APRM: Learning Binary Function Representations with Radially Ordered Context

Anonymous authors
Paper under double-blind review

Abstract

Node-level graph representation learning often requires encoding a directed, variable-size neighborhood into a fixed-dimensional representation. We study this problem in binary code similarity detection (BCSD), where compiled instances of the same source-level function are mapped to comparable representations. Compilation can change the target function and its surrounding function call graph (FCG). Therefore, absolute depth and node membership can differ across compilation pairs. However, among source-matched function nodes present in both FCGs, relative radial order and the directions of pairwise ancestor relations are often preserved. We introduce Augmented Predecessor-Radial Moments (APRM), a graph readout conditioned on the root function. APRM combines a normalized root-relative radial coordinate with signed contrasts between each node and its shortest-path predecessors within the same FCG. Across four fixed upstream encoders and four compilation conditions, APRM improves mean Macro Recall@1 from 6.48% to 14.75% on galleries containing 10,000 functions. APRM also outperforms four matched message-passing baselines, with the strongest baseline reaching 12.75% under the same retrieval protocol. On Dataset-1, an external BCSD benchmark introduced by Marcelli et al. (Marcelli et al., 2022), APRM improves retrieval without using target data for training, checkpoint selection, or adaptation.

1 Introduction

Node-level graph representation learning often requires encoding a directed, variable-size neighborhood into a fixed-dimensional representation. The representation should preserve structural relations relevant to the target node (Veličković et al., 2019; Xu et al., 2019; Dwivedi et al., 2022; Huang et al., 2024; Ying et al., 2021; Abboud et al., 2022). We study this problem in binary code similarity detection (BCSD), where compiled instances of the same source-level function are mapped to comparable representations.

Changing the compiler, target architecture, or optimization level can alter both the instruction sequence of a function and the surrounding function call graph (FCG). Recent BCSD methods use call-graph context, dependency relations, or external features to enhance function representations (Liu et al., 2018; Yu et al., 2023; Collyer et al., 2024; Valenzuela & Kinder, 2026). However, compilation can add or remove functions and change graph reachability. These methods use different forms of context, but they leave open which structural relations remain useful under compilation changes. We use the target function as the root and its reachable descendants as context. We therefore ask which relations should guide the representation under these changes.

To characterize these changes, we analyze paired compilations. FCG membership and absolute shortest-path depth can change across compilations. Symbol information identifies source-matched functions across builds. Among these descendants present in both FCGs, however, relative radial order and the directions of pairwise ancestor relations are often preserved. Figure 1 illustrates this pattern. This observation is conditional on matched descendants. It does not imply invariance of the full graph.

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

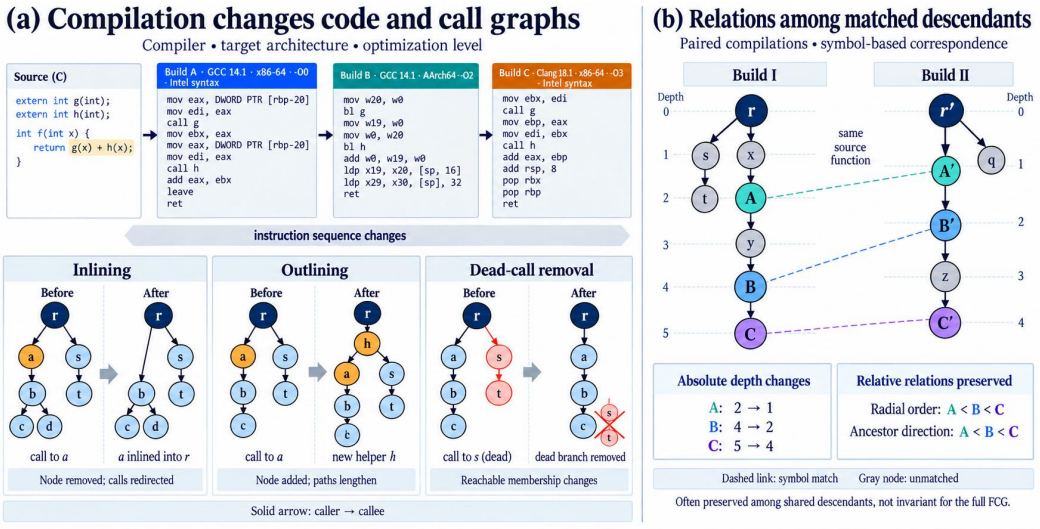

Figure 1: A Motivation Example of Compilation-Induced Code and Call-Graph Variation.

We introduce Augmented Predecessor-Radial Moments (APRM), a graph readout conditioned on the root function. APRM uses a normalized radial coordinate relative to the root and radial moments to summarize the variable-size context. It also uses signed contrasts between each node and its shortest-path predecessors to capture local direction. These predecessor relations are computed within each input FCG. They need not be preserved across compilations. APRM combines the resulting summaries with the target representation to produce one vector for retrieval. APRM does not require source correspondence or pairwise graph matching at inference.

Across four fixed upstream encoders and four compilation conditions, APRM improves mean Macro Recall@1 from 6.48% to 14.75% on 10,000-function galleries. It also outperforms four matched message-passing baselines under the same retrieval protocol. APRM further transfers to the external binary-function similarity benchmark introduced by Marcelli et al. (Marcelli et al., 2022), without using target data for training, checkpoint selection, or adaptation.

Our contributions are:

- We characterize FCG changes across compilations. Our analysis measures the preservation of relative radial order and the direction of ancestor relations among source-matched descendants.
- We introduce a graph readout that combines radial moments with signed predecessor contrasts to enhance the target function representation.
- We evaluate APRM across fixed encoders and compilation conditions, using matched controls, message-passing baselines, and transfer to an external benchmark.

2 Related Work

Graph representation learning. Graph representation learning computes node or graph representations from relational structure. Message-passing graph neural networks (GNNs) compute node representations by recursively aggregating neighboring representations, while GIN studies the expressive limits of this local mechanism (Xu et al., 2019). DGI learns node-centered patch representations together with graph-level summaries (Veličković et al., 2019). Structural and positional encodings describe local structure and node position (Dwivedi et al., 2022; Fesser & Weber, 2024). Recent work studies their stability and efficiency (Huang et al., 2024; Kanatsoulis et al., 2025), as well as positional information in directed graphs (Huang et al., 2025). Shortest-path information provides another way to encode

graph path structure (Ying et al., 2021; Abboud et al., 2022). APRM uses a graph readout conditioned on the root function. It combines a normalized radial coordinate with signed predecessor contrasts computed within each input FCG.

Binary-function representations. Graph-based binary-analysis methods use CFG-derived representations (Feng et al., 2016; Xu et al., 2017). Instruction-sequence and assembly-context encoders include SAFE, jTrans, and DiEmph (Massarelli et al., 2019; Wang et al., 2022; Xu et al., 2023). CodeART regularizes attention in a code model for stripped binaries (Su et al., 2024), while Asteria uses decompiled syntax trees (Yang et al., 2021) and TREX learns from execution traces (Pei et al., 2023). Within BCSD, Order Matters models control-flow order (Yu et al., 2020). Beyond binary-function matching, CodeSymmetries studies semantics-preserving code transformations (Pei et al., 2024). Recent methods also apply pretrained code or language models to binary-code analysis (Jiang et al., 2025; Li et al., 2025; Wang et al., 2026b). In our setting, APRM augments the representations produced by a fixed upstream encoder.

Inter-procedural context for BCSD. Prior BCSD work has incorporated inter-function context into binary-function representations (Liu et al., 2018). CFG2VEC combines control-flow and function-call graphs in a hierarchical representation, while CallGraphlets models caller and callee neighborhoods (Yu et al., 2023; Collyer et al., 2024). CFG2VEC targets cross-architectural software reverse engineering, whereas CallGraphlets supports binary-function search. Other work pretrains on call graphs to study when inter-procedural context benefits binary analysis tasks (Valenzuela & Kinder, 2026). BinEnhance applies relational graph convolution and residual fusion over call, address, data, and string relations (Wang et al., 2025). Its full system also combines embedding similarity with data-feature similarity. BinEnhance-Pro builds on this representation with a revised training procedure (Wang et al., 2026a). Co2FuLL uses shortest-path features to imported functions and strings (Dong et al., 2025). These methods show that external context can be useful for binary-function retrieval in selected settings. Like BinEnhance, APRM uses residual fusion. Its context readout combines a root-relative coordinate with signed shortest-path predecessor contrasts.

Retrieval interfaces for binary-function matching. Some methods refine candidate-level decisions through verification, alignment, or pairwise contextual comparison. Co2FuLL verifies candidates with a large language model (Dong et al., 2025). RevDecode uses relevance decoding on a directed layered graph (Ren et al., 2025). BinEGA uses pairwise alignment of CFG basic blocks (Zhou et al., 2025). Other work studies contextual matching of call-graph neighborhoods for software-version rediscovery (See et al., 2025). Recent work on neural graph representations for subgraph matching also distinguishes early and late interaction, node and edge alignment, and whole-graph versus set-based scoring (Raj et al., 2025). These approaches provide pairwise or candidate-level retrieval interfaces. In contrast, APRM produces one root-level vector from rooted FCG context. This interface supports cosine-similarity retrieval without candidate-level interaction. APRM does not require source correspondence or pairwise graph matching at inference.

