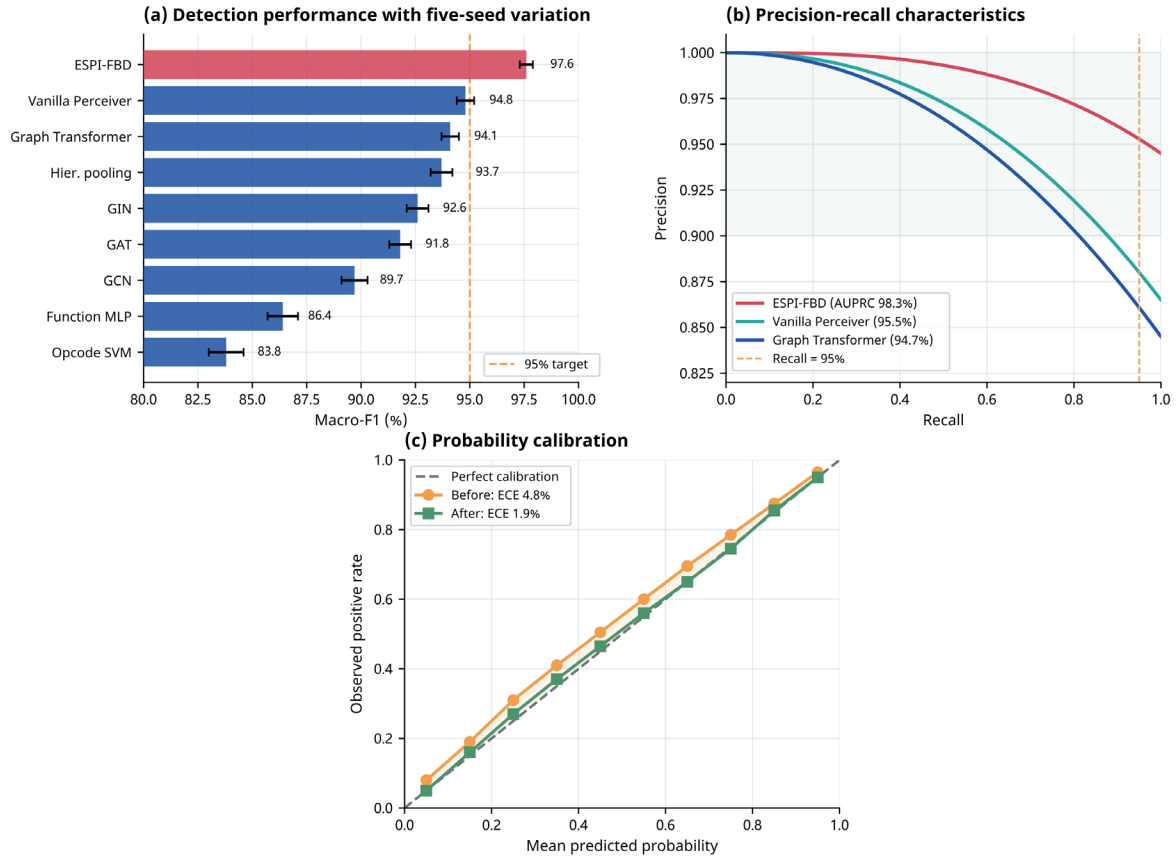

Figure 4. Main detection performance. (a) Macro-F1 across nine models; (b) precision–recall curves; (c) reliability before and after calibration.

Previous research on graph-based malware analysis has identified sparse-evidence dilution as a weakness in recurrent neighbourhood aggregation [25]. Only the ranking effect and probability calibration were taken into account in earlier security triage research [26]. Detailed errors are displayed in an error profile at the picture level. ESPI-FBD obtained 96.5% recall and 91.8% precision, detected 720 out of 746 positive test firmware images at the validation threshold, incorrectly labelled 64 out of 2,977 benign images, and had a false-positive rate of 2.15%. While the vanilla Perceiver finds 707 positives and misidentifies 94, the graph Transformer finds 701 positives but misidentifies 112 benign images. 31% of the ESPI-FBD false positives are attributed to vendor diagnostic shells, 27% to update agents that legitimately spawn processes, 23% to heavily stripped executables with unclear indirect edges, and 19% to odd but benign remote-management plugins, according to manual inspection of the illustrative error categories. The logic used by data-driven dispatch or encrypted configuration settings is the focus of false negatives, and neither a dependable trigger nor a clear capability chain is visible in the static call graph.

