

EvoPatch-IoT: Evolution-Aware Cross-Architecture Vulnerability Retrieval and Patch-State Profiling for BusyBox-Based IoT Firmware

Yinhao Xiao, Huixi Li^{*} and Yongluo Shen^{*}

School of Big Data and Artificial Intelligence, Guangdong University of Finance and Economics, Guangzhou 510320, China

^{*} Correspondence: lihuixi@gdufe.edu.cn (H.L.); sylkyo@gdufe.edu.cn (Y.S.)

How To Cite: Xiao, Y.; Li, H.; Shen, Y. EvoPatch-IoT: Evolution-Aware Cross-Architecture Vulnerability Retrieval and Patch-State Profiling for BusyBox-Based IoT Firmware. *Pragmatic Cybersecurity* 2026, 1(3), 19. <https://doi.org/10.53941/pc.2026.100019>

Received: 12 June 2026

Revised: 17 August 2026

Accepted: 10 September 2026

Published: 1 October 2026

Abstract: BusyBox is widely reused in Linux-based Internet-of-Things (IoT) firmware, but stripped symbols, heterogeneous instruction sets, uncertain component versions, and selective patch backports make N-day vulnerability localization difficult. We propose EvoPatch-IoT, an evolution-aware retrieval framework whose novelty lies not in another standalone encoder, but in coupling target-local geometric localization, architecture-normalized anonymous multi-view evidence, and a version-causal historical prototype memory for reference-guided retrieval. We also release an original benchmark constructed by collecting official BusyBox releases and compiling each selected version separately for AArch64, ARM, MIPS, MIPSEL, and x86_64 in both stripped and unstripped forms. It contains 57 compiled versions and 555 binaries, together with 155,845 high-confidence stripped-to-unstripped anchors; the dataset is available in the public record. On the original symmetric 57-version benchmark (1020 directed architecture pairs), EvoPatch-IoT obtains 34.56% Hit@1 and 56.24% Hit@10. Relative to the actual strongest baseline, the in-house geometry-only ShapeStat control, these are gains of 8.00% and 5.50%, rather than gains over a weaker literature-inspired control. A stricter historical-only evaluation on 15 chronologically held-out recent versions gives 28.76% Hit@1 and 49.93% Hit@10 versus 26.55% and 47.83% for geometry alone; paired tests over 300 architecture pairs yield $p < 1.6 \times 10^{-16}$, with version-block 95% confidence intervals excluding zero. Direct ablations show that geometry and historical prototypes provide most of the gain, while fusion mainly improves first-rank precision and MRR. A single-CVE patch-state study is retained as supporting, not general, evidence.

Keywords: IoT firmware security; BusyBox; stripped binary analysis; cross-architecture retrieval; binary code similarity; patch-state profiling

1. Introduction

Internet-of-Things (IoT) routers, cameras, gateways, and industrial edge nodes commonly retain third-party firmware components long after their upstream release. Large-scale firmware studies and emulation systems show both the prevalence of reused components and the difficulty of analyzing heterogeneous images at scale [1–3]. BusyBox is security-critical in this ecosystem because one compact executable exposes many utilities and parsers; a vulnerable routine can consequently recur across vendors, product generations, and architectures. Recent recurring-vulnerability and data-flow systems confirm that N-day defects remain an operational firmware risk rather than a purely historical concern [4–6].

The analyst rarely receives a clean source build. Symbols and DWARF metadata are often removed, version strings can be stale, and a vendor may backport only part of an upstream fix. The security problem addressed here is therefore reference-guided N-day localization: given a trusted upstream function or patch lineage, identify the

corresponding function in a stripped target component, prioritize the most plausible candidates, and expose evidence for manual patch verification. This problem is narrower than exploit generation but stronger than generic pairwise similarity because it must respect release chronology and must not infer “safe” merely from a high similarity score. Patch-presence research likewise shows that vulnerability verification requires evidence tied to changed program regions, not only whole-binary resemblance [7–9].

Binary representation learning has advanced through instruction language models, jump-aware transformers, graph encoders, contrastive objectives, intermediate representations, and natural-language supervision [10–17]. These methods mainly learn whether two functions are similar. EvoPatch-IoT addresses a different seam: it first localizes a trustworthy anonymous neighborhood inside the target binary, then uses multi-view evidence and a prototype formed only from releases older than the target to rerank that neighborhood. The prototype is keyed by the identity of an analyst-supplied upstream reference; it is unavailable, and is explicitly disabled, for a genuinely unidentified stripped query.

Our novelty is consequently the integration and deployment protocol rather than a claim that geometric distance, cosine fusion, or prototype averaging is individually new. Three elements are coupled: (i) bidirectional stripped/unstripped anchoring that turns weak target-local geometry into auditable supervision; (ii) architecture-normalized token, graph, context, and shape views that resolve ties inside the geometric neighborhood; and (iii) version-causal prototype memory that represents how a known function identity evolved before the target release. The revision evaluates these elements directly and quantifies where the incremental gain over geometry is small, significant, or absent.

We created the supporting dataset ourselves. Official BusyBox source releases were collected from the project archive [18], and 57 selected versions were built one by one with architecture-specific GNU cross-toolchains for AArch64, ARM, MIPS, MIPSEL, and x86_64. Each successful build was retained before and after symbol stripping. The resulting corpus contains 270 unstripped binaries, 285 stripped binaries, and 155,845 high-confidence anonymous anchors. It is an intentionally controlled component benchmark, not a collection extracted from physical IoT products; this design provides exact version/architecture ground truth but does not by itself establish end-to-end firmware extraction performance.

The main contributions are as follows.

- We release an original cross-version BusyBox binary dataset with 57 separately compiled releases, five ISA targets, both stripped and unstripped views, build logs, and function-level correspondence. The public record is available through Science Data Bank [19].
- We formulate reference-guided N-day retrieval and design EvoPatch-IoT as an alignment-aware, evolution-conditioned ranking layer. Unlike generic binary similarity, the deployed prototype bank is strictly historical and its query identity comes from the trusted upstream reference supplied by the analyst.
- We provide executable algorithms and complete score definitions for anonymous alignment, multi-view representation, historical prototype construction, and ranking, together with clearly named in-house component controls and literature-inspired feature-family controls.
- We evaluate 57 versions and 1020 directed architecture pairs, then add a chronological development interval of 12 versions and a test interval of 15 versions, seven direct component ablations, a 66-point coefficient sweep, paired Wilcoxon tests, version-block bootstrap intervals, and measured runtime/memory profiles.
- We delimit the security claim: EvoPatch-IoT prioritizes candidate functions for N-day audit, but does not prove patch presence, discover zero-days, defeat packing or adversarial obfuscation, or replace manual semantic confirmation. The single-CVE study is reported only as an illustrative patch-state probe.

2. Related Work

2.1. Binary Function Representation and Similarity

Early neural binary similarity work established instruction and function embeddings through self-attention and assembly-language pretraining [10,20]; jTrans subsequently modeled jump relations explicitly [11]. Work published mainly in 2024–2025 broadens this space. CLAP transfers natural-language supervision to binary representations [12]; GTrans and RCFG2Vec strengthen graph reasoning under obfuscation and long-range control-flow dependencies [13,14]; StrTune and BinCola use slicing or diversity-sensitive contrastive learning [15,16]; and CRABS-former targets cross-architecture transformer matching [21]. More recent systems explore architecture-neutral IR, global-attention graphs, execution-aware semantics, fine-tuning-free inference, and LLM-mediated source conversion [17,22–25].

These systems contribute substantially richer semantic encoders than our deterministic 361-dimensional representation. EvoPatch-IoT is not presented as a superior replacement encoder. Its distinct contribution is to

connect anonymous target localization with release chronology and a known upstream vulnerability reference. This difference is also why our comparison labels use “-style” when only a paper’s modeling bias is reproduced under common features; they do not imply checkpoint-level reproduction.

2.2. Binary Vulnerability Search and Retrieval

Vulnerability-oriented retrieval asks for stronger evidence than generic similarity. Practical binary verification uses patch semantics or key basic blocks to distinguish vulnerable and fixed code [7, 8]; REACT raises patch-presence reasoning to an intermediate representation [9]. BinQuery instead accepts a natural-language information need and retrieves binary code [26]. EvoPatch-IoT assumes a different but common input: the analyst already knows the upstream function identity or CVE lineage and needs a short candidate list in a stripped target. Its prototype term is therefore reference-conditioned, while its no-prototype ablation represents the harder unidentified-query setting.

2.3. IoT Firmware Security Analysis

Firmware analysis operates at several complementary layers. FirmAE supports large-scale dynamic emulation [1]; VERI and FirmCVI identify reused vulnerable components and component versions [2, 27]; recent recurring-vulnerability and reaching-definition systems search large firmware collections [4, 5]; and LuaTaint and NPFTaint specialize static data-flow analysis for web interfaces or network parsing functions [6, 28]. Dataset studies additionally show how vendor, architecture, packaging, and collection choices affect measured coverage [3].

EvoPatch-IoT is complementary rather than end-to-end: firmware unpacking, component identification, and architecture detection are assumed to have occurred before analysis. The system produces function-level candidates and evolution evidence that a broader triage pipeline can consume. Our controlled builds remove vendor packaging noise and enable exact labels, whereas real-device firmware studies provide stronger ecological validity; both are needed, and the present benchmark addresses only the former.

2.4. Positioning of EvoPatch-IoT

The closest conceptual alternatives are cross-architecture embedding, patch-presence testing, and firmware component identification. EvoPatch-IoT intersects all three but is identical to none: its geometric term is deliberately simple and target-local, its fused term is deterministic rather than pretrained, and its prototype term stores only pre-target versions of the analyst-named upstream function. The empirical question is therefore not whether each ingredient exists elsewhere, but whether their constrained composition improves auditable retrieval beyond the geometry that created the anonymous anchors. The direct ablation and paired tests in Section 6.3 answer that question and also expose that the incremental margin is modest.

3. Benchmark Construction and Problem Definition

3.1. BusyBox-Centered IoT Firmware Benchmark

Table 1 summarizes our original controlled benchmark. We first collected the official BusyBox release archive [18]; the local source inventory contains 130 releases, from which 57 versions between 1.11.0 and 1.37.0 were selected for complete binary experiments. Each selected source tree was compiled separately for AArch64, ARM, big-endian MIPS, little-endian MIPSEL, and x86_64 with the corresponding GNU cross-toolchain. The build orchestrator runs `defconfig`, applies a recorded compatibility configuration for legacy releases, invokes the architecture-specific compiler, retains the successful symbol-bearing executable, and then invokes the corresponding `strip -s` tool to create the stripped view. Builds were performed version by version rather than synthesized from existing firmware images.

The result comprises 270 unstripped and 285 stripped binaries. Fifteen early AArch64 releases have no matching unstripped artifact and are excluded when labels are required, leaving 270 paired binaries. For every artifact we retain version, architecture, build status, ELF metadata, and processing logs. Artifact manifests and the scripts needed to reconstruct the benchmark accompany the public dataset [19].

Unstripped binaries are processed with `readelf` to extract function symbols and binary metadata. This yields 1,550,752 function-symbol rows, including 1,290,369 analysis-function rows after removing evident runtime and compiler-generated records. The stripped branch is processed with headless Ghidra, which emits anonymous instruction tokens, CFG edges, call references, string categories, and constant buckets without names, DWARF paths, or version strings. Across 285 stripped binaries, Ghidra recovers 1879–3614 functions per binary (mean 2770.95), and bidirectional alignment retains 155,845 anchors.