3 Approach

APRM augments the representation of a target function using a rooted K-hop descendant subgraph of its FCG. Given node embeddings and function virtual address (FVA) features, the readout uses the target function as the root and conditions the aggregation on its representation. It computes a radial coordinate from shell-size distributions and signed contrasts along shortest-path predecessor relations. APRM uses shared basis functions from the same fixed cubic Bernstein basis to pool these signals, and adds the resulting context to the root state before projection and normalization. Figure 2 illustrates the overall architecture.

3.1 Problem Setup and Design Principle

Let r be a target binary function, and let $h_r(v) \in \mathbb{N} \cup \{+\infty\}$ be the directed shortest-path distance from r to v in the input FCG, with $h_r(r) = 0$ and $h_r(v) = +\infty$ when v is unreachable. For a fixed hop budget K, let $G_r = (V_r, E_r)$ be the retained directed K-hop

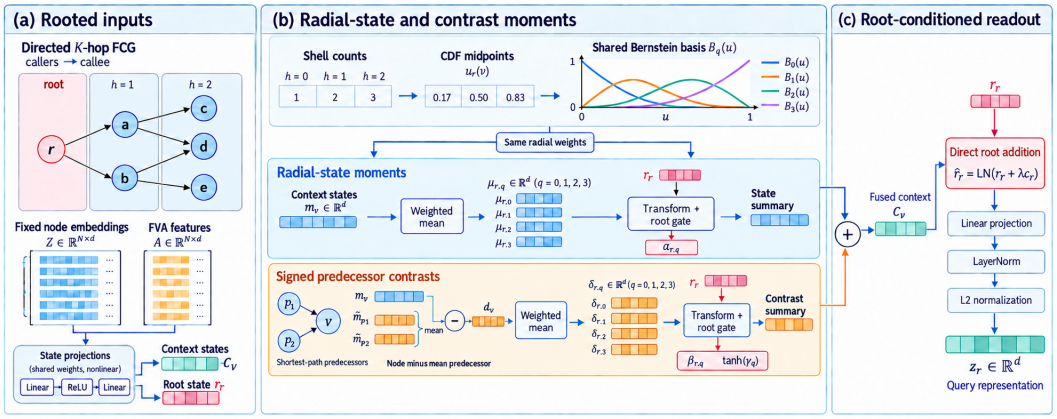

Figure 2: APRM architecture. APRM aims to construct a fixed-dimensional representation $z_r \in \mathbb{R}^{512}$ from a rooted directed K-hop function-call graph by combining fixed node embeddings and FVA features with radial-state and signed predecessor-contrast moments.

descendant subgraph rooted at r, with caller-to-callee edges. Its node set is $V_r = \{r\} \cup \mathcal{C}_r$, where $\mathcal{C}_r = \{v : 1 \le h_r(v) \le K\}$, and E_r contains the FCG edges induced by V_r.

The paired-compilation analysis suggests two structural biases for the readout. First, context is organized by relative radial position rather than raw hop labels alone. Second, edge direction is retained by comparing each node state with the mean state of its shortest-path predecessors within the same input graph. APRM implements these biases with a radial-state branch and a signed predecessor-contrast branch. For each input FCG, it obtains shortest-path predecessors from the local rooted graph and forms state contrasts along these directed relations. The root is the target function; the other retained nodes provide context.

For each $v \in V_r$, let $\bar{x}_v$ be its encoder-specific base vector. Define $q_v \in \{0, 1\}$ by $q_v = 1$ if and only if the base embedding is available; otherwise, $q_v = 0$ and $\bar{x}_v$ is the zero vector. The node input is $x_v = [\bar{x}_v; q_v]$, and X_r collects these inputs. The relation features F_r encode root-to-node function virtual address (FVA) differences and differences along shortest-path predecessor edges. APRM maps the retained graph to a 512-dimensional function representation:

$$z_r = f_\theta(G_r, X_r, F_r), \qquad z_r \in \mathbb{R}^{512}. \tag{1}$$

At inference, labels indicating source-level correspondence are unavailable and are not provided to APRM. Each function and its rooted FCG is encoded independently, and retrieval compares the resulting fixed-dimensional vectors. Source correspondence is used to construct supervised training or evaluation pairs, not as an inference-time model input.

The root state and context messages are $r_r, m_v \in \mathbb{R}^{256}$. Separate projections construct them from the node inputs. Context messages also incorporate the available FVA features. Radial moments summarize variable-size context; signed predecessor contrasts encode directed changes. FVA supplies auxiliary address information, and residual fusion preserves a direct contribution from the root state. Section 4 tests these roles with matched controls.

3.2 Radial Context Encoding

A radial shell is the set of nodes at the same shortest-path hop distance from the root. Let

$$n_r(k) = |\{v \in \mathcal{C}_r : h_r(v) = k\}|, \qquad N_r = \sum_{k=1}^{K} n_r(k).$$

The radial-state branch converts the shell counts into an empirical cumulative distribution over radial depth. We use the midpoint of the empirical-CDF interval associated with each

shell as its coordinate:

$$u_r(k) = \frac{\sum_{\ell<k} n_r(\ell) + \frac{1}{2} n_r(k)}{\max(N_r, 1)}, \qquad u_r(v) = u_r(h_r(v)). \tag{2}$$

This construction lets each retained node contribute one unit to the cumulative distribution. It gives nodes in the same shell the same coordinate and preserves the relative order of nonempty shells within the observed graph. Shell occupancy enters through the number of node contributions at each coordinate and the total radial-basis weights defined below. The coordinate is computed from one observed graph and serves as an instance-local positional statistic. Appendix A.5 gives the monotonicity argument and the resulting representation properties.

APRM first projects each node input x_v into a shared 256-dimensional space. A separate projection produces the root state r_r. For each retained context node, APRM projects the root-to-node and predecessor-edge FVA features separately. It then adds the projected features to the transformed node state to obtain m_v. The predecessor-edge feature is the mean over edges from the immediately preceding radial shell. The radial-state branch summarizes the context with four nonnegative cubic Bernstein basis functions. We call the resulting normalized Bernstein-weighted summaries radial moments:

$$B_q(u) = \binom{3}{q} u^q (1-u)^{3-q}, \qquad q \in \{0, 1, 2, 3\}. \tag{3}$$

The total radial-basis weight and radial-state moment are

$$M_{r,q} = \sum_{v \in \mathcal{C}_r} B_q(u_r(v)), \qquad \mu_{r,q} = \frac{\sum_{v \in \mathcal{C}_r} B_q(u_r(v)) m_v}{\max(M_{r,q}, 10^{-6})}. \tag{4}$$

A learned sigmoid gate $\alpha_{r,q} \in [0, 1]$, conditioned on the root representation, weights each transformed state moment. The gate includes the mask $\mathbf{1}[M_{r,q} > 0]$, so the state branch is zero for an empty context. The full gate input is specified in Appendix A.4. When $N_r = 0$, all context moments are zero under this mask. The radial context is

$$c_r^{\text{state}} = \sum_{q=0}^{3} \alpha_{r,q} T_q(\mu_{r,q}).$$

Here, $T_q : \mathbb{R}^{256} \to \mathbb{R}^{256}$ is a learned basis-specific transformation. These moments provide a fixed-dimensional summary of a variable-size rooted context while allowing the root to weight different radial regions.

3.3 Signed Predecessor Contrasts

Radial state records context at different distances. The predecessor-contrast branch captures changes along directed call paths using edges between adjacent radial shells:

$$\mathcal{P}_r(v) = \{p : (p, v) \in E_r,\ h_r(p) = h_r(v) - 1\}. \tag{5}$$

Because every retained node is reachable from the root and has positive hop distance, the shortest-path definition guarantees at least one such predecessor. Let $\widetilde{m}_p = m_p$ for a non-root predecessor and let $\widetilde{m}_p = r_r$ when $p = r$. We define the signed contrast

$$d_v = m_v - \frac{1}{|\mathcal{P}_r(v)|} \sum_{p \in \mathcal{P}_r(v)} \widetilde{m}_p. \tag{6}$$

The subtraction preserves the orientation of the change from the predecessor state to the node state; taking an absolute value would remove this direction. This contrast captures local representation changes along directed paths in the input FCG.

Every retained context node has at least one shortest-path predecessor. Therefore, the predecessor-contrast branch uses the same radial basis and total radial-basis weights as the radial-state branch:

$$\delta_{r,q} = \frac{\sum_{v \in \mathcal{C}_r} B_q(u_r(v)) d_v}{\max(M_{r,q}, 10^{-6})}. \tag{7}$$

A second learned sigmoid gate $\beta_{r,q} \in [0,1]$, also conditioned on the root representation, weights each transformed contrast moment. It uses the same empty-context mask and radial regions as the state gate, with the transformed contrast in place of the state moment. Here, $U_q : \mathbb{R}^{256} \to \mathbb{R}^{256}$ is a learned basis-specific transformation. Appendices A.4 and A.5 give exact gates and representation properties.