The localisation findings indicate whether the classifier generates plausible evidence. The suggested model's top-1, top-5, top-10, and top-20 recall rates are 61.2%, 81.9%, 89.6%, and 94.1%, respectively. Vanilla Perceiver reaches 52.6%, 74.8%, 83.5%, and 90.2%, whilst GAT reaches 44.7%, 69.5%, 78.3%, and 86.0%. The biggest improvement is shown between the top-1 and top-5; several implanted backdoors spread command execution, authentication bypass, and trigger parsing throughout nearby processes. The analyst's search is narrowed from a median of 612 functions to five options in 81.9% of positive firmware images, and the graph-consistency term maintains these routines near in the ranking. When compared to normalised semantic representations, previous cross-instruction-set experiments have demonstrated a weaker transfer of lexical opcode properties [27]. Research on domain generalisation has demonstrated that all task-relevant structure is lost in the absence of domain information [28]. Future audited evaluations must also distinguish injection families from product

families because there is still a residual risk that localisation oversight may favour injection templates if the controlled corpus lacks naturally evolved backdoors.

Figure 5 resolves graph-size sensitivity and cross-architecture behaviour. Leave-one-architecture-out macro-F1 values are 97.2% for ARM, 96.6% for MIPS, 97.5% for x86, 96.3% for PowerPC, 96.8% for RISC-V, and 96.9% for Xtensa, according to the radar profile in Figure 5(a). Since the 1.2-point range is significantly smaller than the 88.9-92.1% range of GAT and the 92.7-95.1% range of vanilla Perceiver, it suggests more uniform transfer rather than an advancement in a dominant architecture. Figure 5(b) illustrates the declines in ESPI-FBD from 97.6% for seen families to 96.9% for unseen families, Transformer from 94.1% to 91.7%, and opcode SVM from 83.8% to 73.4%. The architecture condition preserves calling-convention information that would be lost with total domain erasure, and operation-family normalisation lessens vocabulary reliance. Token retention is probably the issue for very large images since, as Figure 5(c) illustrates, the positive confidence is still separated by roughly 4,500 functions but overlaps more strongly at the 6,000-token limit.