No physical IoT device or vendor firmware image was used to create this benchmark. This is deliberate: controlled source builds isolate version and ISA effects, guarantee exact ground truth, and permit paired

stripped/unstripped analysis that commercial firmware rarely provides. The tradeoff is ecological validity. The results do not measure firmware unpacking, vendor customization, proprietary compiler flags, or real-device prevalence, and we now state that limitation explicitly rather than calling these binaries device-extracted firmware.

Table 1. Original controlled BusyBox benchmark used in this paper.

Item	Value
Historical BusyBox versions	57
Architectures	5 (AArch64, ARM, MIPS, MIPSSEL, x86_64)
Unstripped binaries	270
Stripped binaries	285
Paired stripped/unstripped binaries	270
Stripped-only binaries	15 (early AArch64 releases)
Source releases	130
Function-symbol rows	1,550,752
Analysis-function rows	1,290,369
Matched stripped functions	155,845
Recovered functions per stripped binary	1879–3614 (avg. 2770.95)
Binary size range	3.84–6.31 MB (avg. 5.25 MB)

3.2. Threat Model and Deployment Assumptions

- **Assets and adversary.** EvoPatch-IoT targets an authorized analyst auditing an extracted ELF component for a known, upstream-disclosed vulnerability. The untrusted vendor build may remove symbols, omit or falsify version strings, change optimization and layout, select another supported ISA, and selectively backport a patch. The security-relevant asset is the analyst’s ability to find and inspect the target routine without trusting vendor metadata.
- **Trusted inputs and inference modes.** We assume that firmware unpacking has produced a parseable BusyBox ELF and that its ISA is known. A curated historical archive and its function identities are trusted and cannot be poisoned by the target. In reference-guided mode, the analyst supplies a known upstream function q associated with the CVE; hence $u(q)$ is available as query metadata and selects a prototype built only from versions older than the target. Candidate identities in the stripped target are never used for scoring. In anonymous-query mode, no trusted $u(q)$ exists, so the prototype term is set to zero; the “without prototype” ablation measures this deployment restriction.
- **Security objective.** The system returns a ranked set of candidate functions, confidence-bearing alignment evidence, and historical neighbors for manual N-day verification. Success means reducing the search space while retaining the true homolog near the top. EvoPatch-IoT does not certify that a binary is patched, prove absence of a vulnerability, generate exploits, or discover previously unknown vulnerabilities.
- **Excluded threats.** Packing, encryption, self-modifying code, deliberate control-flow flattening, adversarial function splitting/merging, compromised analysis tools, and poisoning of the historical archive are outside the present model. These exclusions are reasonable for a first component-level study because the method requires recoverable function boundaries and a trusted upstream lineage, but they delimit the security claim and motivate the failure analysis in Section 7.3.

3.3. Tasks

We formalize three progressively richer tasks and separate generic retrieval from the reference identity required by historical prototypes.

- **Task 1: Anonymous cross-architecture function retrieval.** Given a stripped query q from architecture a_i and candidate set $\mathcal{C}_{a_j}$ from a different architecture a_j , rank homologous functions without using $u(q)$; only geometry and anonymous multi-view features are permitted.
- **Task 2: Reference-guided evolution-aware retrieval.** Given a trusted upstream reference function with identity $u(q)$, a target release v_q , and a stripped candidate pool, rank homologs using only target-anonymous evidence and prototypes from releases $v < v_q$. This is the primary N-day deployment task.
- **Task 3: Binary-level patch-state profiling.** Given a target binary and a CVE-related reference pattern, estimate whether aggregate evidence resembles the vulnerable or patched lineage and expose evidence functions. This output is a triage signal, not a proof of patch presence.

3.4. Evaluation Metrics

For retrieval tasks we report Hit@1, Hit@5, Hit@10, MRR@10, and Mean Inspected@10. Let $r(q)$ denote the rank of the first correct candidate for query q , clipped at 11 if no correct result appears in the top 10. Then

$$\text{MRR@10} = \frac{1}{|Q|} \sum_{q \in Q} \frac{\mathbb{I}[r(q) \leq 10]}{r(q)}. \quad (1)$$

Here Q is the query set, $|Q|$ is the number of evaluated queries, $\mathbb{I}[\cdot]$ is the indicator function that equals 1 when its condition is true and 0 otherwise, and $r(q)$ is the first-hit rank of query q after top-10 clipping. We also use the inspection-reduction view that is natural for firmware auditing:

$$\text{InspectionReduction} = 1 - \frac{\mathbb{E}[r(q)]}{\mathbb{E}[|C|]}. \quad (2)$$

In this expression, $\mathbb{E}[\cdot]$ denotes expectation over the evaluated queries, $r(q)$ is again the inspected rank of the first correct answer, C is the candidate pool for the corresponding retrieval task, and $|C|$ is the candidate-pool size. For patch-state profiling we report accuracy, precision, recall, and F1.

4. EvoPatch-IoT Method

4.1. Overview

EvoPatch-IoT is split into four layers: stripped-compatible feature extraction, anonymous alignment, multi-view representation construction, and evolution-aware retrieval. Figure 1 separates the offline trusted archive from online target analysis. On the left, official release sources are compiled one version and architecture at a time to create paired labeled and stripped views. These paired artifacts are used to establish supervision anchors and release histories; they are not presented as vendor-extracted firmware. A deployment target follows only the stripped path.

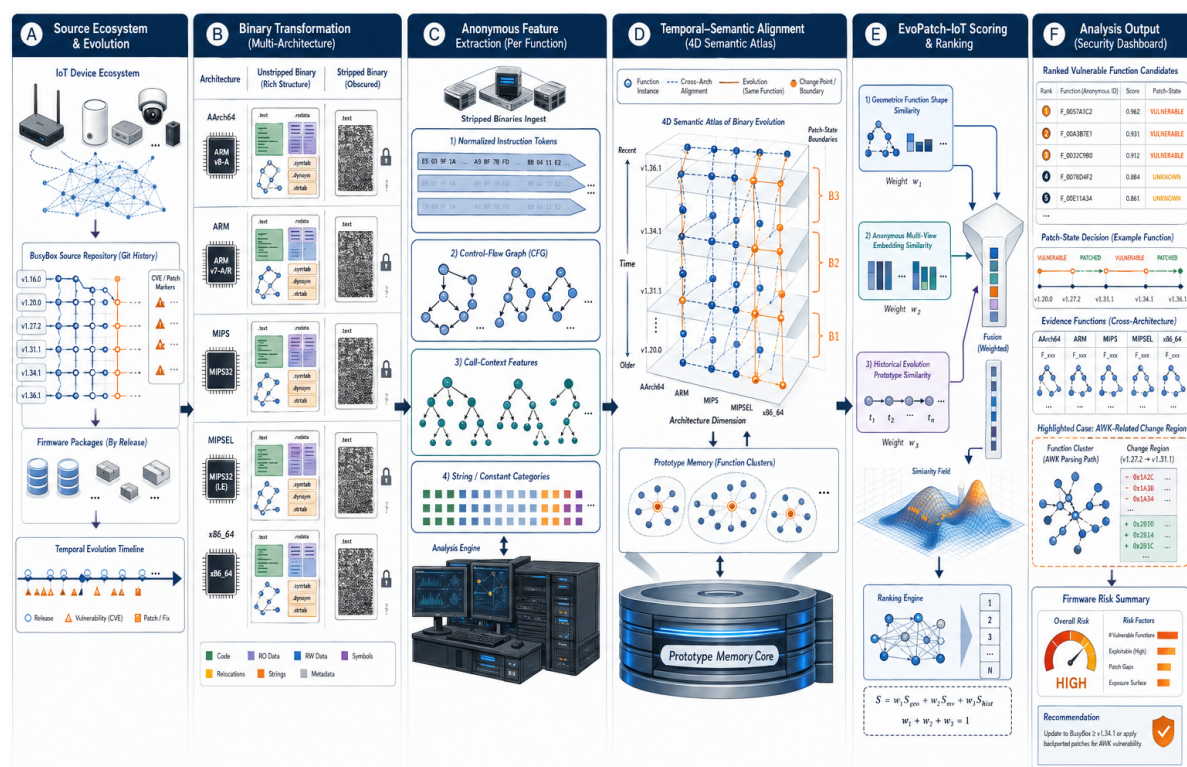


Figure 1. Overview of EvoPatch-IoT and its trust boundary. The offline trusted branch collects official BusyBox releases, compiles paired stripped/unstripped artifacts, and constructs labels and historical prototypes. At deployment, only the extracted target ELF, its ISA, and an analyst-supplied upstream reference identity enter the scorer. The target branch recovers anonymous functions, localizes candidates through bidirectional geometric anchors, fuses token/graph/context/shape evidence, and compares candidates with prototypes formed strictly before the target release. The output is a ranked audit queue, not an automated claim that the target is safe.

In the center, the analysis engine recovers anonymous functions and extracts normalized instruction tokens, control-flow statistics, call context, string categories, constants, and target-local layout geometry. Bidirectional nearest-neighbor agreement links stripped training functions to labeled historical functions, while acceptance distance and match confidence remain visible to the analyst. In the target binary, the same shape representation narrows the candidate neighborhood without consulting candidate names.

On the right, the scorer combines three distinct evidence channels. The geometry score asks whether query and candidate occupy comparable target-local structural positions; fused similarity asks whether their anonymous token/graph/context/shape views agree; and prototype similarity asks whether the candidate resembles pre-target realizations of the analyst-named function. The output is a top- k queue with component scores and historical neighbors. If no trusted query identity exists, the prototype channel is disabled rather than inferred from the target ground truth.

4.2. Stripped-Compatible Anonymous Features

For each recovered stripped function f , we build a geometric shape descriptor from its location inside the current binary:

$$\mathbf{s}(f) = [\ell(f), \eta_{\text{addr}}(f), \eta_{\text{rank}}(f), \bar{\ell}_{\mathcal{N}_2(f)}, \Delta\ell(f)]^\top, \quad (3)$$

where $\ell(f) = \log(1 + \text{size}(f))$, $\eta_{\text{addr}}(f) = \frac{\text{addr}(f) - a_{\min}}{a_{\max} - a_{\min}}$, $\eta_{\text{rank}}(f) = \frac{\text{rank}(f)}{N-1}$, and $\Delta\ell(f) = \ell(f) - \bar{\ell}_{\mathcal{N}_2(f)}$. Here $\text{size}(f)$ is the recovered function size in bytes, $\text{addr}(f)$ is the start address of f , $a_{\min}$ and $a_{\max}$ are the minimum and maximum recovered function addresses in the current binary, $\text{rank}(f)$ is the address-sorted position of f , N is the number of recovered functions in that binary, $\mathcal{N}_2(f)$ is the two-sided local neighborhood around f , and $\bar{\ell}_{\mathcal{N}_2(f)}$ is the mean log-size inside that neighborhood. This five-dimensional vector is crucial because stripped-to-unstripped correspondence is otherwise ill-posed in the absence of names.