3.4 Fusion and Function-Level Readout

We combine the two branches using the same radial basis:

$$c_r = c_r^{\text{state}} + \sum_{q=0}^{3} \tanh(\gamma_q)\beta_{r,q}U_q(\delta_{r,q}). \tag{8}$$

Here, γ_q is a learned scalar branch scale. We initialize $\gamma_q = 0$, so the predecessor-contrast branch is inactive at initialization. The context is injected only into the root state. In the final projection, $r_r, c_r, \widehat{r}_r \in \mathbb{R}^{256}$, $W_{\text{out}} \in \mathbb{R}^{512\times256}$, and $b_{\text{out}} \in \mathbb{R}^{512}$:

$$\widehat{r}_r = \text{LN}(r_r + \lambda c_r), \qquad z_r = \text{Normalize}_2(\text{LN}(W_{\text{out}}\widehat{r}_r + b_{\text{out}}))\,. \tag{9}$$

λ scales the fused context from both branches before it is added to the root state. The final vector is normalized with a numerical floor; the zero vector remains zero. The implementation details, invariance properties, and complexity analysis are given in Appendix A. At fixed K, rooted-graph traversal and moment aggregation require $O(|V_r| + |E_r|)$ time, excluding the upstream encoder and FVA preprocessing.

4 Experiments

4.1 Research Questions and Experimental Setup

We organize the evaluation around four research questions (RQs). RQ1 asks whether APRM improves fixed-vector retrieval across encoders and compilation conditions. RQ2 asks whether paired compilations exhibit the radial-order and relational patterns used to motivate the method. RQ3 asks which components affect performance and how APRM compares with matched message-passing baselines. RQ4 asks whether the method transfers to the external Dataset-1 benchmark and how it compares with BinEnhance-Pro under scoped full-gallery and pairwise-reranking protocols.

The primary benchmark is constructed from BinKit (Kim et al., 2023). It evaluates fixed-vector retrieval from rooted, directed FCG neighborhoods. It uses four fixed encoders: Gemini, TREX, DiEmph, and CodeART, with seeds 42–44. XC changes the full compiler; XA changes architecture or bitness; XO changes the optimization level; and XM combines an architecture, bitness, or full-compiler change with the optimization shift. The exact task definitions are given in Appendix A.1. Training, validation, and test projects are disjoint. Each task has 1,000 queries and a 10,000-function gallery shared across variants. A query is matched to a compiled instance of the same source-level function, and each gallery source class contributes at most one function. The primary benchmark uses one root-level vector per function and full-gallery cosine retrieval. At inference, query and gallery functions are encoded independently; source-level correspondence is used only to define the ground-truth match for evaluation and is not provided to the model. We use Macro Recall@1 as the primary metric, computed as the equal-weight mean over XC, XA, XO, and XM. We also report Macro Recall@10 and Macro MRR@10, defined analogously. Appendices A.1–A.3 define the splits, filtering, base embeddings, and graph inputs.

The main APRM and predecessor-contrast variants keep the upstream BCSD encoders fixed. We first train and select the radial-state model, then hold the selected radial-state parameters fixed while optimizing the predecessor-contrast transforms, gates, and scales. Completed contrast variants use 14 FP32 epochs with newly initialized optimizer states; Radial-State Only uses the matched extra continuation described in Appendix B.3. The objective combines the paper-specific adaptive-margin triplet term, an InfoNCE-style contrastive term, and a Circle-loss term (Schroff et al., 2015; Chen et al., 2020; Sun et al., 2020).

The final context scale is selected on validation data using the radial-state component and then fixed at $\lambda = 1.5$ for the final evaluation.

The hop-budget scan follows a strictly validation-only protocol and selects five hops before checkpoint selection. Checkpoints use validation Macro Recall@1, with Recall@10 and the earlier epoch breaking exact ties. No test score or test-derived ranking is used to select a hop budget, checkpoint, or model. Test scores in the appendix are reported only as an additional descriptive analysis. We report three-seed means and cluster-aware uncertainty; supplementary sensitivity analyses for function pool size and hop budget are provided in Appendices C.5 and C.6.

4.2 Retrieval Performance Across Encoders (RQ1)

We compare the Base Encoder and APRM across four encoders—Gemini, TREX, DiEmph, and CodeART—and four compilation conditions: XC, XA, XO, and XM. Both methods use the same queries, 10,000-function galleries, fixed-vector cosine retrieval, and seeds 42–44. Table 1 reports the primary comparison. APRM improves Recall@1 on all four compilation conditions for every encoder. The four-encoder mean Macro Recall@1 increases from 6.48% to 14.75%, a gain of 8.27 percentage points.

Table 1: Recall@1 scores of Base Encoder and APRM across four encoders and four compilation conditions.

Model	XC	XA	XO	XM	Δ Macro Recall@1 (pp)	Macro Recall@1
Gemini	4.80	19.80	6.23	3.05	–	8.47
Gemini+APRM	**16.58**	**31.69**	**22.76**	**11.97**	**+12.28**	**20.75 ± 0.31**
TREX	7.52	7.54	12.26	3.21	–	7.63
TREX+APRM	**23.10**	**10.77**	**28.13**	**9.07**	**+10.14**	**17.77 ± 0.38**
DiEmph	1.43	5.27	2.28	1.14	–	2.53
DiEmph+APRM	**8.09**	**12.37**	**16.15**	**5.59**	**+8.02**	**10.55 ± 0.84**
CodeART	5.70	5.97	14.65	2.76	–	7.27
CodeART+APRM	**9.27**	**7.97**	**17.83**	**4.67**	**+2.66**	**9.93 ± 0.20**
Four-encoder mean	4.86	9.65	8.86	2.54	–	6.48
Four-encoder mean+APRM	**14.26**	**15.70**	**21.22**	**7.83**	**+8.27**	**14.75 ± 0.39**

Gains vary across encoders, but every project-clustered interval remains above zero (Appendix C.1).

4.3 Radial-Order Patterns Across Compilations (RQ2)

For RQ2, we analyze 150,000 BinKit positive pairs defining the training transformations. Validation and test projects, negative pairs, and retrieval outcomes are excluded. We retain only source-matched descendants observable in both rooted FCGs and compare radial order and ancestor-relation direction against the corresponding random controls. We first examine the structural observation that motivates APRM. Of the 150,000 paired examples, 58,021 contain at least two source-matched descendants observable in both rooted FCGs. Of these graph pairs, 48,552 meet the eligibility criteria for comparing radial order. Among descendants linked by symbol information to the same source-level function in both graphs, tie-aware radial-order agreement is 95.72%, compared with 40.04% under the random-control baseline (Table 2). The five-hop view gives a similar result.

Table 2: Radial-order agreement across paired compilations.

View	Relation-valid pairs	Eligible pairs	Observed agreement	Random-control agreement
Full reachable	58,021	48,552 (32.37%)	95.72%	40.04%
Five-hop	57,997	48,445 (32.30%)	95.25%	39.26%

These rates are computed on descendants that symbol information links to the same source-level function and that are present in both graphs. They provide the empirical basis for the root-relative radial coordinate. In the same analysis, the project-clustered same-direction rate for pairwise ancestor relations is 98.99%, compared with 29.95% under the corresponding random-control baseline. This statistic summarizes graph-reachability direction among function pairs linked to the same source-level roles. The definitions and coverage analysis are provided in Appendices B, B.2, and B.3.

4.4 Mechanism Controls and Matched Message-Passing Baselines (RQ3)

We compare the Base Encoder, APRM, component controls, and matched message-passing baselines under the same rooted-FCG inputs, retrieval protocol, seeds, and validation-only selection rule. Table 3 reports TREX-only controls grouped by design axis. Radial-State Only uses matched-budget continuation; the other trained controls use the fixed radial-state protocol.

Table 3: TREX ablations on the held-out test set. Scores are three-seed means (42–44) of Macro Recall@1 across XC, XA, XO, and XM. Drop from APRM is APRM minus the variant, computed before rounding. The dagger marks a matched-budget continuation rather than contrast training with fixed radial-state parameters.

Variant	Macro R@1 (percent)	Drop (pp)	Intervention
APRM	17.77	0.00	Full model; CDF midpoint
Radial coordinate			
Collapsed Radial Position	14.12	3.65	Set context coordinates to 0.5
Uniform Hop Coordinate	17.22	0.55	Replace coordinate with $h_r(v)/K$
Predecessor contrast			
Radial-State Only†	16.23	1.53	Omit contrast; continue state training
Magnitude-Only Contrast	16.21	1.56	Use magnitude contrast
All-Incoming Edges	17.25	0.52	Use all incoming edges
Root readout			
Without Root-Conditioned Gates	17.48	0.28	Remove root-conditioned gating
Without Direct Root Addition	16.41	1.36	Remove direct root addition
Broader references			
Static-Context Control	13.28	4.49	Use the static-context alternative
Base Encoder	7.63	10.14	Remove FCG context module

APRM improves Macro Recall@1 by 10.14 percentage points over Base Encoder in the TREX panel. Collapsed Radial Position reaches 14.12% while retaining signed contrasts, FVA, and the direct root addition. APRM is 3.65 percentage points higher than Collapsed Radial Position on Macro Recall@1.