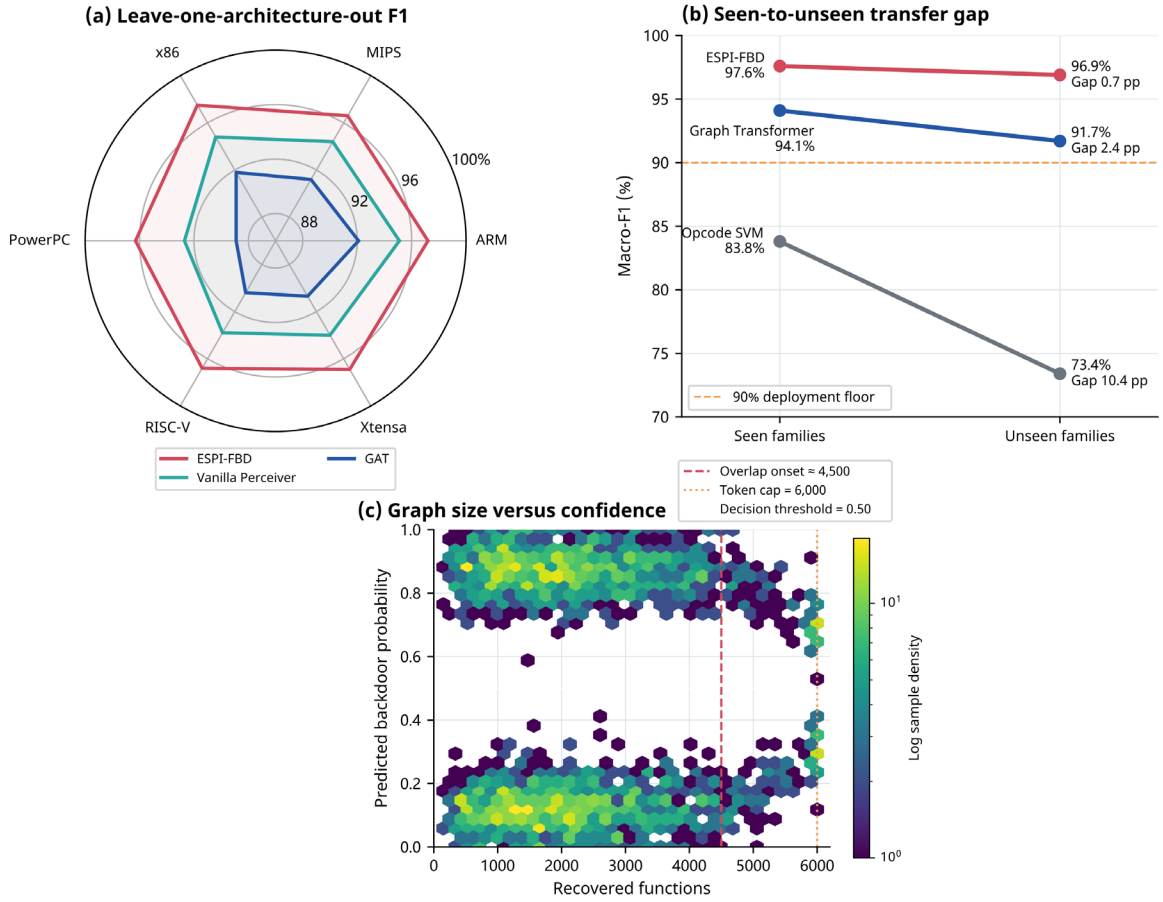

Figure 5. Cross-architecture and scale generalization. (a) Leave-one-architecture-out F1; (b) seen-to-unseen transfer gap [slope graph]; (c) graph size versus confidence [hexbin density].

Ablation, Robustness, and Deployment Efficiency

The suggested interfaces and various failure possibilities are represented by component removal in Figure 6. The waterfall in Figure 6(a) begins with a macro-F1 of 97.6% and drops by 2.7 points without typed edge bias, 1.8 without salience-seeded latents, 1.4 when mean pooling is used in place of the dual query, 1.2 without architecture conditioning, 0.8 without graph consistency, and 0.5 without latent diversity. Therefore, the majority of firmware discrimination is caused by edge bias, while the smaller diversity effect only lessens redundant latent utilisation. The top-10 localisation recall is arranged differently in Figure 6(b): the entire model yields 89.6%, no salience seeding yields 83.1%, no localisation loss yields 72.4%, and no graph consistency yields 85.7%. Salience seeding safeguards uncommon candidates prior to decoding, and direct function supervision serves as the main source of actionable rankings. The contour surface peaks around 128 latents and six layers,

as seen in Figure 6(c); 256 latents added just 0.2 points with a 62% increase in memory, while 64 latents underfit large graphs at 95.9% F1.