The Ghidra exporter additionally generates anonymous instruction tokens, graph statistics, call contexts, string categories, and constant buckets. Let $\mathcal{T}(f)$ be the token multiset and $\mathcal{X}(f)$ be the context multiset. We convert both into TF-IDF weighted sparse vectors and then hash them into fixed-length dense embeddings:

$$\begin{aligned} \mathbf{t}(f) &= H_{256}(\text{TFIDF}(\mathcal{T}(f))), \\ \mathbf{c}(f) &= H_{64}(\text{TFIDF}(\mathcal{X}(f))). \end{aligned} \quad (4)$$

Here $\mathcal{T}(f)$ is the multiset of anonymous instruction-like tokens extracted from function f , $\mathcal{X}(f)$ is the multiset of anonymous context events such as calls, strings, and constants, $\text{TFIDF}(\cdot)$ converts a multiset into a sparse term-weight vector, $H_{256}(\cdot)$ and $H_{64}(\cdot)$ are fixed-dimensional hashing projections, and $\mathbf{t}(f)$ and $\mathbf{c}(f)$ are the resulting token and context embeddings.

Graph and structural statistics are summarized in two parts. First,

$$\mathbf{g}_{\text{base}}(f) = [\ell(f) \parallel \rho(f) \parallel \bar{i}_{bb}], \quad (5)$$

where

$$\ell(f) = [\log(1 + s_f), \log(1 + i_f), \log(1 + b_f), \log(1 + e_f)]$$

and

$$\rho(f) = [\rho_{\text{call}}, \rho_{\text{branch}}, \rho_{\text{ret}}, \rho_{\text{str}}, \rho_{\text{const}}, \rho_{\text{edge}}].$$

We then concatenate these summary terms with histogram views:

$$\begin{aligned} \mathbf{g}(f) &= \mathbf{g}_{\text{base}}(f) \parallel \mathbf{h}_{\text{op}}(f) \\ &\quad \parallel \mathbf{h}_{\text{edge}}(f). \end{aligned} \quad (6)$$

In these expressions, $\parallel$ denotes vector concatenation, s_f is the byte size of function f , i_f is its instruction count, b_f is its number of basic blocks, e_f is its number of CFG edges, ρ_{call} , ρ_{branch} , ρ_{ret} , ρ_{str} , ρ_{const} , and ρ_{edge} are normalized densities of call sites, branches, returns, string references, constant references, and outgoing control-flow edges, respectively, $\bar{i}_{bb}$ is the mean number of instructions per basic block, and $\mathbf{h}_{\text{op}}(f)$ and $\mathbf{h}_{\text{edge}}(f)$ are the operation-class and edge-type histograms.

4.3. Bidirectional Anonymous Alignment

The unstripped branch provides normalized function identifiers used only for supervision and evaluation. To connect stripped functions with these labels without leakage, EvoPatch-IoT performs a bidirectional mutual nearest-neighbor match inside each (v, a) bucket. The shape distance is

$$D_s(f_i, f_j) = \sum_{k=1}^5 \left(\frac{s_k(f_i) - s_k(f_j)}{\alpha_k} \right)^2, \quad (7)$$

where f_i and f_j are two functions being compared, $s_k(f)$ is the k -th component of the shape vector $s(f)$ in Equation (3), the index k runs over the five shape dimensions, and $\alpha = [1.0, 0.20, 0.20, 1.0, 1.0]$ is the positive scale vector used to normalize the contribution of each dimension in the implementation.

Algorithm 1 is an *offline labeling procedure*, not an online use of target symbols. Line 1 declares the labeled function set $\mathcal{L}_{v,a}$, stripped set $\mathcal{S}_{v,a}$, local window width w , and acceptance threshold δ for the same controlled version v and architecture a . Lines 2–5 search from each labeled function ℓ into a bounded stripped window $\mathcal{C}_\ell$ and store the nearest stripped neighbor $b_S(\ell)$. Lines 6–9 reverse the search from each stripped function s and store $b_L(s)$. Line 10 initializes $\mathcal{M}$. Lines 11–15 accept a pair only when both directions agree and $D_s \leq \delta$; this rejects one-way collisions caused by repetitive helper functions. Line 16 returns anchors for training and experimental ground truth. A deployment target follows the anonymous feature and ranking path and does not require $\mathcal{L}_{v,a}$.

This process yields 155,845 mutual matches across 270 paired binaries, averaging 577.2 anchors per binary. Figure 2d shows architecture-dependent coverage; unmatched functions remain ineligible as labeled queries rather than being silently assigned a possibly incorrect identity.

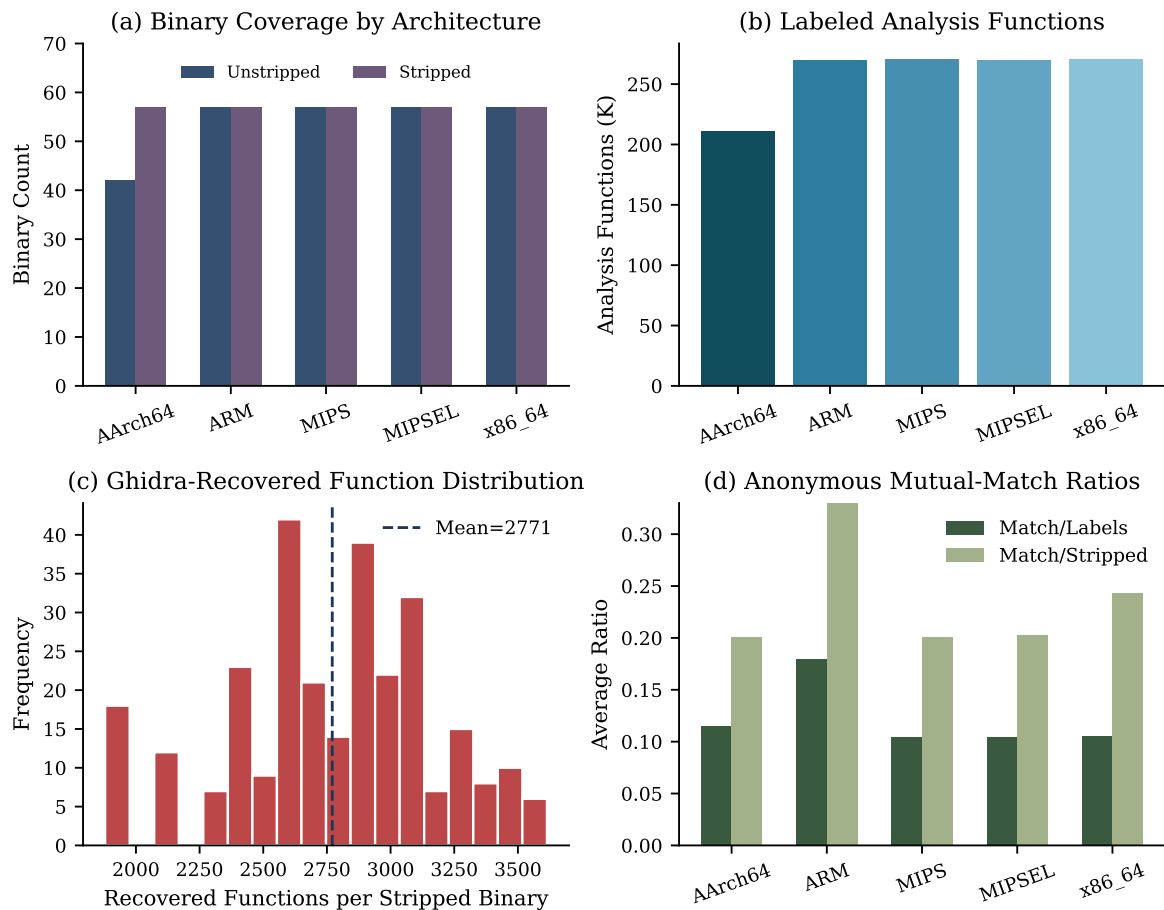


Figure 2. Controlled benchmark overview. Panels report (a) paired and stripped binary coverage by ISA; (b) analysis-function labels obtained only from the unstripped branch; (c) the number of anonymous functions recovered from each stripped binary; and (d) the retained mutual-match ratio used to quantify alignment coverage. These plots characterize the compiled component corpus; they do not represent counts of physical devices.

Algorithm 1 Offline Bidirectional Alignment for Supervision Anchors**Require:** labeled functions $\mathcal{L}_{v,a}$, stripped functions $\mathcal{S}_{v,a}$, search window w , distance threshold δ

```

1: for each  $\ell \in \mathcal{L}_{v,a}$  do
2:    $\mathcal{C}_\ell \leftarrow \text{TopWindow}(\mathcal{S}_{v,a}, s_1(\ell), w)$ 
3:    $b_S(\ell) \leftarrow \arg \min_{s \in \mathcal{C}_\ell} D_s(\ell, s)$ 
4: end for
5: for each  $s \in \mathcal{S}_{v,a}$  do
6:    $\mathcal{C}_s \leftarrow \text{TopWindow}(\mathcal{L}_{v,a}, s_1(s), w)$ 
7:    $b_L(s) \leftarrow \arg \min_{\ell \in \mathcal{C}_s} D_s(s, \ell)$ 
8: end for
9:  $\mathcal{M} \leftarrow \emptyset$ 
10: for each  $s \in \mathcal{S}_{v,a}$  do
11:   if  $D_s(s, b_L(s)) \leq \delta$  and  $b_S(b_L(s)) = s$  then
12:     add  $(s, b_L(s))$  to  $\mathcal{M}$ 
13:   end if
14: end for
15: return  $\mathcal{M}$ 

```

4.4. Architecture-Normalized Multi-View Fusion

Once anonymous anchors are available, we normalize graph and shape vectors with statistics from the training split. Let $\mu_a^{(g)}, \sigma_a^{(g)}$ be per-architecture moments for graph features, and similarly $\mu_a^{(s)}, \sigma_a^{(s)}$ for shape features:

$$\begin{aligned}\tilde{g}(f) &= \frac{g(f) - \mu_{a(f)}^{(g)}}{\sigma_{a(f)}^{(g)} + \epsilon}, \\ \tilde{s}(f) &= \frac{s(f) - \mu_{a(f)}^{(s)}}{\sigma_{a(f)}^{(s)} + \epsilon}.\end{aligned}\tag{8}$$

Here $a(f)$ is the architecture label of function f , $\mu_{a(f)}^{(g)}$ and $\sigma_{a(f)}^{(g)}$ are the training-split mean and standard-deviation vectors for graph features under architecture $a(f)$, $\mu_{a(f)}^{(s)}$ and $\sigma_{a(f)}^{(s)}$ are the corresponding moments for shape features, $\tilde{g}(f)$ is the normalized graph vector, $\tilde{s}(f)$ is the normalized shape vector, and ϵ is a small numerical-stability constant. The final fused representation is the concatenation

$$z(f) = t(f) \parallel \tilde{g}(f) \parallel c(f) \parallel \tilde{s}(f).\tag{9}$$

In Equation (9), $\parallel$ again denotes concatenation, $t(f)$ is the token embedding, $\tilde{g}(f)$ is the normalized graph vector, $c(f)$ is the context embedding, $\tilde{s}(f)$ is the normalized shape vector, and $z(f) \in \mathbb{R}^{361}$ is the final fused representation. In the current implementation, $z(f)$ is a deterministic 361-dimensional multi-view representation rather than a black-box end-to-end embedding. This design keeps the pipeline reproducible and transparent while still allowing strong retrieval performance.