Magnitude-Only Contrast reduces Recall@1 by 1.56 percentage points and All-Incoming Edges by 0.52 percentage points. Static-Context Control reduces it by 4.49 percentage points, but this control changes several components. Radial-State Only reaches 16.23% on TREX, compared with 17.77% for APRM, for a matched difference of 1.53 percentage points. Appendix C.2 reports taskwise TREX scores; the pooled intervals in Appendix B.3 instead cover all four encoders.

Without Root-Conditioned Gates reaches 17.48%, a 0.28-percentage-point drop from APRM. Its Macro Recall@10 is 40.23%, compared with 40.20% for APRM. Without Direct Root Addition reaches 16.41%, a 1.36-percentage-point drop. This control removes only the direct root addition; root state remains available to the internal gates and predecessor-contrast computation. The gate control replaces root-conditioned predecessor gates with learned per-basis scalar gates while retaining the frozen radial-state context gate. These TREX results compare the tested variants under the same protocol.

Uniform Hop Coordinate reaches 17.22%, which is 0.55 percentage points below APRM and 3.10 percentage points above Collapsed Radial Position. Its Macro Recall@10 and Macro MRR@10 are 39.59% and 23.77%, compared with 40.20% and 24.20% for APRM. Under this frozen TREX protocol, the result supports retaining radial position, while the CDF midpoint comparison remains descriptive.

On the four-encoder mean, APRM exceeds all four matched message-passing baselines. Gated-MPNN is the strongest baseline at 12.75%, compared with 14.75% for APRM (Appendix C.3).

The FVA factorial is a separate frozen checkpoint family with its own Both-FVA reference. Removing both explicit FVA inputs reduces Recall@1 by 0.41 percentage points relative to that reference (Appendix C.4).

4.5 Transfer and External Comparisons (RQ4)

We evaluate zero-shot transfer on the released Dataset-1 benchmark introduced by Marcelli et al. (Marcelli et al., 2022). We also compare APRM with BinEnhance-Pro (Wang et al., 2026a) on a common test pool. For these evaluations, checkpoint selection uses only source-side validation data; no Dataset-1 data is used for training, selection, or adaptation. The BinEnhance-Pro comparison follows a separate common-pool protocol. Full protocols are given in Appendix D.

On Dataset-1, APRM increases Macro Recall@1 from 13.06% to 16.29%, a gain of 3.24 percentage points. The checkpoints are selected on the source validation split. All four encoders have positive three-seed mean gains. This result supports target-data-free transfer on Dataset-1. Detailed task and project results are provided in Appendix E.

In an additional NOVA encoder evaluation, APRM also improves representations produced by NOVA. On the x86-64 XC/XO evaluation, Macro Recall@1 increases from 1.73% to 5.83%. This result uses an 8,192-function gallery. It is not a cross-architecture comparison and is not included in the four-encoder mean (Appendix D.1).

Table 25 includes BinEnhance-Pro's FCG-only and Full-EESG representations, the Full-EESG pairwise-reranking variant, and matched APRM rows. The comparison uses Gemini seed 44 and a common pool of 169 queries and 2,424 gallery functions per task. With full-gallery cosine ranking, APRM reaches 25.15% Macro Recall@1, compared with 13.61% for BinEnhance-Pro (FCG-only) and 16.72% for BinEnhance-Pro (Full-EESG). Under the matched ANN and pairwise-reranking protocol, APRM reaches 48.08%, compared with 42.90% for BinEnhance-Pro (Full-EESG), a difference of 5.18 percentage points. APRM also has higher ANN candidate recall (94.08% versus 80.92%). Thus, APRM ranks higher under both retrieval protocols on this common pool. This comparison is reported separately from the four-encoder, three-seed benchmark (Table 25).

Efficiency measurements are reported in Appendix F. They describe the reported implementation and are not an end-to-end comparison with systems that use pairwise reranking.

5 Conclusion

We introduced APRM, a root-conditioned graph readout for variable-size directed FCG neighborhoods in BCSD. A paired-compilation analysis showed that FCG membership and absolute shortest-path depth can change, whereas relative radial order and ancestor-relation orientation among source-matched functions are often preserved. APRM turns this observation into normalized root-relative radial moments and signed shortest-path predecessor contrasts computed within each input FCG. Across four fixed encoders and four compilation conditions, APRM raises mean Macro Recall@1 from 6.48% to 14.75% and outperforms all four matched message-passing baselines. It also transfers to a released external BCSD benchmark without target-data training, checkpoint selection, or adaptation. The structural evidence is conditional on source-level matches observable in both builds, and broader generalization remains open.

AI Use Statement

Generative AI tools were used only to aid language editing and polishing of this manuscript. The authors reviewed the generated suggestions and verified the final manuscript. The authors remain responsible for the final manuscript and submission.

Ethics Statement

This study analyzes binaries compiled from open-source software and involves neither human subjects nor personal data. Binary similarity is dual-use and can support defensive analysis as well as unauthorized reverse engineering. The submission artifact excludes credentials, private infrastructure paths, proprietary binaries, and operational vulnerability targets. Potential uses such as vulnerability search, malware analysis, and reverse engineering are not validated here: the study evaluates a BCSD retrieval benchmark with source-level labels and provides no stripped-binary or source-free node-identity assessment.

Reproducibility Statement

The paper and appendix specify the model equations, optimization and checkpoint selection protocol, task definitions, and statistical procedure. The supplementary ZIP includes remapped test metadata and pair mappings, gallery membership, deterministic scoring code, result records, and artifact checksums. It supports exact result-level replay of all primary held-out metrics from per-query outcomes and vector-level verification for two representative test-only embedding matrices. It does not include model or ablation implementations, checkpoints, training or validation embeddings, private pair files, or the means to retrain and regenerate all model outputs. The public release matrix records this artifact-release boundary. The BinEnhance common-pool result is separate from the primary BCSD benchmark.

References

Ralph Abboud, Radoslav Dimitrov, and Ismail Ilkan Ceylan. Shortest path networks for graph property prediction. In Proceedings of the First Learning on Graphs Conference, volume 198 of Proceedings of Machine Learning Research, pp. 5:1–5:25, 2022.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pp. 1597–1607, 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Joshua Collyer, Tim Watson, and Iain Phillips. Know your neighborhood: General and zero-shot capable binary function search powered by call graphlets. arXiv preprint arXiv:2406.02606, 2024. doi: 10.48550/arXiv.2406.02606.

Chaopeng Dong, Jingdong Guo, Shouguo Yang, Yi Li, Dongliang Fang, Yang Xiao, Yongle Chen, and Limin Sun. Advancing binary code similarity detection via context–content fusion and LLM verification. In Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering, pp. 304–316, 2025. doi: 10.1109/ASE63991.2025.00033.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=wTTjnvGphYj.

Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. Scalable graph-based bug search for firmware images. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 480–491, 2016. doi: 10.1145/2976749.2978370.

Lukas Fesser and Melanie Weber. Effective structural encodings via local curvature profiles. In International Conference on Learning Representations, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6188c02ccc16a7587716de2efd754033-Paper-Conference.pdf.

Yinan Huang, William Lu, Joshua Robinson, Yu Yang, Muhan Zhang, Stefanie Jegelka, and Pan Li. On the stability of expressive positional encodings for graphs. In International Conference on Learning Representations, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/ad52f9c2b232a719963a620dcf88fbe5-Abstract-Conference.html.

Yinan Huang, Haoyu Wang, and Pan Li. What are good positional encodings for directed graphs? In International Conference on Learning Representations, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/59db52d2abe2c60ec722e50e2852baad-Paper-Conference.pdf.

Nan Jiang, Chengxiao Wang, Kevin Liu, Xiangzhe Xu, Lin Tan, Xiangyu Zhang, and Petr Babkin. NOVA: Generative language models for assembly code with hierarchical attention and contrastive learning. In International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=4ytRL3HJrq.

Charilaos I. Kanatsoulis, Evelyn Choi, Stephanie Jegelka, Jure Leskovec, and Alejandro Ribeiro. Learning efficient positional encodings with graph neural networks. In International Conference on Learning Representations, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/59883a1524bf58cf2dd86d0fc8d499e2-Paper-Conference.pdf.

Dongkwan Kim, Eunsoo Kim, Sang Kil Cha, Sooel Son, and Yongdae Kim. Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned. IEEE Transactions on Software Engineering, 49(4):1661–1682, 2023. doi: 10.1109/TSE.2022.3187689. URL https://doi.org/10.1109/TSE.2022.3187689.

Litao Li, Leo Song, Steven H. H. Ding, Benjamin C. M. Fung, and Philippe Charland. Transforming generic coder LLMs to effective binary code embedding models for similarity detection. In Advances in Neural Information Processing Systems, volume 38, pp. 189001–189024, 2025. doi: 10.52202/085713-5681.

Bingchang Liu, Wei Huo, Chao Zhang, Wenchao Li, Feng Li, Aihua Piao, and Wei Zou. αDiff: Cross-version binary code similarity detection with DNN. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 667–678, 2018. doi: 10.1145/3238147.3238199.

Andrea Marcelli, Mariano Graziano, Xabier Ugarte-Pedrero, Yanick Fratantonio, Mohamad Mansouri, and Davide Balzarotti. How machine learning is solving the binary function similarity problem. In 31st USENIX Security Symposium (USENIX Security 22), pp. 2099–2116, Boston, MA, 2022. URL https://www.usenix.org/conference/usenixsecurity22/presentation/marcelli.