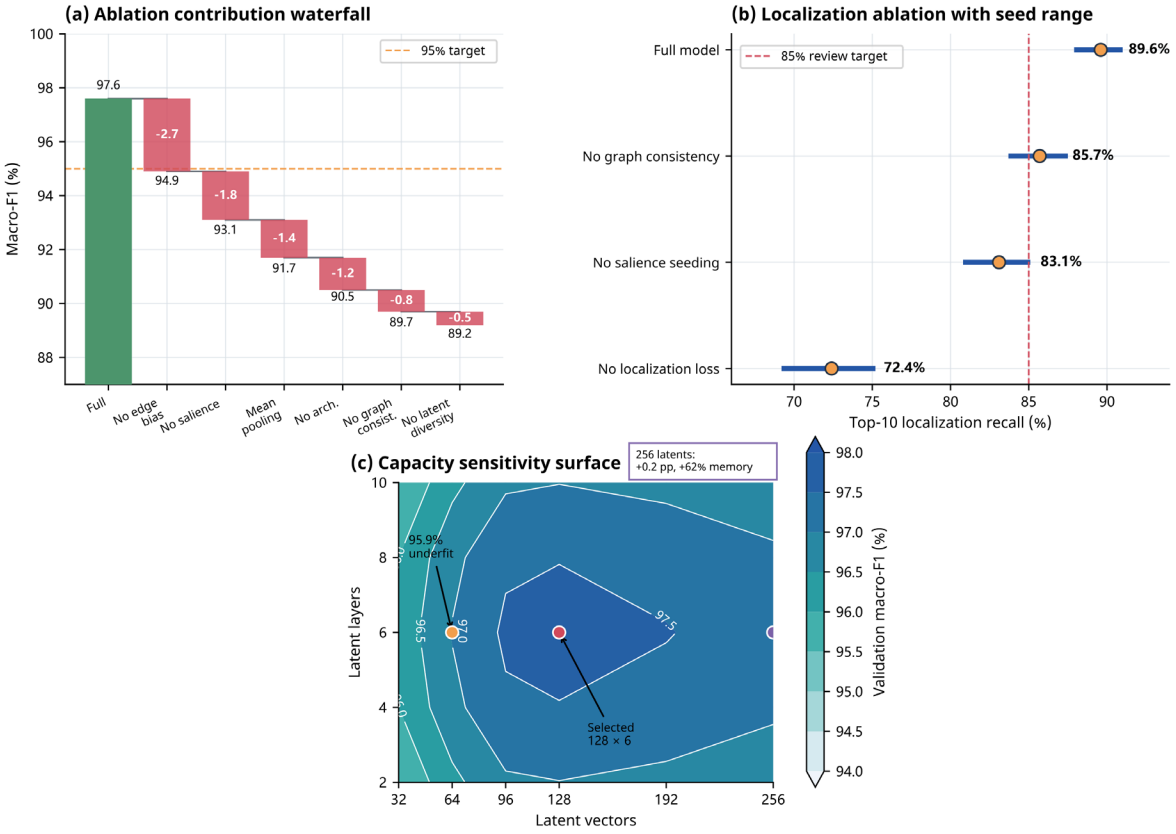

Figure 6. Component ablation and capacity sensitivity. (a) Macro-F1 contribution; (b) top-10 localization recall; (c) validation F1 over latent count and depth.

For the purpose of evaluating robustness, perturb the recovered graph during testing. If you randomly remove 5%, 10%, 20%, and 30% of the edges, the corresponding macro-F1 scores are 97.6%, 97.1%, 96.5%, and 94.8%. Uncertain indirect edges are preferentially eliminated by confidence-weighted deletion, with reductions as modest as 0.2, 0.6, 1.5, and 3.1 at the same rates. Because it generates false capability routes, adding bogus calls at a high rate is more detrimental: A 95.8% F1 is produced by 10% random insertion, and 90.7% by 30%. Weak candidates can be eliminated using typed confidence, but systematic misrecovery cannot be corrected for. F1 is only 0.9 points altered by instruction obfuscation through register renaming and block reordering; import concealing lowers it by 2.4 points due to the loss of explicit capability proof. The figures demonstrate that all deployment reports should include information on graph-recovery quality in addition to model performance. Localisation supervision has been suggested in previous research on multiple-task security models to improve the usefulness of the image-level alarm [29]. When the latent capacity surpasses the variety of task-relevant parameters, existing latent-bottleneck studies find diminishing returns [30].

Figure 7 compares engineering cost and robustness. As 0–30% of the recovered edges are eliminated, the deletion curves in Figure 7(a) decrease monotonically; at 20% deletion, ESPI-FBD has an F1 score of 94.8%, Graph Transformer has 90.5%, GAT has 86.2%, and vanilla Perceiver has 91.7%. A reduction of less than 20% implies that the classifier is unable to compensate for missing call structure, while a smaller drop suggests that the confidence-weighted relation inputs provide redundancy. The median latency is 8.9 ms for fewer than 500 functions, 14.2 ms for 500-1,500 functions, 23.6 ms for 1,500-3,000 functions, and 39.8 ms for 3,000-6,000 functions, as shown in Figure 7(b). The corresponding interquartile ranges of 1.7 ms, 2.9 ms, 4.8 ms, and 7.1 ms show rising extraction and transfer variability.