4.5. Historical Evolution Prototypes

For a query targeting release v_q , the deployed prototype bank is causal with respect to version order:

$$\begin{aligned}\mathcal{T}_u^{<v_q} &= \{f \mid u(f) = u, v(f) < v_q\}, \\ p_u^{<v_q} &= \frac{1}{|\mathcal{T}_u^{<v_q}|} \sum_{f \in \mathcal{T}_u^{<v_q}} z(f).\end{aligned}\tag{10}$$

Here u is a normalized identity in the trusted upstream archive, $v(f)$ is the release containing historical function f , v_q is the target release (or the latest release known to precede an undated target), $\mathcal{T}_u^{<v_q}$ is the set of matched realizations of u strictly older than v_q , $|\mathcal{T}_u^{<v_q}|$ is its cardinality, $z(f)$ is the fused representation of f , and $p_u^{<v_q}$ is the resulting prototype. Current-target and future releases are excluded. If the target date cannot be bounded, a prototype may only use the archive cutoff declared before target ingestion; if no eligible realization exists, the prototype score is zero.

Algorithm 2 implements the deployed scorer. Line 1 makes the trust boundary explicit: $u(q)$ belongs to the analyst-supplied upstream reference, while $\mathcal{C}$ is anonymous and the archive ends before v_q . Line 2 computes query geometry and fusion once. Line 3 constructs the version-causal prototype and disables it when the query identity is unavailable. Lines 4–10 iterate over candidates. Line 5 builds candidate representations; Lines 6–8 compute geometry, fused, and eligible historical-prototype scores without accessing a candidate label; Line 9 applies the fixed design coefficients. Line 11 returns the descending rank. Thus a ground-truth target name is used only after ranking to score the experiment, never to choose $\mathbf{p}_{u(q)}^{<v_q}$ or score a candidate.

Algorithm 2 Version-Causal Evolution-Aware Retrieval Scoring

Require: trusted reference query q with $u(q)$, target version v_q , anonymous candidate pool $\mathcal{C}$, historical archive $\mathcal{H}^{<v_q}$

```

1: build  $\mathbf{s}(q)$  and  $\mathbf{z}(q)$ 
2: build  $\mathbf{p}_{u(q)}^{<v_q}$  from  $\mathcal{H}^{<v_q}$ ; use no prototype if  $u(q)$  is unavailable
3: for each  $c \in \mathcal{C}$  do
4:   compute  $\mathbf{s}(c)$  and  $\mathbf{z}(c)$ 
5:    $r_s(q, c) \leftarrow -D_s(q, c)$ 
6:    $r_f(q, c) \leftarrow \cos(\mathbf{z}(q), \mathbf{z}(c))$ 
7:    $r_p(q, c) \leftarrow \cos(\mathbf{z}(c), \mathbf{p}_{u(q)}^{<v_q})$  if an eligible prototype exists, else 0
8:    $R(q, c) \leftarrow 0.70 r_s(q, c) + 0.10 r_f(q, c) + 0.20 r_p(q, c)$ 
9: end for
10: return candidates ranked by  $R(q, c)$ 

```

Equation (10) and Algorithm 2 correspond to the deployed EvoPatch-IoT scorer:

$$R(q, c) = \lambda_s [-D_s(q, c)] + \lambda_f \cos(\mathbf{z}(q), \mathbf{z}(c)) + \lambda_p \cos(\mathbf{z}(c), \mathbf{p}_{u(q)}^{<v_q}), \quad (11)$$

In Equation (11), q is the trusted upstream query, c is an anonymous target candidate, $D_s(q, c)$ is the shape distance, $\cos(\cdot, \cdot)$ is cosine similarity, $\mathbf{z}(q)$ and $\mathbf{z}(c)$ are fused vectors, $u(q)$ is the identity supplied with the upstream reference, v_q is the target release bound, and $\mathbf{p}_{u(q)}^{<v_q}$ is its strictly historical prototype. The nonnegative coefficients $(\lambda_s, \lambda_f, \lambda_p) = (0.70, 0.10, 0.20)$ sum to one. They were selected before the full benchmark as an interpretable design choice—geometry as localization backbone, fusion as a small tie-breaker, and prototypes as historical memory—not as the optimum of a reported grid search. The revision therefore removes the earlier unsupported “empirical optimum” wording. A post-hoc 66-point, validation-only sensitivity analysis in Section 6.3 reports both the fixed point’s rank and better alternatives without changing the coefficients after seeing test results.

4.6. Binary-Level Patch-State Proxy

To approximate patch-state profiling at the binary level, we additionally build a lightweight proxy classifier from whole-binary statistics:

$$\mathbf{m}(b) = \begin{bmatrix} \text{size}_{MB}, n_{\text{sym}}, n_{\text{ana}}, \mu_{\text{size}}, \text{med}_{\text{size}}, \\ p90_{\text{size}}, p99_{\text{size}}, n_{\text{sec}}, n_{\text{debug}} \end{bmatrix}. \quad (12)$$

Here b denotes a whole BusyBox binary, size_{MB} is its file size in megabytes, n_{sym} is the number of available function symbols in the metadata branch, n_{ana} is the number of Ghidra-recovered analysis functions, μ_{size} is the mean recovered function size, med_{size} is the median recovered function size, $p90_{\text{size}}$ and $p99_{\text{size}}$ are the 90th and 99th percentiles of recovered function sizes, n_{sec} is the number of binary sections, and n_{debug} is the number of debug-related sections or debug-bearing records retained by the loader. For a specific CVE label $y \in \{0, 1\}$, the proxy predicts the closest standardized class centroid:

$$\hat{y}(b) = \arg \min_{y \in \{0, 1\}} \left\| \frac{\mathbf{m}(b) - \mu_y}{\sigma_y + \epsilon} \right\|_2. \quad (13)$$

In this classifier, $\hat{y}(b)$ is the predicted patch-state label of binary b , the two values of y correspond to the vulnerable and patched classes, μ_y and σ_y are the class-specific mean and standard-deviation vectors estimated from training binaries, ϵ is a small stabilizer, and $\|\cdot\|_2$ is the Euclidean norm. We use this proxy only as an auxiliary

binary-level study; the core contribution of the paper remains the function-level evolution-aware retrieval layer.

4.7. Complexity and Deployment Profile

The current pipeline is intentionally lightweight. Anonymous alignment uses a bounded search window instead of full quadratic pairwise matching. If w is the shape-window width, then the matching cost per paired binary is

$$T_{\text{align}} = O((|\mathcal{L}| + |\mathcal{S}|) \cdot w), \quad (14)$$

where T_{align} is the alignment cost for one paired binary, $|\mathcal{L}|$ is the number of labeled functions, $|\mathcal{S}|$ is the number of stripped functions, and w is the bounded local search-window width. This bound holds because each labeled or stripped function only compares against a local neighborhood around the primary shape coordinate. After alignment, ranking reduces to dense similarity scoring:

$$T_{\text{rank}} = O(|\mathcal{Q}| \cdot |\mathcal{C}| \cdot d), \quad (15)$$

where T_{rank} is the ranking cost, $|\mathcal{Q}|$ is the number of queries, $|\mathcal{C}|$ is the candidate-pool size, and $d = 361$ is the fused feature dimension. In our implementation, $w = 96$, the weighted candidate pool size is 609.41 functions, and EvoPatch-IoT inspects only 6.20 functions on average before hitting a correct top-10 candidate. This makes the framework practical for firmware triage pipelines that must scan large device fleets quickly.

5. Experimental Setup

5.1. Compared Methods

All methods receive the same aligned queries, target candidate pools, token hash dimension (256), context hash dimension (64), graph/shape normalization, maximum 300 queries per directed pair, and deterministic tie policy. No external baseline checkpoint is downloaded. This improves feature and candidate parity but does not constitute an exact reproduction of every cited neural system. We therefore replace ambiguous venue/year labels with either “-style” names for literature-inspired modeling biases or a dagger for in-house component controls. Table 2 gives the exact implemented scorer, which is the reproducible object being compared.

Table 2. Exact definitions and provenance of the unified comparison controls.

Displayed Method	Exact Implemented Score or Feature Emphasis	Reproduction Status/Inspiration
Size-only [†]	Negative absolute log-size gap	In-house lower-bound control
Geometry-only (ShapeStat) [†]	$-D_s(q, c)$	In-house geometry control; identical to EvoPatch-IoT’s r_s component
CLAP-style [12]	Cosine similarity of hashed anonymous instruction tokens	Feature-family control; not the original language-supervised checkpoint
TSC-Fusion [†] [12,15]	0.60 token + 0.25 normalized shape + 0.15 context	In-house token–shape–context composite
GTrans-style [13]	Cosine similarity of deterministic graph statistics	Graph-family control; not the original graph transformer
SCG-Fusion [†] [13,14]	0.40 shape + 0.35 context + 0.25 graph	In-house shape–context–graph composite
AMMF [†] [10,14]	0.55 token + 0.30 graph + 0.15 context	In-house anonymous multi-feature fusion
AMV [†] [13,26]	0.45 fused + 0.20 token + 0.20 graph + 0.15 context	In-house anonymous multi-view control
Binary2vec-style [22]	0.60 fused + 0.20 graph + 0.20 normalized shape	Feature-family control; no learned global-attention GNN
Prototype-Fusion [†] [8,9]	0.50 fused + 0.25 shape + 0.25 prototype	In-house historical-memory control
VexIR2Vec-style [17]	0.40 graph + 0.35 token + 0.20 shape + 0.05 context	IR-inspired feature-family control; no VEX-trained embedding
Ex2Vec-style [23]	0.35 token + 0.30 graph + 0.20 context + 0.15 shape	Execution-inspired feature-family control; no learned emulator
EvoPatch-IoT	0.70 geometry + 0.10 fused + 0.20 version-causal prototype	Proposed framework

[†] Implemented by us as a controlled component or composite baseline; the dagger denotes neither “ours” in the method-contribution sense nor an external published algorithm. “-style” denotes shared modeling emphasis only.

The two simplest controls require special clarification. Size-only and Geometry-only (ShapeStat) were implemented by us inside this benchmark; they are neither prior published algorithms nor contributions claimed as EvoPatch-IoT. For query q and candidate c , Size-only ranks by

$$r_{\text{size}}(q, c) = -|\log(1 + \text{size}(q)) - \log(1 + \text{size}(c))|. \quad (16)$$

Here $\text{size}(\cdot)$ is recovered byte size and a larger score means a smaller log-size gap. Geometry-only reuses the five-dimensional descriptor and distance from Equations (3)–(7):

$$r_{\text{shape}}(q, c) = -D_s(q, c). \quad (17)$$

The geometry control is intentionally stringent because it is the same r_s component that receives weight 0.70 in EvoPatch-IoT. It therefore reveals the incremental contribution of fusion and prototypes rather than acting as an independent competitor. Only EvoPatch-IoT is the proposed framework.

5.2. Protocol

We report two protocols and do not conflate them. Symmetric full-corpus comparison reproduces the original run over 57 versions, 1020 directed architecture pairs, and 128,084 queries. For a test version, its prototype excludes that same version but may contain both earlier and later releases. This protocol is useful for controlled cross-version comparison but is not a causal deployment simulation.

The new historical-only protocol removes that weakness. Feature document frequencies, architecture normalization statistics, and prototypes for target v_q are fitted only on versions $v < v_q$. The 12 versions 1.27.0–1.29.3 form a development interval used only for the 66-point sensitivity diagnostic; coefficients remain the pre-existing (0.70, 0.10, 0.20). The 15 versions 1.30.0–1.37.0 are held out chronologically, yielding 300 directed architecture pairs and 42,498 queries. Across both protocols, functions smaller than 16 bytes or four instructions are filtered, alignment uses window $w = 96$ and threshold $\delta = 0.20$, and function names are consulted only after ranking to compute correctness.