Luca Massarelli, Giuseppe Antonio Di Luna, Fabio Petroni, Roberto Baldoni, and Leonardo Querzoni. SAFE: Self-attentive function embeddings for binary similarity. In Detection of Intrusions and Malware, and Vulnerability Assessment, volume 11543 of Lecture Notes in Computer Science, pp. 309–329, 2019. doi: 10.1007/978-3-030-22038-9_15.

Kexin Pei, Zhou Xuan, Junfeng Yang, Suman Jana, and Baishakhi Ray. Learning approximate execution semantics from traces for binary function similarity. IEEE Transactions on Software Engineering, 49(4):2776–2790, 2023. doi: 10.1109/TSE.2022.3231621.

Kexin Pei, Weichen Li, Qirui Jin, Shuyang Liu, Scott Geng, Lorenzo Cavallaro, Junfeng Yang, and Suman Jana. Exploiting code symmetries for learning program semantics. In Proceedings of the 41st International Conference on Machine Learning, volume 235 of Proceedings of Machine Learning Research, pp. 40092–40113, 2024. URL https://proceedings.mlr.press/v235/pei24b.html.

Vaibhav Raj, Indradyumna Roy, Ashwin Ramachandran, Soumen Chakrabarti, and Abir De. Charting the design space of neural graph representations for subgraph matching. In International Conference on Learning Representations, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/255f2cd7a78060aa689c330fed9ac8b0-Paper-Conference.pdf.

Tongwei Ren, Ronghan Che, Guin R. Gilman, Lorenzo De Carli, and Robert J. Walls. RevDecode: Enhancing binary function matching with context-aware graph representations and relevance decoding. In 34th USENIX Security Symposium (USENIX Security 25), pp. 6857–6875, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/ren.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823, 2015. doi: 10.1109/CVPR.2015.7298682.

August See, Moritz Mönnich, and Mathias Fischer. Enhancing binary code similarity analysis for software updates: A contextual diffing framework. In Proceedings of the 20th ACM Asia Conference on Computer and Communications Security, pp. 1724–1740, 2025. doi: 10.1145/3708821.3736207.

Zian Su, Xiangzhe Xu, Ziyang Huang, Zhuo Zhang, Yapeng Ye, Jianjun Huang, and Xiangyu Zhang. CodeArt: Better code models by attention regularization when symbols are lacking. Proceedings of the ACM on Software Engineering, 1(FSE):562–585, 2024. doi: 10.1145/3643752.

Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. Circle loss: A unified perspective of pair similarity optimization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6397–6406, 2020. doi: 10.1109/CVPR42600.2020.00643.

Samuel Valenzuela and Johannes Kinder. Pretraining on call graphs: When binary analysis tasks profit from context. In Proceedings of the 34th IEEE/ACM International Conference on Program Comprehension, pp. 14–25, 2026. doi: 10.1145/3794763.3794795.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ.

Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang. jTrans: Jump-aware transformer for binary code similarity detection. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1–13, 2022. doi: 10.1145/3533767.3534367.

Yongpan Wang, Hong Li, Xiaojie Zhu, Siyuan Li, Chaopeng Dong, Shouguo Yang, and Kangyuan Qin. BinEnhance: An enhancement framework based on external environment semantics for binary code search. In Proceedings of the Network and Distributed System Security Symposium, 2025. doi: 10.14722/ndss.2025.240568.

Yongpan Wang, Qi Liu, Zhen Lei, Siyuan Li, Xiaojie Zhu, Zhihui Zhao, Xiaodong Gu, and Yongle Chen. BinEnhance-Pro: Enhancing binary code search by distinguishing similar but non-homologous functions. IEEE Transactions on Dependable and Secure Computing, 23(4):8537–8554, 2026a. doi: 10.1109/TDSC.2026.3685419.

Yuanda Wang, Ji Zhou, Xinhui Han, and Chao Zhang. BDLF-Qwen3: Enhanced cross-architecture binary function similarity detection through binary dynamic layer fusion. Proceedings of the AAAI Conference on Artificial Intelligence, 40(2):1222–1230, 2026b. doi: 10.1609/aaai.v40i2.37094.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In International Conference on Learning Representations, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Xiangzhe Xu, Shiwei Feng, Yapeng Ye, Guangyu Shen, Zian Su, Siyuan Cheng, Guanhong Tao, Qingkai Shi, Zhuo Zhang, and Xiangyu Zhang. Improving binary code similarity transformer models by semantics-driven instruction deemphasis. In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1106–1118, 2023. doi: 10.1145/3597926.3598121.

Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376, 2017. doi: 10.1145/3133956.3134018.

Shouguo Yang, Long Cheng, Yicheng Zeng, Zhe Lang, Hongsong Zhu, and Zhiqiang Shi. Asteria: Deep learning-based AST-encoding for cross-platform binary code similarity detection. In Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 224–236, 2021. doi: 10.1109/DSN48987.2021.00036.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In Advances in Neural Information Processing Systems, volume 34, pp. 28877–28888, 2021.

Shih-Yuan Yu, Yonatan Gizachew Achamyeleh, Chonghan Wang, Anton Kocheturov, Patrick Eisen, and Mohammad Abdullah Al Faruque. CFG2VEC: Hierarchical graph neural network for cross-architectural software reverse engineering. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, pp. 281–291, 2023. doi: 10.1109/ICSE-SEIP58684.2023.00031.

Zeping Yu, Rui Cao, Qiyi Tang, Sen Nie, Junzhou Huang, and Shi Wu. Order matters: Semantic-aware neural networks for binary code similarity detection. Proceedings of the AAAI Conference on Artificial Intelligence, 34(1):1145–1152, 2020. doi: 10.1609/aaai.v34i01.5466.

Shize Zhou, Lirong Fu, Peiyu Liu, and Wenhai Wang. BinEGA: Enhancing DNN-based binary code similarity detection through efficient graph alignment. In 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 488–499, 2025. doi: 10.1109/SANER64311.2025.00052.

A Implementation and Training

This section supplements the model in Section 3 and the evaluation protocol in Section 4. Sections B–F provide the analysis and experiments discussed in Section 4.

A.1 Data and retrieval protocol

A source-function identity consists of package and version, binary name, and function name. It links compiled instances derived from the same source-level function, not independently written programs with equivalent behavior. Each binary instance also records compiler, architecture, bitness, optimization level, and function virtual address (FVA).

Training, validation, and test projects are disjoint. Validation uses gdbm, libiconv, recutils, gcal, and gsl. Testing uses a2ps, gss, gawk, inetutils, and readline.

The corrected structural filter removes confirmed MIPS GOT/PLT transfer veneers, direct aliases, and forwarding wrappers. It retains getters, setters, constant-return functions, return-only functions, and indirect function-pointer dispatch. Shortness alone is not an exclusion criterion. All reported retrieval scores use this filter.

"Full compiler" means compiler family and complete version string. XC changes the full compiler while keeping the architecture, bitness, and optimization setting fixed. XA changes architecture or bitness while keeping full compiler and optimization fixed. XO changes $O0 \rightarrow \{O3, Os, Ofast\}$ while keeping architecture, bitness, and full compiler fixed. XM

changes architecture, bitness, or full compiler together with the optimization shift. XC and XM share 100 query-positive pairs; no other exact pairwise overlap occurs. The four-task macro is an equal-weight descriptive average.

Each task gallery contains 1,000 designated positives and 9,000 additional negatives. Other designated positives act as distractors for a query. The additional negative pool excludes the query and its source-identity class. Each gallery source class contributes at most one function.

A.2 Base embeddings and training triplets

Tables 4 and 5 specify the training inputs and fixed encoder checkpoints. Anchor and positive functions share a source-function identity. We mine negatives from a global base-embedding index, excluding the anchor, positive transformation group, and disallowed projects or splits. During optimization, we select the hardest negative from the current batch and a cache of at most 4,096 embeddings.

Table 4: Encoder-specific enhancer training inputs. The base dimension excludes the appended embedding-availability bit.

Models	Triplets	Base dimension	Train batch
Gemini	71,888	64	128
TREX	133,837	768	16
DiEmph	75,823	768	16
CodeART	120,450	768	16

Table 5: Fixed base-vector sources used by the enhancer.

Models	Checkpoint source	Dimension
Gemini	s2v-annotations-1601	64
TREX	similarity/checkpoint_best.pt	768
DiEmph	binkit-diemph/finetune_epoch_10	768
CodeART	released_binary_similarity_checkpoint, commit 0f787140e51e	768

Base vectors are ℓ_2-normalized before cosine retrieval and are not whitened.

A.3 Graph construction and input features

Inputs are directed caller-to-callee FCGs. We retain the root and descendants within five shortest-path hops. Parallel call sites become one topology edge. Cycles remain, and each node uses its minimum distance from the root. External or indirect targets absent from the function table are not added.

Each graph stores one base embedding and one embedding-availability bit per node. The bit is one exactly when the base embedding is available; otherwise, the embedding is stored as a zero vector and the bit is zero.