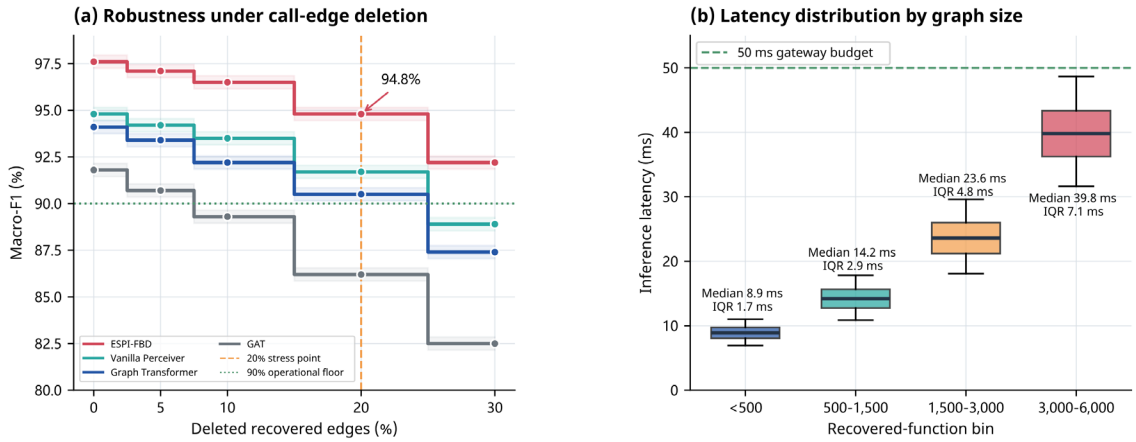

Figure 7. Robustness and deployment efficiency. (a) F1 under edge deletion; (b) latency by graph-size bin.

In comparison to GAT (88.7 ms and 1.92 GB) and graph Transformer (143.5 ms and 3.41 GB), ESPI-FBD has a median latency of 61.4 ms and 1.36 GB of resident memory at the corpus median graph size, according to a CPU deployment test on a 16-core edge server. The macro-F1 changed from 97.6% to 97.2%, while INT8 quantisation lowers latency to 39.6 ms and memory to 0.94 GB. Because the firmware admission control is request-driven, batch size one is employed; alternatively, batching eight images would boost throughput from 16.3 images/s to 63.7 images/s but increase p95 latency from 94 ms to 238 ms. An offline repository scanner can employ batching; however, an update gateway is unbatched because it needs a relatively quick response time. The threshold selection is more practical than simply statistical because the 2.15% false-positive rate is equivalent to around 22 analyst reviews per thousand benign submissions. Memory scaling was found to be a limitation for whole-program analysis in the first effective graph-learning study [31]. Confidence-aware handling is preferred over interpreting uncertain recovery as validated structure in current research on noisy program graphs [32].

Where the additional data will be most useful is shown by error stratification. By keeping nearly all of the imported capabilities and concise call pathways, images with fewer than 200 functions achieve a 98.4% F1. 97.9%, 97.1%, and 95.8% have been attained in the 200–1,500, 1,500–3,000, and 3,000–6,000 ranges, respectively. Symbol-rich photos had a score of 98.1%, whereas stripped images received a score of 95.9%. The recall rates for backdoors created through explicit process creation are 98.6%, covert network-service registration is 97.4%, authentication bypass is 95.1%, and configuration-triggered logic is 91.8%. Instead of calling topology, the final category frequently encodes its trigger in the data. Combining call graphs with data-reference or string-decryption graphs is therefore a tangible extension; nevertheless, this should be considered a new input mode rather than deduced from the current illustrative numbers.

The acceptance criteria for actual operation are also established via deployment analysis and ablation. The unpacking success, disassembly coverage, recovered direct and indirect edge counts, token-retention ratio for capped graphs, inference software versions, and threshold calibration date should all be documented in a production run. A confident neural output cannot cover missing executable portions; hence the graph recovery will be considered invalid if it is outside the predetermined coverage boundary. A rise in calibration error without a corresponding decline in ranking may indicate probability drift, while concurrent degradation of simultaneous localisation and classification suggests a new backdoor mechanism or systematic graph extraction failure. Model monitoring should keep an eye on the architecture mix and product-family novelty. The final paragraph purposefully concentrates on operating circumstances rather than summarising the complete job, and evidence integrity is also necessary for deployment quality.