5.3. Implementation Details and Cost

Symbol extraction uses `readelf`, stripped features use headless Ghidra, and ranking uses Python 3.10/PyTorch 2.5 with `PYTHONHASHSEED=0`. The revision experiments ran on an Intel Core i9-12900KF (16 physical/24 logical cores), 15.81 GiB host RAM, and an NVIDIA GeForce RTX 3080 Ti with 12 GiB VRAM. Reconstructing all 155,845 aligned records and completing the 27-version sensitivity/temporal experiment took 402.5 s, of which 348.9 s was temporal scoring. Peak process working set was 5.02 GiB; peak CUDA reservation was 0.03 GiB because pair matrices are processed independently. The GPU is therefore an accelerator, not a capacity requirement.

The archived Ghidra export for 285 stripped binaries took 5.04 h. Mean per-binary times were 43.49 s (AArch64), 58.63 s (ARM), 82.19 s (MIPS), 84.31 s (MIPSEL), and 49.96 s (x86_64). On disk, source trees occupy 0.96 GiB, unstripped binaries 1.39 GiB, stripped binaries 0.28 GiB, and anonymous Ghidra JSONL features 2.53 GiB. Alignment is $O((|\mathcal{L}| + |\mathcal{S}|)w)$ and ranking is $O(|\mathcal{Q}||\mathcal{C}|d)$; repository-scale deployment can stream one target binary and batch its candidates, while prototype storage grows linearly with the number of historical identities rather than with every query–candidate pair.

5.4. Research Questions

We organize the evaluation around seven research questions.

- **RQ1:** How does EvoPatch-IoT compare with unified stripped-compatible controls on the symmetric full corpus?
- **RQ2:** What does each Equation (11) component contribute under historical-only held-out testing?
- **RQ3:** How sensitive is retrieval to the three fixed coefficients, and are gains statistically reliable?
- **RQ4:** Which architecture pairs remain difficult, and where does the evolution-aware score help most?
- **RQ5:** Can version-evolution signals transfer across architectures?
- **RQ6:** What limited patch-state evidence is obtained for CVE-2021-42386?
- **RQ7:** What resource, scalability, generalization, and failure constraints govern deployment?

6. Experimental Results

6.1. Alignment Quality Before Retrieval

Before considering retrieval accuracy, it is important to inspect the quality of the anonymous stripped-to-unstripped bridge. Table 3 summarizes this stage. ARM yields the highest average match ratio against labels (0.1792) and stripped functions (0.3944), while AArch64 attains the smallest median match distance (0.0358), suggesting that once a paired sample exists, its geometric anchors are especially stable. MIPS and MIPSEL recover fewer matches than ARM but remain highly consistent, which helps explain their strong pairwise retrieval behavior.

These observations also clarify why EvoPatch-IoT retains a geometry-aware backbone. Without a trustworthy anonymous bridge, historical prototypes and fused similarity cannot be exploited safely. EvoPatch-IoT is therefore best viewed as an *alignment-aware* retrieval framework rather than a generic embedding model transplanted onto firmware binaries.

Table 3. Anonymous alignment quality and extraction cost by architecture.

Arch	Matches	Match/Lbl	Match/Strp	Med.Dist	Time (s)
AArch64	23,703	0.1149	0.2005	0.0358	43.49
ARM	48,229	0.1792	0.3944	0.0988	58.63
MIPS	28,419	0.1043	0.2012	0.0847	82.19
MIPSEL	28,443	0.1042	0.2031	0.0862	84.31
x86_64	27,051	0.1054	0.2430	0.0509	49.96

6.2. RQ1: Overall Cross-Architecture Retrieval

Table 4 and Figure 3 reproduce the symmetric full-corpus comparison. EvoPatch-IoT obtains weighted Hit@1 34.56%, Hit@5 48.70%, Hit@10 56.24%, and MRR@10 40.57%. The actual strongest baseline is Geometry-only (ShapeStat), not Prototype-Fusion: it reaches 32.00% Hit@1 and 53.31% Hit@10. The corresponding absolute gains are 2.56 and 2.93 percentage points, or 8.00% and 5.50% relative. We correct both the abstract and the headline comparison accordingly.

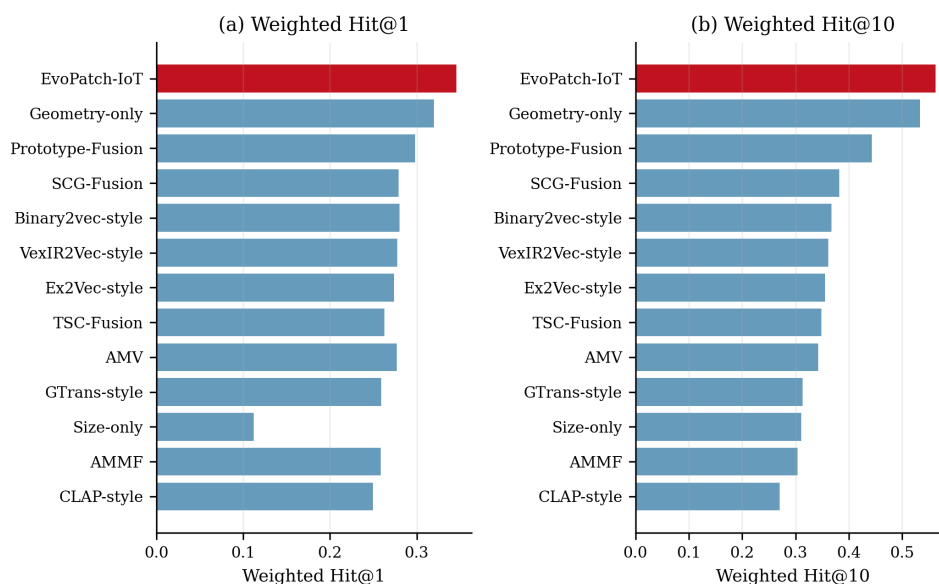


Figure 3. Symmetric full-corpus ranking under unified anonymous features. Labels ending in “-style” are feature-family controls rather than exact neural reproductions; daggered labels are in-house controls. Geometry-only (ShapeStat), which is also EvoPatch-IoT’s dominant score component, is the strongest baseline.

This is a stringent comparison because Geometry-only is exactly the r_s term used by EvoPatch-IoT and receives 70% of the final coefficient mass. The full method’s advantage is therefore incremental, not evidence that geometry is weak. Across 1020 paired architecture/version observations, the Hit@10 difference is positive for 541 pairs, tied for 435, and negative for 44; the detailed paired analysis appears under RQ3. Because this symmetric protocol permits post-target versions in a prototype, it is retained for reproducibility rather than used as the causal deployment claim.

The weighted candidate pool contains 609.41 functions and Mean Inspected@10 is 6.20, corresponding to a descriptive 98.98% inspection reduction within this benchmark. This ratio measures ranking effort after component extraction; it does not include firmware unpacking or prove that all true vulnerable routines were recovered.

Table 4. Symmetric full-corpus comparison on 57 BusyBox versions.

Method	Pairs	Queries	Weighted Hit@1	Weighted Hit@5	Weighted Hit@10	Weighted MRR@10	Mean Inspected@10
Size-only [†]	1020	128,084	0.1122	0.2551	0.3106	0.1720	8.5597
Geometry-only (ShapeStat) [†]	1020	128,084	0.3200	0.4532	0.5331	0.3771	6.5128
CLAP-style [12]	1020	128,084	0.2493	0.2607	0.2701	0.2547	8.3887
TSC-Fusion [†] [12,15]	1020	128,084	0.2627	0.3043	0.3485	0.2825	7.9224
GTrans-style [13]	1020	128,084	0.2589	0.2895	0.3133	0.2731	8.0938
SCG-Fusion [†] [13,14]	1020	128,084	0.2791	0.3340	0.3822	0.3040	7.6405
AMMF [†] [10,14]	1020	128,084	0.2583	0.2850	0.3040	0.2702	8.1511
AMV [†] [13,26]	1020	128,084	0.2768	0.3128	0.3428	0.2934	7.8592
Binary2vec-style [22]	1020	128,084	0.2799	0.3294	0.3674	0.3021	7.7002
Prototype-Fusion [†] [8,9]	1020	128,084	0.2978	0.3809	0.4433	0.3352	7.1848
VexIR2Vec-style [17]	1020	128,084	0.2772	0.3238	0.3610	0.2983	7.7520
Ex2Vec-style [23]	1020	128,084	0.2739	0.3188	0.3557	0.2943	7.7963
EvoPatch-IoT	1020	128,084	0.3456	0.4870	0.5624	0.4057	6.1957

[†] Implemented by us as a controlled component or composite baseline; the dagger does not denote an external published algorithm. Bold values indicate the best result in each column.

6.3. RQ2: Direct Historical-Only Component Ablation

Table 5 evaluates Equation (11) itself on the 15 chronologically held-out releases. The full fixed score reaches Hit@1 0.2876 and Hit@10 0.4993, compared with 0.2655 and 0.4783 for geometry alone: absolute gains of 2.20 and 2.10 percentage points. Prototype-only is much stronger than fusion-only, confirming the reviewer's observation that the historical term, rather than generic fusion, carries most of the non-geometric signal.

Removal results are more informative than isolated components. Removing geometry decreases Hit@1 by 3.83 and Hit@10 by 3.48 points. Removing the prototype decreases them by 2.12 and 2.22 points. Removing fusion decreases Hit@1 by 1.42 points and MRR@10 by 0.84 points, but *increases* Hit@10 by 0.14 points. Thus fusion improves early ordering but is not uniformly beneficial at depth ten. We now state this tradeoff plainly rather than claiming equal contribution from all three terms.

Table 5. Direct Equation (11) ablation on 15 historical-only held-out versions.

Configuration	Geometry	Fusion	Prototype	Hit@1	Hit@5	Hit@10	MRR@10
Geometry only	✓			0.2655	0.3960	0.4783	0.3226
Fusion only		✓		0.2248	0.2636	0.2924	0.2425
Prototype only			✓	0.2798	0.4195	0.4883	0.3399
Without geometry		✓	✓	0.2493	0.3917	0.4644	0.3107
Without fusion	✓		✓	0.2734	0.4202	0.5007	0.3374
Without prototype	✓	✓		0.2664	0.3958	0.4770	0.3234
Full (0.70/0.10/0.20)	✓	✓	✓	0.2876	0.4210	0.4993	0.3458

Bold values indicate the best result in each metric column.

6.4. RQ3: Coefficient Sensitivity and Statistical Reliability

We evaluate all 66 nonnegative triples on a 0.1 simplex grid whose weights sum to one, using only versions 1.27.0–1.29.3. The fixed Equation (11) point ranks 28th by validation Hit@10 (0.5445). The best Hit@10 point is (0.10, 0, 0.90) at 0.5853, while (0.20, 0.20, 0.60) gives the best Hit@1 (0.3436) and MRR@10 (0.4104). This surface shows that historical-only retrieval favors more prototype mass than the original design. We nevertheless retain the pre-revision coefficients to avoid selecting a new method after observing the benchmark, and report the surface as sensitivity evidence rather than as a tuning narrative.