Each graph also stores a root index, caller-to-callee edge indices, a three-dimensional root-to-node FVA relation, and a three-dimensional edge FVA relation. For a node or edge, let

$$\Delta = \text{int64(dst FVA)} - \text{int64(src FVA)}.$$

The encoded feature is

$$[\text{sign}(\Delta),\ \log(1+|\Delta|),\ \text{midrank}(|\Delta|)]\,.$$

The tie-aware absolute-delta midrank is normalized to $[0,1]$ within each rooted graph. Node deltas use the root FVA as source; edge deltas use the caller and callee endpoints. The root,

self-loops, missing attributes, and inactive entries are encoded as zero while their validity metadata is retained. The edge-FVA message for a node is the mean over predecessor edges that connect it to the immediately preceding radial shell.

A.4 Forward computation

Using the inputs x_v from Section 3.1, we set the hidden dimension to 256 and the LoRA rank to 16. The shared projection is

$$e_v = \mathrm{LN}(\mathrm{LoRA}_{16}(\mathrm{LN}(x_v))) .$$

The transformed root and context paths apply LoRA, LayerNorm, and SiLU to e_v, with separate linear projections for the three-dimensional node and predecessor-edge FVA features. The radial-state and predecessor-contrast branches use four cubic Bernstein moments. Using the notation from Section 3, the exact gates are

$$\begin{aligned} \alpha_{r,q} &= \mathbf{1}[M_{r,q} > 0]\, \sigma\left(g_q\left[r_r; T_q(\mu_{r,q}); \log(1+N_r); \frac{M_{r,q}}{\max(N_r,1)} \right] \right), \\ \beta_{r,q} &= \mathbf{1}[M_{r,q} > 0]\, \sigma\left(a_q\left[r_r; U_q(\delta_{r,q}); \log(1+N_r); \frac{M_{r,q}}{\max(N_r,1)} \right] \right). \end{aligned} \tag{10}$$

Here $[\,;\,]$ denotes vector concatenation. The maps g_q and a_q are learned two-layer scalar-output networks whose sigmoid outputs are conditioned on the root representation. Both gates use the same total radial-basis weight. Because every retained context node has a shortest-path predecessor, no separate predecessor mask is required. Each contrast scale is initialized to zero, so the predecessor-contrast branch initially contributes nothing.

The final operator uses $\mathrm{Normalize}_2(y) = y/\max(\|y\|_2, 10^{-12})$. It returns a unit-norm vector when $\|y\|_2 \geq 10^{-12}$. The zero vector remains zero, while a nonzero vector below this threshold is scaled by the same numerical floor.

A.5 Representation properties

The coupled state-and-contrast representation has the following properties. Setting all $\gamma_q = 0$ recovers the radial-state path exactly. The contrast branch can distinguish inputs with identical radial-state moments when their signed predecessor moments differ after the learned transforms, gates, fusion, and readout.

Node reindexings that preserve the rooted graph, features, and directed edges leave the moment sums unchanged. An empty reachable context contributes zero and falls back to the root path.

The radial coordinate is strictly increasing across nonempty shells. For any $a < b$ with $n_r(a) > 0$ and $n_r(b) > 0$,

$$u_r(b) - u_r(a) = \frac{\frac{1}{2}n_r(a) + \sum_{a<\ell<b} n_r(\ell) + \frac{1}{2}n_r(b)}{\max(N_r,1)} > 0.$$

A.6 Optimization and checkpoint selection

For normalized outputs, define

$$d_i^+ = 1 - \langle z_{a_i}, z_{p_i} \rangle, \qquad d_i^- = 1 - \langle z_{a_i}, z_{\tilde{n}_i} \rangle.$$

The adaptive margin and triplet term are

$$m_i = 0.25 + 0.65\, \sigma(4(d_i^- - d_i^+)), \qquad \mathcal{L}_{\mathrm{tri}} = \frac{1}{B}\sum_i [d_i^+ - d_i^- + m_i]_+.$$

The complete objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{tri}} + 0.2\mathcal{L}_{\mathrm{NCE}} + 0.03\mathcal{L}_{\mathrm{circle}}.$$

InfoNCE uses temperature 0.02. Circle loss uses margin 0.25 and scale 40. The paired positive is the diagonal target in the anchor–positive similarity matrix; off-diagonal positives and an explicit negative block enter the denominator. Selection indices are not differentiated. Current-batch negatives retain gradients for triplet and Circle terms, whereas extra InfoNCE cache negatives are detached.

All runs use AdamW with learning rate 4×10^{-4}, weight decay 3×10^{-3}, gradient-norm clipping at 1.0, and cosine warm restarts with $T_0 = 3200$, $T_{\text{mult}} = 2$, and minimum learning rate 5×10^{-7}. Training uses 14 epochs, dropout 0.3, hidden dimension 256, output dimension 512, LoRA rank 16, and FP32 arithmetic.

Table 6: Parameter status for the final APRM protocol.

Parameter group	Status in epochs 1–14
Radial-state parameters	fixed
Predecessor-contrast transforms and gates	trainable
Predecessor-contrast scales	trainable
Residual connection/output projection	fixed

The optimizer contains only the trainable parameters in Table 6. Batches with no trainable loss path are skipped without advancing the optimizer, scheduler, gradient accumulation, or global step.

Fixing the radial-state parameters does not fix the final output vector. The contrast branch still changes the input to the shared normalization and readout. Zero contrast scales recover the radial-state model at initialization, but do not guarantee nondecreasing retrieval accuracy.

For each epoch e, the selector computes

$$S_1(e) = \frac{1}{4}\left(R@1_{\text{XC}} + R@1_{\text{XA}} + R@1_{\text{XO}} + R@1_{\text{XM}}\right)$$

and the analogous $S_{10}(e)$. The selection key is

$$(S_1(e), S_{10}(e), -e)\,.$$

Macro Recall@1 is primary. Macro Recall@10 is compared only when Recall@1 is exactly tied, and the lower epoch resolves a complete tie.

Similarities are recomputed in CPU/NumPy float64 and sorted with the stable-tie policy. Converting stored FP32 embeddings to float64 does not recover discarded mantissa bits; it only avoids additional loss while accumulating dot products, norms, and ranking scores.

Training uses NVIDIA A800-SXM4-80GB GPUs on a dual-socket Intel Xeon Platinum 8358 host. We use Python 3.12.4, PyTorch 2.6.0, CUDA 12.6, and FP32 without automatic mixed precision. ANN measurements use Annoy with 10 trees.

B Radial-Order Analysis

This section gives the definitions and coverage analysis for Section 4.3.

B.1 Source correspondence and metrics

The analysis uses the 150,000 BinKit positive pairs that define training transformations. It excludes validation and test projects, negative pairs, and retrieval outcomes. Topology is taken from the FCG edge lists.

We match descendants through a deterministic 64-bit hash of source file, start line, source-level function name, project, version, and library. When several binary nodes share an identity, we use the minimum root-to-node hop.

For a graph pair obtained from paired compilations, let U_r and $U_{r'}$ be the source-identified context-node sets retained and observable in the two rooted graphs. Let $S_r \subseteq U_r$ and $S_{r'} \subseteq U_{r'}$ be the subsets of nodes that share a canonicalized source-level identity across the two graphs. After canonicalization by source identity, $\pi : S_r \to S_{r'}$ denotes the correspondence between these shared nodes. We refer to the corresponding compiled functions as source-matched functions. APRM computes its structural features from each input FCG. The motivating order-preservation condition is

$$h_r(v) < h_r(w) \implies h_{r'}(\pi(v)) \leq h_{r'}(\pi(w)), \qquad v, w \in S_r. \tag{11}$$

The weak inequality allows two depths in one graph to collapse into one radial shell in the other graph, while the analysis accommodates changes in graph membership.

For source-mapped nodes shared by a binary pair, define the tie-aware radial relation

$$\rho_r(v, w) = \mathrm{cmp}(h_r(v), h_r(w)) \in \{-1, 0, +1\}, \tag{12}$$

where cmp returns the sign of the distance difference. For the eligible unordered shared-node pairs $\mathcal{P}_{r,r'}$, agreement is

$$A(r, r') = \frac{1}{|\mathcal{P}_{r,r'}|} \sum_{(v,w)\in\mathcal{P}_{r,r'}} \mathbf{1}[\rho_r(v, w) = \rho_{r'}(\pi(v), \pi(w))]\,. \tag{13}$$

Strict inversion counts reversals among pairs strictly ordered in both graphs:

$$I(r, r') = \frac{\sum_{(v,w)\in\mathcal{P}_{r,r'}} \mathbf{1}[\rho_r(v, w) = -\rho_{r'}(\pi(v), \pi(w)),\ |\rho_r(v, w)| = |\rho_{r'}(\pi(v), \pi(w))| = 1]}{\sum_{(v,w)\in\mathcal{P}_{r,r'}} \mathbf{1}[|\rho_r(v, w)| = |\rho_{r'}(\pi(v), \pi(w))| = 1]}. \tag{14}$$

We define $I(r, r')$ only when the denominator is nonzero; graph pairs with no strictly ordered pair in both graphs are excluded from this statistic. For tie rates p_L and p_R, the random-control baseline preserves the observed radial-shell multiplicities while removing source-specific pairing. Its expectation is

$$p_L p_R + \frac{1}{2}(1 - p_L)(1 - p_R). \tag{15}$$

Ancestor-relation analysis. The companion analysis uses every unordered pair of shared source roles. For source roles a and b, it records $a \prec_r b$ when a retained node with role a reaches a retained node with role b in the rooted FCG; it records $b \prec_r a$ analogously, and otherwise records an incomparable relation (or a cyclic/duplicate relation when both directions hold). A same-direction relation is counted when the two paired graphs report the same comparable orientation. This statistic summarizes graph reachability direction between shared source roles and provides the empirical basis for a relation-aware operator. The primary unique-role scope excludes a pair when either source role maps to multiple nodes in either graph. APRM instantiates the corresponding direction-aware computation with shortest-path predecessors available in each input graph.