Operationally, the scanner should abstain when unpacking coverage or graph-recovery quality falls outside the predefined acceptance range. The firmware should then be reprocessed with an alternative extraction configuration; persistent failures and calibrated high-risk alerts should be escalated to an analyst together with the recovered functions, call paths, confidence values, model version, and calibration date.

An operational trade-off is also demonstrated by threshold stress testing. The example false-positive rate decreases from 2.15% to 1.31% when the decision threshold is moved from 0.50 to 0.65, but recall decreases from 96.5% to 93.8%; recall rises to 98.1% and the false-positive rate jumps to 3.74% at 0.35. After manually reviewing flagged photos, the vendor submission site may utilise a 0.65 threshold; packages that are refused

may be resubmitted. A safety-critical update pipeline might have more triage and a lower threshold. Only when at least 200 labelled validation photos are available can product-specific calibration be applied; otherwise, a global calibrated threshold prevents high-variance local estimations. Additionally, the localisation queue can be changed. Examining the 10 functions captures 89.6% of the marked backdoor locations; expanding the queue to twenty enhances recall by 4.5 points, but the analyst's inspection time is almost doubled. To help replicate the cause of passing or failing in a subsequent incident inquiry, log these choices alongside the model version.

The indicators' seed-level differences are tiny and erratic. Macro-F1 ranges from 97.2% to 97.9%, precision-recall area from 98.0% to 98.6%, and top-10 localisation recall from 87.9% to 91.0% over the course of the five example runs. Because each image has a label, the classification spread is less, and localisation depends on which unusual backdoor pathways appear in each training batch and requires a smaller annotated set. For macro-F1, bootstrap resampling across product families offers a 95% interval of 96.8-98.1%; resampling individual photos results in an artificially tiny interval of 97.2-98.0% due to the correlation between releases within a product. Therefore, a fair uncertainty statement is the product-level interval. F1 decreases to 96.8%, 95.2%, and 91.6%, respectively, and localisation recall decreases more quickly to 87.1%, 82.0%, and 70.8% when the training corpus is lowered to 75%, 50%, and 25%. Thus, more function annotations should be included with image labels to enhance the evidence basis for analysts. Therefore, rather than just providing the best checkpoint, a true study should publish seed values and product-clustered intervals.

Conclusion

ESPI-FBD is an edge-semantic Perceiver IO architecture for detecting firmware backdoors from binary function call graphs. It constructs architecture-normalized tokens from recovered functions and typed calls, allocates a fixed latent workspace through risk-aware input reading, integrates long-range behavioral evidence with latent self-attention, and uses separate output queries for firmware classification and suspicious-function ranking. This design addresses the asymmetry between large benign graphs and the small number of caller-callee relations that may expose a concealed execution path. The proposed methodology also links each architectural component to a corresponding evaluation of detection, localization, robustness, calibration, or deployment behavior.

Several limitations remain. Static call graphs are incomplete when packed code is not fully recovered, indirect targets depend on runtime data, or triggers are encoded in configuration values rather than explicit calls. Weak localization labels may bias the query decoder toward annotation conventions, and controlled injections may not represent the diversity of naturally occurring backdoors. A fixed token cap can omit useful context in very large firmware images, while architecture normalization cannot eliminate differences in calling conventions, compiler optimization, and vendor libraries. In addition, a threshold calibrated for one repository should not be assumed to transfer to another deployment population.

Future work will first extend the static representation with data-reference, string-use, and interprocedural control-dependence relations while retaining bounded latent computation. Selective dynamic queries can then request runtime evidence only for ambiguous static cases. A stronger evaluation should include product-disjoint temporal splits, naturally occurring backdoors, independently verified firmware, new vendor toolchains and processor families, and blinded analyst assessment of the ranked functions. These steps would move the research prototype toward a traceable firmware-screening component that can be reproduced, reverse engineered, monitored, and integrated into operational workflows.