Statistical tests in Table 6 use two-sided paired Wilcoxon signed-rank tests over directed architecture/version pairs. Confidence intervals are 20,000-sample percentile bootstraps that resample whole versions, preserving the 20 dependent architecture directions inside each version. On the symmetric 57-version protocol, EvoPatch-IoT improves Geometry-only by 2.56 points Hit@1 and 2.93 points Hit@10. On the stricter recent holdout, gains are 2.20 and 2.10 points. Every interval excludes zero and every paired test is significant, although statistical reliability should not be confused with a large practical effect. Figure 4 summarizes the ablation, sensitivity, pair-level difference, and recent-test results.

Table 6. Paired significance and version-block 95% confidence intervals versus geometry-only.

Protocol	Metric	Pairs	Mean Δ	95% CI	Positive/Tied/Negative	Two-Sided p
Symmetric 57-version	Hit@1	1020	0.0256	[0.0234, 0.0276]	418/480/122	2.22×10^{-59}
Symmetric 57-version	Hit@10	1020	0.0293	[0.0271, 0.0314]	541/435/44	1.63×10^{-90}
Symmetric 57-version	MRR@10	1020	0.0287	[0.0267, 0.0305]	744/123/153	4.67×10^{-88}
Historical-only recent	Hit@1	300	0.0220	[0.0202, 0.0237]	131/125/44	1.54×10^{-16}
Historical-only recent	Hit@10	300	0.0210	[0.0186, 0.0235]	154/121/25	1.10×10^{-25}
Historical-only recent	MRR@10	300	0.0232	[0.0217, 0.0245]	228/17/55	6.62×10^{-24}

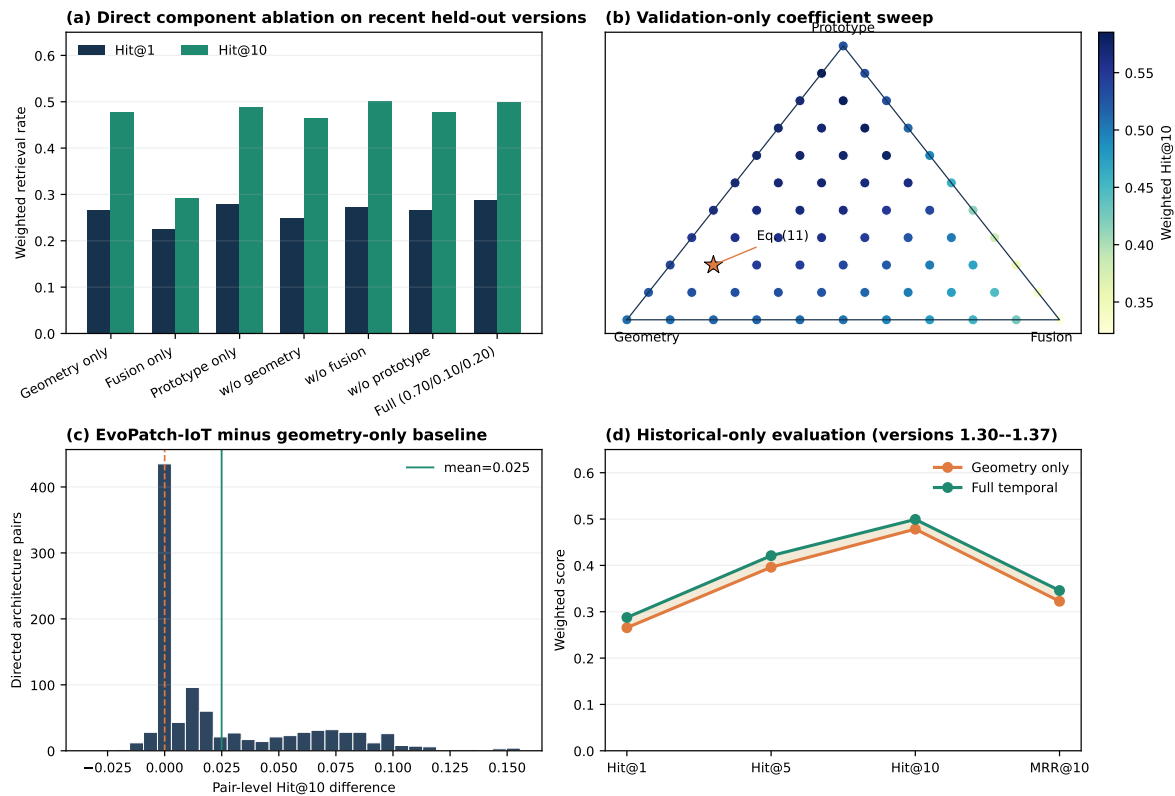


Figure 4. Revision evidence. (a) Direct historical-only ablations show that geometry and prototypes dominate, while fusion mainly improves Hit@1. (b) A validation-only simplex sweep marks the fixed Equation (11) point and reveals a prototype-heavy optimum. (c) Pair-level Hit@10 differences on the symmetric protocol are concentrated near zero but have a positive mean. (d) The full fixed score remains above geometry alone on every aggregate recent-test metric.

6.5. Descriptive Version-Wise Robustness

Under the symmetric protocol, EvoPatch-IoT is best on 56 of 57 versions and Geometry-only is best on BusyBox 1.14.0. Figure 5 visualizes this descriptive trend. Table 7 groups legacy (1.11–1.14), middle (1.21–1.29), and recent (1.30–1.37) releases. Because the symmetric prototype bank may use future releases, these curves describe benchmark difficulty and should not be interpreted as causal evidence that future-unseen evolution was handled. The historical-only RQ2 experiment supplies that evidence.

Table 7. Descriptive version-segment results under the symmetric protocol.

Method	Legacy Hit@1	Legacy Hit@10	Middle Hit@1	Middle Hit@10	Recent Hit@1	Recent Hit@10
EvoPatch-IoT	0.5280	0.7150	0.3315	0.5575	0.2919	0.5055
Geometry-only [†]	0.5160	0.6910	0.3027	0.5256	0.2657	0.4784
Prototype-Fusion [†] [8,9]	0.5102	0.6533	0.2763	0.4277	0.2440	0.3811
Binary2vec-style [22]	0.4890	0.5837	0.2583	0.3501	0.2274	0.3052

[†] Implemented by us as a controlled component or composite baseline; the dagger does not denote an external published algorithm. Bold values indicate the best result in each column.

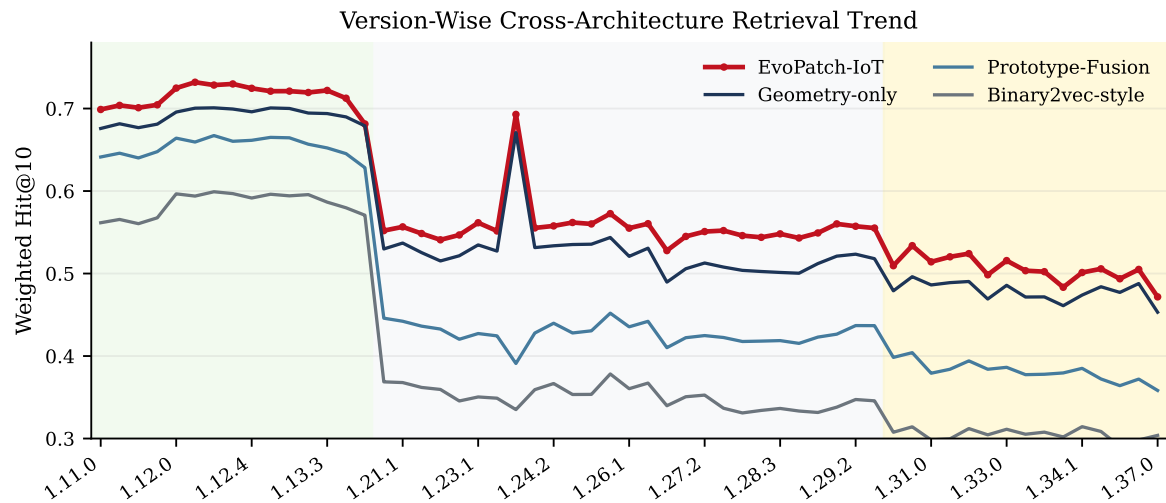


Figure 5. Version-wise weighted Hit@10 under the symmetric protocol. The shaded intervals separate legacy, middle, and recent releases. The plot is descriptive because its prototype bank excludes the target version but is not restricted to prior releases.

6.6. RQ4: Architecture-Pair Analysis

Figure 6 reports symmetric-protocol Hit@10 and absolute gain over Geometry-only. MIPS↔MIPSEL approaches 0.99 because endianness changes preserve much of the compiler’s function organization. AArch64→x86_64 and the reverse direction reach 0.8791 and 0.8541, showing that some cross-ISA pairs retain a stable candidate neighborhood.

The aggregate result hides severe failures. x86_64→ARM reaches only 0.0494 Hit@10, AArch64→ARM reaches 0.0656, and MIPS/MIPSEL→x86_64 stays near 0.056–0.057. These values are too low for autonomous decisions. They indicate layout drift, target-side helper-function ambiguity, and homogeneous post-stripping graph statistics. For these directions the system should surface a low-confidence warning and expand the analyst budget rather than treating the top result as security evidence.

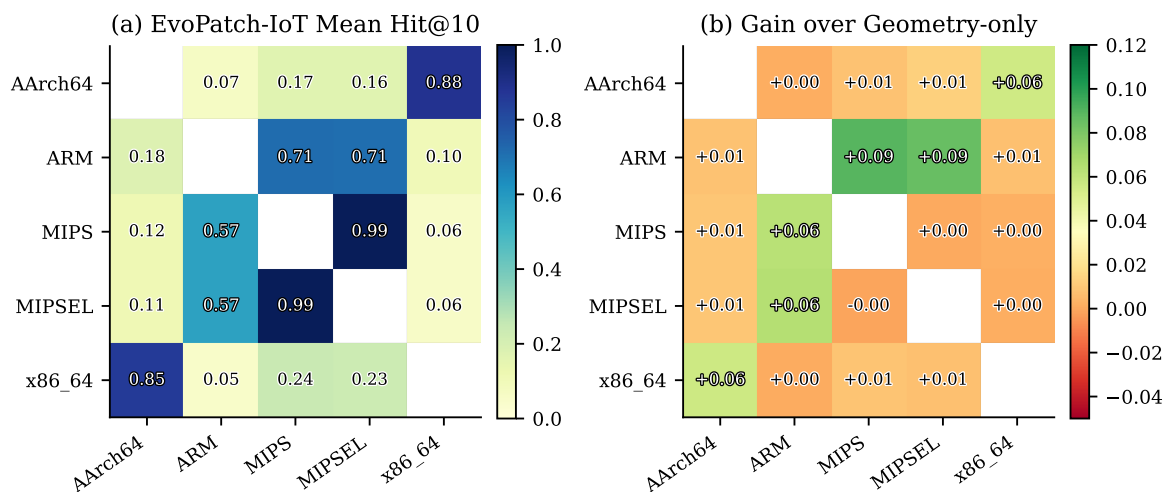


Figure 6. Symmetric architecture-pair analysis. Left: mean EvoPatch-IoT Hit@10. Right: absolute gain over Geometry-only. Very low ARM- and x86_64-target cells identify deployment failure regions even when the average gain is positive.

6.7. RQ5: Version-Change Transfer

The auxiliary version-change experiment asks whether function-size changes across adjacent releases in one architecture predict changed functions in another. Across 801 directed transitions, mean ROC-AUC is 0.9887 and mean average precision is 0.9152 at a 1.05% positive rate. MIPSEL→MIPS and its reverse exceed 0.98 average precision, while ARM→x86_64 remains above 0.86. This result establishes that coarse change events can repeat across architectures; it does not by itself validate the final retrieval weights or patch state of a particular binary.