B.2 Shared-node coverage

Let $\mathcal{S}$ be the source-matched descendants observed in both graphs. Let $\mathcal{D}_a$ and $\mathcal{D}_b$ contain the non-root nodes retained within five hops in each graph. The symmetric shared-node coverage is

$$C_{\text{shared}} = \frac{2|\mathcal{S}|}{|\mathcal{D}_a| + |\mathcal{D}_b|}.$$

Coverage measures shared nodes; eligibility measures graph pairs that meet the minimum node-count and depth requirements. The relation-first analysis motivates the APRM design, and APRM does not use source correspondence at inference.

Table 7: Eligibility and coverage resolution for the radial-order audit. The full-reachable row reproduces the headline statistic; the deployed five-hop row uses the same hop budget as APRM inference. Relation-valid means at least two shared source-identified descendants.

View	Rows	Root IDs	≥ 1 shared	≥ 2	≥ 3	≥ 3, depth ≥ 2	Rate (%)
Full reachable	150,000	150,000	74,994	58,021	50,469	48,552	32.37
Deployed 5-hop	150,000	150,000	74,978	57,997	50,362	48,445	32.30

Table 8: Overall radial-order agreement after aligning the coverage definition with the deployed five-hop view. The coverage-weighted estimate weights each eligible pair by C_{shared}; its interval is a 10,000-replicate project-cluster bootstrap.

View	Eligible	Mean C_{shared}	Pair mean A	Pair mean null	Wtd. A / null	Wtd. margin (95% CI, pp)
Full reachable	48,552	–	95.72	40.04	–	55.68
Deployed 5-hop	48,445	0.789	95.25	39.26	95.81 / 39.48	56.33 [54.10, 57.91]

Table 9: Deployed five-hop eligibility and coverage-weighted radial-order results by build-condition task. A is observed tie-aware agreement; null is the shell-multiplicity-matched null.

Task	Rows	Rel. valid	Eligible	Rate (%)	Mean C_{shared}	Wtd. A / null	Wtd. margin (pp)
XA	43,253	17,008	14,051	32.49	0.392	0.993 / 0.367	62.66
XC	28,229	8,393	6,419	22.74	0.328	0.915 / 0.372	54.31
XM	35,106	8,626	6,649	18.94	0.232	0.907 / 0.363	54.40
XO	43,412	23,970	21,326	49.12	0.570	0.961 / 0.426	53.55

Table 10: Deployed five-hop eligibility and coverage-weighted radial-order results by compiler-family direction.

Compiler pair	Rows	Rel. valid	Eligible	Rate (%)	Mean C_{shared}	Wtd. A / null	Wtd. margin (pp)
clang->clang	66,177	28,264	23,820	35.99	0.420	0.961 / 0.395	56.59
clang->gcc	17,757	4,584	3,418	19.25	0.270	0.914 / 0.363	55.10
gcc->clang	11,126	2,826	2,165	19.46	0.264	0.905 / 0.368	53.73
gcc->gcc	54,940	22,323	19,042	34.66	0.429	0.967 / 0.402	56.47

Table 11: Project-stratified deployed five-hop coverage audit. Rows with no eligible pair have undefined relation statistics and are shown with dashes. The overall confidence interval is clustered by project; project rows are descriptive point estimates.

Project	Rows	Eligible	Rate (%)	Mean C_{shared}	Wtd. margin (pp)
binutils	65,155	22,634	34.74	0.395	58.15
bool	396	82	20.71	0.493	55.64
ccd2cue	222	92	41.44	0.558	52.12
cflow	1,330	405	30.45	0.417	52.42
coreutils	33,703	14,846	44.05	0.503	53.66
cpio	870	332	38.16	0.431	56.94
cppi	246	65	26.42	0.242	48.91
dap	324	54	16.67	0.366	50.19
datamash	1,095	355	32.42	0.423	57.08
direvent	2,149	669	31.13	0.381	57.20
findutils	2,606	1,078	41.37	0.408	57.11
glpk	4,299	643	14.96	0.298	59.19
gmp	4,225	96	2.27	0.147	51.63
gnu-pw-mgr	2	0	0.00	0.000	–
gnudos	1,388	419	30.19	0.468	58.41
grep	1,148	300	26.13	0.284	55.42
gsasl	747	240	32.13	0.395	59.37
gzip	555	249	44.86	0.437	57.54
hello	139	28	20.14	0.389	58.33
libidn	595	85	14.29	0.306	56.76
libmicrohttpd	710	93	13.10	0.299	52.41
libtasn1	498	91	18.27	0.317	52.90
libtool	454	30	6.61	0.360	43.05
libunistring	2,061	40	1.94	0.338	51.61
lightning	1,377	334	24.26	0.408	58.73
macchanger	145	9	6.21	0.471	46.76
nettle	234	54	23.08	0.422	56.49
osip	3,681	33	0.90	0.214	24.15
patch	1,422	367	25.81	0.315	57.42
sed	696	276	39.66	0.385	55.70
sharutils	1,534	468	30.51	0.326	53.82
spell	69	14	20.29	0.315	46.11
tar	3,156	1,473	46.67	0.416	57.26
texinfo	197	43	21.83	0.290	55.50
time	45	2	4.44	0.400	50.00
units	492	151	30.69	0.482	53.02
wdiff	110	29	26.36	0.469	48.95
which	153	47	30.72	0.502	57.97
xorriso	11,772	2,219	18.85	0.278	56.14

The five-hop view excludes 107 pairs whose required descendants lie beyond five hops. Tables 7 and 8 give the eligibility chain and coverage-weighted results. Weighting by C_{shared} emphasizes pairs with more shared context.

Tables 9, 10, and 11 separate results by task, compiler, and project. These checks address variation within the observed sample.

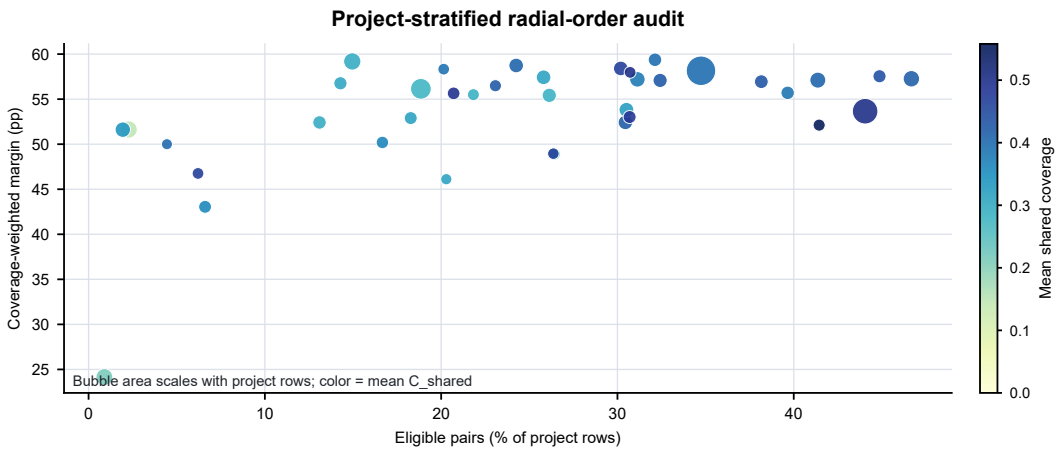

Figure 3: Project-stratified radial-order audit. Each point is one project. The horizontal coordinate is the eligible-pair rate, the vertical coordinate is the coverage-weighted radial-order margin, bubble area scales with the number of project rows, and color denotes mean shared coverage C_{shared}. The exact project values remain in Table 11.

B.3 Ancestor-Relation Analysis and Retrieval Gains

The training-side pairwise ancestor-relation analysis has an observed same-direction rate of 98.99%, compared with 29.95% for the corresponding random-control baseline. The project-clustered 95% interval for the margin is [+63.74,+76.40] percentage points. This analysis covers shared source-role pairs in the training-side sample. The held-out test set contains no cross-binary source mappings, so preservation labels are unavailable for its 4,000 query-positive pairs.

We separately compare +APRM with Radial-State Only on those test pairs. Radial-State Only uses the matched extra training budget on the radial-state branch and omits the predecessor-contrast branch. Pairing outcomes across four encoders and three seeds gives 48,000 observations. Table 12 reports project-clustered intervals for the taskwise Recall@1 differences. These intervals concern retrieval outcomes, not preservation of ancestor relations on the test set. They pool results across all four encoders.