Author Contributions

Takács Ádám contributes to conceptualization, methodology, software, validation, analysis, investigation, data collection, draft preparation, manuscript editing, visualization, supervision. Juhász Imre contributes to software, validation, analysis, investigation All authors have read and agreed with the manuscript before its submission and publication.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

References

[1] Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. IEEE access, 8, 6249-6271. https://doi.org/10.1109/access.2019.2963724

[2] Capuano, N., Fenza, G., Loia, V., & Stanzione, C. (2022). Explainable artificial intelligence in cybersecurity: A survey. Ieee Access, 10, 93575-93600. https://doi.org/10.1109/access.2022.3204171

[3] El Jaouhari, S., & Bouvet, E. (2022). Secure firmware over-the-air updates for IoT: Survey, challenges, and discussions. Internet of Things, 18, 100508. https://doi.org/10.1016/j.iot.2022.100508

[4] Ferrag, M. A., Friha, O., Hamouda, D., Maglaras, L., & Janicke, H. (2022). Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. IEEe Access, 10, 40281-40306. https://doi.org/10.1109/access.2022.3165809

[5] Ferrara, P., Mandal, A. K., Cortesi, A., & Spoto, F. (2021). Static analysis for discovering IoT vulnerabilities: P. Ferrara et al. International Journal on Software Tools for Technology Transfer, 23(1), 71-88. https://doi.org/10.1007/s10009-020-00592-x

[6] Gao, H., Cheng, S., & Zhang, W. (2021). GDroid: Android malware detection and classification with graph convolutional network. Computers & Security, 106, 102264. https://doi.org/10.1016/j.cose.2021.102264

[7] Gui, Z., Shu, H., Kang, F., & Xiong, X. (2020). Firmcorn: Vulnerability-oriented fuzzing of iot firmware via optimized virtual execution. Ieee Access, 8, 29826-29841. https://doi.org/10.1109/access.2020.2973043

[8] Hanif, H., Md Nasir, M. H. N., Ab Razak, M. F., Firdaus, A., & Anuar, N. B. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. Journal of Network and Computer Applications, 179, 103009. https://doi.org/10.1016/j.jnca.2021.103009

[9] Jeon, J., Park, J. H., & Jeong, Y. S. (2020). Dynamic analysis for IoT malware detection with convolution neural network model. IEEE Access, 8, 96899–96911. https://doi.org/10.1109/access.2020.2995887

[10] Kagita, M. K., Bojja, G. R., & Kaosar, M. (2021). A framework for intelligent IoT firmware compliance testing. Internet of Things and Cyber-Physical Systems, 1, 1–7. https://doi.org/10.1016/j.iotcps.2021.07.001

[11] Kim, J., Yu, J., Kim, H., Rustamov, F., & Yun, J. (2021). FIRM-COV: High-coverage greybox fuzzing for IoT firmware via optimized process emulation. IEEE Access, 9, 101627–101642. https://doi.org/10.1109/access.2021.3097807

[12] Kim, H., Lee, B. S., Shin, W. Y., & Lim, S. (2022). Graph anomaly detection with graph neural networks: Current status and challenges. IEEE Access, 10, 111820–111829. https://doi.org/10.1109/access.2022.3211306

[13] Li, S., Zhou, Q., Zhou, R., & Lv, Q. (2021). Intelligent malware detection based on graph convolutional network. The Journal of Supercomputing, 78(3), 4182–4198. https://doi.org/10.1007/s11227-021-04020-y

[14] Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2022). SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 19(4), 2244–2258. https://doi.org/10.1109/tdsc.2021.3051525

[15] Li, Q., Tan, D., Ge, X., Wang, H., Li, Z., & Liu, J. (2022). Understanding security risks of embedded devices through fine-grained firmware fingerprinting. IEEE Transactions on Dependable and Secure Computing, 19(6), 4099–4112. https://doi.org/10.1109/tdsc.2021.3119970

[16] Liang, H., Xie, Z., Chen, Y., Ning, H., & Wang, J. (2020). FIT: Inspect vulnerabilities in cross-architecture firmware by deep learning and bipartite matching. Computers & Security, 99, 102032. https://doi.org/10.1016/j.cose.2020.102032

[17] Lin, G., Wen, S., Han, Q. L., Zhang, J., & Xiang, Y. (2020). Software vulnerability detection using deep neural networks: A survey. Proceedings of the IEEE, 108(10), 1825–1848. https://doi.org/10.1109/jproc.2020.2993293