6.8. RQ6: Limited CVE-2021-42386 Patch-State Case Study

CVE-2021-42386 affects the BusyBox AWK implementation and provides one upstream transition around versions 1.33.1 and 1.34.0 [29]. The probe finds 25 changed function/architecture instances among 30 inspected rows. The largest relative changes occur in `awk_printf` and `awk_main`, with repeated smaller changes in `awk_exit`, `awk_split`, and `awk_getline`. This is useful localization evidence, but a size delta is not equivalent to semantic proof that the vulnerability is fixed.

Table 8 exposes the complete confusion counts behind the nearest-centroid proxy. AArch64 yields 31 true positives, six true negatives, no false positives, and four false negatives. ARM, MIPS, MIPSEL, and x86_64 each yield 27/6/0/8. Their identical metric rows are therefore not a copy error: all four held-out sets contain the same 35 vulnerable and six patched version labels, and the standardized centroids produce the same four additional false-negative version decisions. We report the counts so this numerical anomaly is inspectable rather than rationalized after the fact.

Only one CVE is evaluated. Consequently, 82.44% mean accuracy and 88.47% mean F1 characterize this proxy on CVE-2021-42386 only; they do not support a general patch classifier claim. Multi-CVE validation, patch-level semantic ground truth, and vendor backports remain future work. Figure 7 summarizes the proxy metrics and AWK function-size changes.

Table 8. CVE-2021-42386 proxy with explicit confusion counts.

Held-Out	TP	TN	FP	FN	Accuracy	Precision	Recall	F1
AArch64	31	6	0	4	0.9024	1.0000	0.8857	0.9394
ARM	27	6	0	8	0.8049	1.0000	0.7714	0.8710
MIPS	27	6	0	8	0.8049	1.0000	0.7714	0.8710
MIPSEL	27	6	0	8	0.8049	1.0000	0.7714	0.8710
x86_64	27	6	0	8	0.8049	1.0000	0.7714	0.8710
Mean	—	—	—	—	0.8244	1.0000	0.7943	0.8847

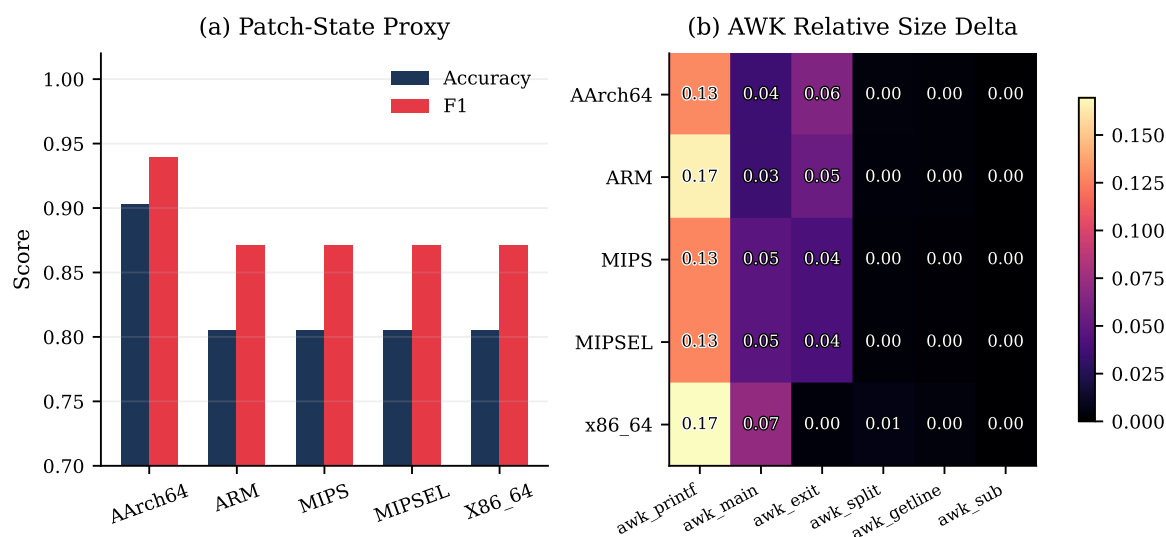


Figure 7. Single-CVE supporting evidence. (a): leave-one-architecture-out nearest-centroid proxy for CVE-2021-42386. (b): relative size deltas of six AWK functions from BusyBox 1.33.1 to 1.34.0. The repeated metric bars correspond to the explicit, identical confusion counts in Table 8; neither panel is a semantic proof of patch presence.

6.9. RQ7: Deployment Efficiency and Failure Modes

Figure 8 separates extraction cost from retrieval difficulty. MIPS and MIPSEL have the longest mean extraction times (82.19 and 84.31 s) but are the easiest targets; x86_64 requires 49.96 s yet has only 0.2312 mean target-side Hit@10. Alignment density and decompilation time therefore cannot serve as reliability proxies.

Directional asymmetry is operationally important. ARM, MIPS, and MIPSEL source queries average roughly 0.44–0.46 Hit@10, whereas AArch64 and x86_64 sources fall to 0.3192 and 0.3091. As targets, MIPS/MIPSEL

exceed 0.55 while x86_64 falls sharply. A deployment should maintain architecture-pair calibration, suppress automatic conclusions below a validation threshold, and route low-performing pairs to manual analysis.

Observed failures have concrete causes: compiler layout and calling-convention changes break address rank; function splitting, merging, and inlining distort local size neighborhoods; repetitive BusyBox helpers produce near-identical token/CFG summaries; and Ghidra boundary errors propagate into every score. Packing or adversarial obfuscation may eliminate recoverable functions entirely. Positive average significance does not remove these failure modes, especially in cells whose absolute Hit@10 is below 0.10.

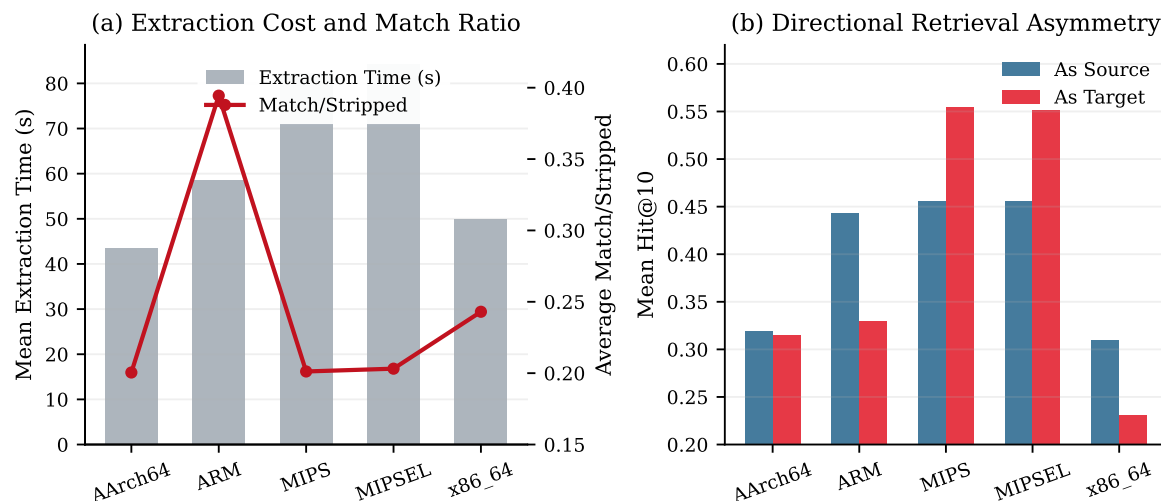


Figure 8. Operational heterogeneity. (a): per-ISA Ghidra extraction time and anonymous match ratio, which do not track retrieval difficulty monotonically. (b): symmetric-protocol Hit@10 when each ISA acts as source or target; the source/target gap motivates pair-specific confidence thresholds.

7. Discussion

The revision changes the interpretation of the result in three ways. First, target-local geometry is not a weak preliminary baseline: it is the strongest control and the dominant component of the proposed score. Second, prototypes provide most of the additional depth-ten retrieval signal in the historical-only setting; fusion provides a smaller first-rank/MRR benefit and can slightly reduce Hit@10. Third, the fixed weights are a transparent design point rather than an empirically optimal setting. The contribution is therefore a modest but statistically reliable, auditable composition over a strong geometry backbone.

7.1. Implications for IoT Auditing Workflows

EvoPatch-IoT is best used as a decision-support stage after unpacking and component identification. For a disclosed CVE, an analyst registers the upstream reference identity and an archive cutoff, analyzes the stripped target, and receives candidate-level geometry, fusion, and prototype scores. The analyst can then inspect callers, constants, and patch semantics using established patch-presence techniques [7–9]. No binary is labeled safe solely because its top candidate resembles a historical patched prototype.

For fleet-scale monitoring, prototype banks can be cached by identity and archive cutoff, while target binaries are streamed independently. MIPS↔MIPSEL pairs can support high-confidence screening; ARM- and x86_64-target jobs require larger candidate budgets and explicit low-confidence warnings. This scheduling policy follows observed pair-level accuracy, not the raw extraction time.

7.2. Generalizability Beyond BusyBox

The framework requires four properties rather than a BusyBox-specific opcode vocabulary: a recoverable native-code component, a sequence of trusted historical releases, at least some paired labeled/stripped builds for offline anchoring, and a known upstream reference identity for prototype mode. Components such as OpenSSL, Dropbear, dnsmasq, or lightweight web servers could satisfy these conditions. Token hashing, CFG statistics, address-normalized shape, and causal prototypes are otherwise component agnostic.

Generalization is not guaranteed. C++ template-heavy libraries, link-time optimization, aggressive identical-code folding, plugins, JIT-generated code, or components with frequent file-level reorganization may violate local rank stability. A new component would require rebuilding anchors and architecture statistics; the BusyBox

numbers cannot be transferred unchanged. Cross-component and real-vendor evaluation is therefore necessary before claiming a general firmware-retrieval system.

7.3. Security Analysis

The solved security problem is candidate localization for authorized N-day review under missing target metadata. Geometry limits the initial search region; anonymous multi-view evidence distinguishes nearby candidates; the historical prototype asks whether a candidate is consistent with pre-target realizations of the trusted reference. Version causality prevents future releases from influencing deployment evaluation, and hiding target names prevents direct label lookup.

Three residual security risks remain. False negatives can bury a vulnerable function outside the top- k , especially for low-performing architecture pairs; analysts must not treat absence from the list as absence of vulnerability. False positives consume reverse-engineering effort and can resemble patched code because semantic patch predicates are not modeled. Evasion through packing, control-flow flattening, function splitting/merging, or archive poisoning is outside the evaluated threat model. Cryptographic artifact manifests, trusted archive provenance, pair-specific confidence thresholds, and manual patch-semantic confirmation are required compensating controls.

7.4. Threats to Validity

- Construct validity. Function names provide experimental identity labels but are not a complete definition of semantic equivalence; compiler-generated duplicates and aliases can make top- k scoring conservative or ambiguous. Geometry is used both to create anchors and as a retrieval feature, so the benchmark favors functions whose layout is stable enough to be aligned. We expose this circularity through the Geometry-only control and do not call it an independent external baseline.
- Internal validity. The original symmetric prototype protocol admits future-version information. We preserve it only for reproducibility and add a fully historical protocol in which feature statistics and prototypes use $v < v_q$. Python hashing is fixed, candidate sampling is deterministic, and version-block confidence intervals account for dependence among architecture directions. Remaining implementation defects may still affect results; all revision CSVs and the experiment script are released for audit.
- External validity. The corpus consists of controlled BusyBox builds, not binaries extracted from purchased devices. Real firmware introduces packaging, vendor patches, proprietary toolchains, optimization diversity, and component-identification errors. BusyBox alone cannot establish transfer to other components or device families.
- Comparison validity. Literature-inspired controls share our features and candidate pools but are not exact pretrained reimplementations. Their labels now state “-style,” exact formulas are disclosed, and claims are restricted to those controls. Direct Equation (11) ablations, rather than proxy methods, support component-level conclusions.
- Patch-state validity. The nearest-centroid study covers one CVE and uses coarse binary statistics. Its identical rows are explained with confusion counts, but the experiment remains illustrative and cannot justify general patch-state detection. A multi-CVE, vendor-backport corpus with semantic patch labels is required.

8. Conclusions

This paper presented EvoPatch-IoT and an original 57-version, five-architecture BusyBox binary dataset. The corrected comparison shows 8.00% and 5.50% relative Hit@1/Hit@10 gains over the actual strongest baseline, Geometry-only, on the symmetric corpus. Under a stricter historical-only recent holdout, the fixed score improves absolute Hit@1 and Hit@10 by 2.20 and 2.10 points with confidence intervals excluding zero. Direct ablation shows that geometry and causal prototypes carry most of the benefit and that fusion offers a smaller early-ranking tradeoff. EvoPatch-IoT should therefore be viewed as an evidence-ranking tool for known-vulnerability audit, not a general patch verifier. Real-device, cross-component, adversarial-obfuscation, and multi-CVE evaluation remain necessary.

Author Contributions

Y.X.: conceptualization, methodology, software, data curation, investigation, and writing—original draft preparation; H.L.: supervision, validation, resources, and writing—review and editing; Y.S.: supervision, project administration, funding acquisition, and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding

This research was funded by the Guangdong Basic and Applied Basic Research Foundation (Grant Nos. 2025A1515010111 and 2025A1515010513) and the National Natural Science Foundation of China (Grant Nos. 72101059, 62406076, and 62506081). The funders had no role in the study design; data collection, analysis, or interpretation; manuscript preparation; or the decision to publish the results.

Institutional Review Board Statement

Not applicable. This study used public software releases and compiled binaries only; it involved no humans, human data, or animals.

Informed Consent Statement

Not applicable. This study involved no human participants or identifiable personal data.

Code and Data Availability

All source code, experiment scripts, configuration files, and reproducibility documentation are publicly available in the EvoPatch-IoT GitHub repository (<https://github.com/xxxxyyzzz3984/EvoPatch-IoT>). The original cross-version BusyBox dataset is publicly available through Science Data Bank (<https://cstr.cn/31253.11.sciencedb.34638>) [19]. It was constructed by collecting official BusyBox releases and compiling 57 selected versions one by one for five architecture targets, retaining both stripped and unstripped artifacts where available. The release also contains manifests and build information. The code repository intentionally excludes BusyBox source trees, compiled binaries, Ghidra installations, generated features, and experiment-result files; these artifacts can be regenerated locally from the released code and dataset.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

During the preparation of this work, the authors used ChatGPT-5.6 for language polishing and coding assistance. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

1. Kim, M.; Kim, D.; Kim, E.; et al. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December 2020; pp. 733–745. <https://doi.org/10.1145/3427228.3427294>.
2. Cheng, Y.; Yang, S.; Lang, Z.; et al. VERI: A Large-scale Open-Source Components Vulnerability Detection in IoT Firmware. *Comput. Secur.* **2023**, *126*, 103068. <https://doi.org/10.1016/j.cose.2022.103068>.
3. Zhang, Y.; Yu, B.; Zhou, L.; et al. Comprehensive IoT Firmware Characterization Based on a Large-Scale Firmware Dataset. In Proceedings of the 2024 9th International Conference on Cyber Security and Information Engineering, Kuala Lumpur, Malaysia, 15–17 September 2024; pp. 35–40. <https://doi.org/10.1145/3689236.3689262>.
4. Xiao, H.; Zhang, Y.; Shen, M.; et al. Accurate and Efficient Recurring Vulnerability Detection for IoT Firmware. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, 14–18 October 2024; pp. 3317–3331. <https://doi.org/10.1145/3658644.3670275>.
5. Gao, Z.; Zhang, C.; Liu, H.; et al. Faster and Better: Detecting Vulnerabilities in Linux-Based IoT Firmware with Optimized Reaching Definition Analysis. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 26 February–1 March 2024. <https://doi.org/10.14722/ndss.2024.24346>.
6. Yue, S.; Li, Q.; Zhang, G.; et al. NPFTaint: Detecting Highly Exploitable Vulnerabilities in Linux-Based IoT Firmware with Network Parsing Functions. *Comput. Secur.* **2025**, *159*, 104679. <https://doi.org/10.1016/j.cose.2025.104679>.
7. Yang, S.; Xu, Z.; Xiao, Y.; et al. Towards Practical Binary Code Similarity Detection: Vulnerability Verification via Patch Semantic Analysis. *ACM Trans. Softw. Eng. Methodol.* **2023**, *32*, 1–29. <https://doi.org/10.1145/3604608>.
8. Xu, X.; Zheng, Q.; Yan, Z.; et al. PatchDiscovery: Patch Presence Test for Identifying Binary Vulnerabilities Based on Key Basic Blocks. *IEEE Trans. Softw. Eng.* **2023**, *49*, 5279–5294. <https://doi.org/10.1109/tse.2023.3332732>.
9. Zhan, Q.; Hu, X.; Xia, X.; et al. REACT: IR-Level Patch Presence Test for Binary. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, Sacramento, CA, USA, 27 October–1 November 2024; pp. 381–392. <https://doi.org/10.1145/3691620.3695012>.

10. Li, X.; Qu, Y.; Yin, H. PalmTree: Learning an Assembly Language Model for Instruction Embedding. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Online, 15–19 November 2021; pp. 3236–3251. <https://doi.org/10.1145/3460120.3484587>.
11. Wang, H.; Qu, W.; Katz, G.; et al. jTrans: Jump-Aware Transformer for Binary Code Similarity Detection. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Online, 18–22 July 2022; pp. 1–13. <https://doi.org/10.1145/3533767.3534367>.
12. Wang, H.; Gao, Z.; Zhang, C.; et al. CLAP: Learning Transferable Binary Code Representations with Natural Language Supervision. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Vienna, Austria, 16–20 September 2024; pp. 503–515. <https://doi.org/10.1145/3650212.3652145>.
13. Zhang, Y.; Liu, Y.; Cheng, G.; et al. GTrans: Graph Transformer-Based Obfuscation-Resilient Binary Code Similarity Detection. In Proceedings of the Workshop on Binary Analysis Research (BAR) 2024, San Diego, CA, USA, 1 March 2024. <https://doi.org/10.14722/bar.2024.23006>.
14. Li, W.; Lu, J.; Xiao, R.; et al. RCFG2Vec: Considering Long-Distance Dependency for Binary Code Similarity Detection. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, Sacramento, CA, USA, 27 October–1 November 2024; pp. 770–782. <https://doi.org/10.1145/3691620.3695070>.
15. He, K.; Hu, Y.; Li, X.; et al. StrTune: Data Dependence-Based Code Slicing for Binary Similarity Detection With Fine-Tuned Representation. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 10233–10245. <https://doi.org/10.1109/tifs.2024.3484944>.
16. Jiang, S.; Fu, C.; He, S.; et al. BinCola: Diversity-Sensitive Contrastive Learning for Binary Code Similarity Detection. *IEEE Trans. Softw. Eng.* **2024**, *50*, 2485–2497. <https://doi.org/10.1109/tse.2024.3411072>.
17. VenkataKeerthy, S.; Banerjee, S.; Dey, S.; et al. VexIR2Vec: An Architecture-Neutral Embedding Framework for Binary Similarity. *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–54. <https://doi.org/10.1145/3721481>.
18. BusyBox Release Archive. Available online: <https://busybox.net/downloads/> (accessed on 17 August 2026).
19. Xiao, Y.; Li, H.; Shen, Y. Cross-Architecture Cross-Version BusyBox Binaries. Available online: <https://cstr.cn/31253.11.sciencedb.34638> (accessed on 21 September 2026).
20. Massarelli, L.; Di Luna, G.A.; Petroni, F.; et al. SAFE: Self-Attentive Function Embeddings for Binary Similarity. In Proceedings of the 16th International Conference, DIMVA 2019, Gothenburg, Sweden, 19–20 June 2019; pp. 309–329. https://doi.org/10.1007/978-3-030-22038-9_15.
21. Feng, Y.; Li, H.; Cao, Y.; et al. CRABS-former: CRoss-Architecture Binary Code Similarity Detection based on Transformer. In Proceedings of the 15th Asia-Pacific Symposium on Internetware, Macau, China, 24–26 July 2024; pp. 11–20. <https://doi.org/10.1145/3671016.3671390>.
22. Gao, Z.; Xiao, L.; Weng, W.; et al. Binary2vec: Cross-Architecture Binary Embeddings with Global Attention-Enhanced Graph Neural Networks. *Array* **2025**, *27*, 100491. <https://doi.org/10.1016/j.array.2025.100491>.
23. Gong, X.; Xu, Y.; Zhang, S.; et al. Ex2Vec: Enhancing Assembly Code Semantics with End-to-End Execution-Aware Embeddings. *Neural Netw.* **2025**, *189*, 107506. <https://doi.org/10.1016/j.neunet.2025.107506>.
24. Gu, Y.; Shu, H.; Kang, F.; et al. UniASM: Binary Code Similarity Detection without Fine-Tuning. *Neurocomputing* **2025**, *630*, 129646. <https://doi.org/10.1016/j.neucom.2025.129646>.
25. Wan, B.; Wang, S.; Wei, Z.; et al. Binary Code Similarity Detection via LLM-Based Source Code Conversion. *IEEE Internet Things J.* **2025**, *12*, 51842–51853. <https://doi.org/10.1109/jiot.2025.3579231>.
26. Zhang, B.; Gao, Z.; Wang, H.; et al. BinQuery: A Novel Framework for Natural Language-Based Binary Code Retrieval. *Proc. ACM Softw. Eng.* **2025**, *2*, 1167–1189. <https://doi.org/10.1145/3728927>.
27. Zhang, Y.; Yu, B. FirmCVI: Taint Analysis-Based Component Version Identification Method for Large-Scale IoT Firmware. In Proceedings of the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), Ocean Flower Island, China, 17–21 December 2023; pp. 920–927. <https://doi.org/10.1109/icpads60453.2023.00137>.
28. Xiang, J.; Fu, L.; Ye, T.; et al. LuaTaint: A Static Analysis System for Web Configuration Interface Vulnerability of Internet of Things Devices. *IEEE Internet Things J.* **2025**, *12*, 5970–5984. <https://doi.org/10.1109/jiot.2024.3490661>.
29. National Institute of Standards and Technology. CVE-2021-42386 Detail. Available online: <https://nvd.nist.gov/vuln/detail/CVE-2021-42386> (accessed on 17 August 2026).