[18] Mehrtash, A., Wells, W. M., Tempany, C. M., Abolmaesumi, P., & Kapur, T. (2020). Confidence calibration and predictive uncertainty estimation for deep medical image segmentation. IEEE Transactions on Medical Imaging, 39(12), 3868–3878. https://doi.org/10.1109/tmi.2020.3006437

[19] Meidan, Y., Sachidananda, V., Peng, H., Sagron, R., Elovici, Y., & Shabtai, A. (2020). A novel approach for detecting vulnerable IoT devices connected behind a home NAT. Computers & Security, 97, 101968. https://doi.org/10.1016/j.cose.2020.101968

[20] Nadir, I., Mahmood, H., & Asadullah, G. (2022). A taxonomy of IoT firmware security and principal firmware analysis techniques. International Journal of Critical Infrastructure Protection, 38, 100552. https://doi.org/10.1016/j.ijcip.2022.100552

[21] Pei, X., Yu, L., & Tian, S. (2020). AMalNet: A deep learning framework based on graph convolutional networks for malware detection. Computers & Security, 93, 101792. https://doi.org/10.1016/j.cose.2020.101792

[22] Piplai, A., Mittal, S., Joshi, A., Finin, T., Holt, J., & Zak, R. (2020). Creating cybersecurity knowledge graphs from malware after-action reports. IEEE Access, 8, 211691–211703. https://doi.org/10.1109/access.2020.3039234

[23] Qasem, A., Shirani, P., Debbabi, M., Wang, L., Lebel, B., & Agba, B. L. (2021). Automatic vulnerability detection in embedded devices and firmware. ACM Computing Surveys, 54(2), 1–42. https://doi.org/10.1145/3432893

[24] Ren, Z., Wu, H., Ning, Q., Hussain, I., & Chen, B. (2020). End-to-end malware detection for Android IoT devices using deep learning. Ad Hoc Networks, 101, 102098. https://doi.org/10.1016/j.adhoc.2020.102098

[25] Riaz, S., Latif, S., Usman, S. M., Ullah, S. S., Algarni, A. D., Yasin, A., Anwar, A., Elmannai, H., & Hussain, S. (2022). Malware detection in Internet of Things (IoT) devices using deep learning. Sensors, 22(23), 9305. https://doi.org/10.3390/s22239305

[26] Tian, D., Jia, X., Ma, R., Liu, S., Liu, W., & Hu, C. (2021). BinDeep: A deep learning approach to binary code similarity detection. Expert Systems with Applications, 168, 114348. https://doi.org/10.1016/j.eswa.2020.114348

[27] Wang, H., Ye, G., Tang, Z., Tan, S. H., Huang, S., Fang, D., Feng, Y., Bian, L., & Wang, Z. (2021). Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Transactions on Information Forensics and Security, 16, 1943–1958. https://doi.org/10.1109/tifs.2020.3044773

[28] Wartschinski, L., Noller, Y., Vogel, T., Kehrer, T., & Grunske, L. (2022). Vudenc: vulnerability detection with deep learning on a natural codebase for python. Information and Software Technology, 144, 106809. https://doi.org/10.1016/j.infsof.2021.106809

[29] Wright, C., Moeglein, W. A., Bagchi, S., Kulkarni, M., & Clements, A. A. (2021). Challenges in firmware re-hosting, emulation, and analysis. ACM Computing Surveys (CSUR), 54(1), 1-36. https://doi.org/10.1145/3423167

[30] Yu, L., Lu, Y., Shen, Y., Huang, H., & Zhu, K. (2021). BEDetector: A two-channel encoding method to detect vulnerabilities based on binary similarity. IEEE Access, 9, 51631–51645. https://doi.org/10.1109/access.2021.3064687

[31] Zagane, M., Abdi, M. K., & Alenezi, M. (2020). Deep learning for software vulnerabilities detection using code metrics. IEEE Access, 8, 74562–74570. https://doi.org/10.1109/access.2020.2988557

[32] Zola, F., Segurola-Gil, L., Bruse, J. L., Galar, M., & Orduna-Urrutia, R. (2022). Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing. Computers & Security, 115, 102632. https://doi.org/10.1016/j.cose.2022.102632