Table 12: +APRM minus Radial-State Only on the held-out test set. Differences and project-clustered bootstrap 95% intervals are in percentage points. Results pool the 12 matched encoder–seed cells.

Task	ΔR@1 (pp)	95% CI (pp)
XC	+1.54	[+1.31,+2.11]
XA	+0.49	[+0.26,+0.69]
XO	+0.50	[+0.07,+1.07]
XM	+0.52	[-0.46,+0.90]

C Additional Experimental Results

These experiments supplement the retrieval, uncertainty, component, baseline, and sensitivity analyses in the main text.

C.1 Retrieval uncertainty and secondary metrics

We pair methods by query within each task. Query bootstrap intervals use 10,000 resamples within tasks, followed by an equal-weight task average. They condition on the fixed projects and galleries. The project- and source-cluster intervals in Table 13 use the corresponding clusters and seeds.

The four-encoder mean and SD use the three encoder-averaged seed scores. Combined analyses retain the 100 shared XC–XM query-positive pairs.

The sign-flip test uses one source-class cluster within a task and seed as its unit for each encoder. Projects, galleries, and seeds remain fixed blocks. Holm correction covers the four primary encoders. The unadjusted p value is 0.0001 for each encoder; Table 13 reports the adjusted values. Table 14 gives Recall@10.

Table 13: Seed- and cluster-aware uncertainty for the primary held-out benchmark for the four-encoder comparison. Rows without the +APRM suffix are Base Encoder results; rows with the suffix are +APRM results. Values are percentages except d_z and Holm p; intervals are percentile 95 percent intervals from 10,000 seed-and-cluster bootstrap replicates. Holm-adjusted p values use the four-encoder source-cluster sign-flip family defined above.

Models	Macro Recall@1	Δ (pp)	Project 95 percent CI	Source 95 percent CI	d_z	Holm p
Gemini	8.47	–	–	–	–	–
Gemini+APRM	20.75	+12.28	[10.12,14.84]	[11.53,13.05]	0.31	0.0004
TREX	7.63	–	–	–	–	–
TREX+APRM	17.77	+10.14	[9.31,11.03]	[9.43,10.87]	0.28	0.0004
DiEmph	2.53	–	–	–	–	–
DiEmph+APRM	10.55	+8.02	[5.63,11.15]	[7.09,8.92]	0.28	0.0004
CodeART	7.27	–	–	–	–	–
CodeART+APRM	9.93	+2.66	[2.16,3.39]	[2.22,3.11]	0.11	0.0004

Table 14: Secondary held-out-test Recall@10 summary. Rows without the +APRM suffix are Base Encoder results; rows with the suffix are +APRM results. Base vectors are deterministic and +APRM values are means across seeds 42–44.

Models	Macro Recall@10 (%)
Gemini	22.06
Gemini+APRM	47.86
TREX	20.85
TREX+APRM	40.20
DiEmph	8.07
DiEmph+APRM	26.61
CodeART	12.68
CodeART+APRM	20.86
Four-encoder mean	15.92
Four-encoder mean+APRM	33.88

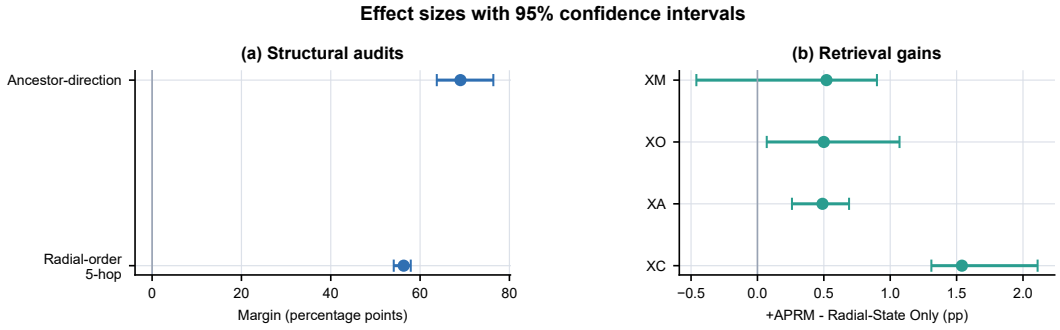

Figure 4: Effect sizes with 95 percent confidence intervals. Panel (a) summarizes the deployed five-hop radial-order margin and the ancestor-direction margin. Panel (b) shows +APRM minus Radial-State Only for each held-out task. Intervals are project-clustered bootstrap 95% confidence intervals; the exact values and definitions are given in Tables 8 and 12.

C.2 Mechanism controls

The Magnitude-Only Contrast and All-Incoming Edges variants use the same selected radial-state component as APRM for each encoder and seed. They train the contrast branch for 14 FP32 epochs and select a checkpoint on validation data. The APRM row uses the same primary held-out evaluation. Radial-State Only omits the contrast branch and continues the radial-state branch for the matched extra budget.

Table 15 gives taskwise Recall@1 for the TREX mechanism controls. No four-encoder average is used in this table or Table 3. The root-readout controls use the interventions named in the main text.

Table 15: TREX mechanism controls (percent). Macro Recall@1 reports the mean and sample SD over seeds 42–44; taskwise entries are seed means. Radial-State Only uses matched-budget continuation.

Variant	XC	XA	XO	XM	Macro R@1
APRM	23.10	10.77	28.13	9.07	17.77 ± 0.38
Without Root-Conditioned Gates	21.77	10.37	29.00	8.80	17.48 ± 0.19
Without Direct Root Addition	21.37	8.90	27.14	8.23	16.41 ± 0.23
Radial-State Only	20.27	10.03	27.07	7.57	16.23 ± 0.42
Collapsed Radial Position	17.63	8.77	23.47	6.60	14.12 ± 0.17
Uniform Hop Coordinate	22.10	10.00	28.07	8.70	17.22 ± 0.43
Magnitude-Only Contrast	20.17	9.43	27.50	7.73	16.21 ± 0.42
All-Incoming Edges	21.67	10.27	28.33	8.73	17.25 ± 0.27
Static-Context Control	16.67	8.87	21.37	6.20	13.28 ± 0.18

Coordinate comparison. The coordinate comparison uses the CDF midpoint in +APRM as its reference. Uniform Hop Coordinate replaces the coordinate with $h_r(v)/K$ for $K = 5$. It reaches 17.22 ± 0.43 Macro Recall@1, with task means of 22.10, 10.00, 28.07, and 8.70 for XC, XA, XO, and XM. The comparison uses the fixed-encoder, fixed-radial-state APRM protocol with TREX, seeds 42–44, the selected radial-state parent for each seed, and 14 FP32 contrast-training epochs. Checkpoints are selected by validation Macro Recall@1, then Recall@10 and the earlier epoch for exact ties. Fixed queries, galleries, expected-tie scoring, and output dimension match the completed controls. Relative to +APRM, the coordinate gives -0.55, -0.61, and -0.43 percentage points for Macro Recall@1, Macro Recall@10, and Macro MRR@10, respectively.

The two root-readout controls are reported separately. Without Root-Conditioned Gates replaces the root-conditioned predecessor gate with one learned scalar sigmoid gate per Bernstein basis; the radial-state branch and its context gate remain frozen. Without Direct Root Addition removes only the direct r_r addition in the readout. The root state remains available to the internal gates and contrast computation, so this is not removal of all root information. Their formal validation runs, held-out test evaluations, selected checkpoints, and evaluation inputs were verified before inclusion.

For the TREX panel, the Base Encoder–Collapsed Radial Position difference is 6.49 percentage points, approximately 64% of the full Base Encoder–+APRM difference.

C.3 Matched message-passing baselines

The four matched message-passing baselines (GCN, GAT, gated-MPNN, and RGCN) use the same node and edge inputs, five-hop cap, output dimension, objective, seeds, and 14-epoch schedule. The FCG edge list is caller-to-callee. GCN, GAT, and gated-MPNN append deduplicated reverse edges before message passing. RGCN keeps the original caller-to-callee edges as relation 0 and adds explicit reverse edges as relation 1. Each model selects one checkpoint on the held-out validation split. The held-out test set contains 21,812 functions and uses the query count and function pool size defined in Section 4.1.

Each baseline is an independent model with its own graph operator and root-only readout. It does not load APRM parameters or its contrast branch. The feature encoder, output normalization, loader, and loss interface are shared. Parameter matching uses the APRM reference for the same encoder and seed.

Table 16 aggregates the four encoders and three seeds. APRM scores are given in Tables 1 and 14.

Table 16: Held-out-test matched message-passing results for Gemini, TREX, DiEmph, and CodeART and seeds 42–44. Values are means in percent; Δ is relative to APRM. The APRM reference row is listed first.

Control	Macro Recall@1	Δ vs APRM (pp)	XC	XA	XO	XM	Macro Recall@10
APRM	14.75	0.00	14.26	15.70	21.22	7.83	33.88
GCN	11.57	-3.18	11.43	12.61	16.56	5.67	27.28
GAT	10.92	-3.83	10.26	12.51	15.54	5.36	26.60
Gated-MPNN	12.75	-2.00	12.22	14.22	18.02	6.54	29.73
RGCN	10.38	-4.37	9.91	12.02	14.48	5.12	24.59

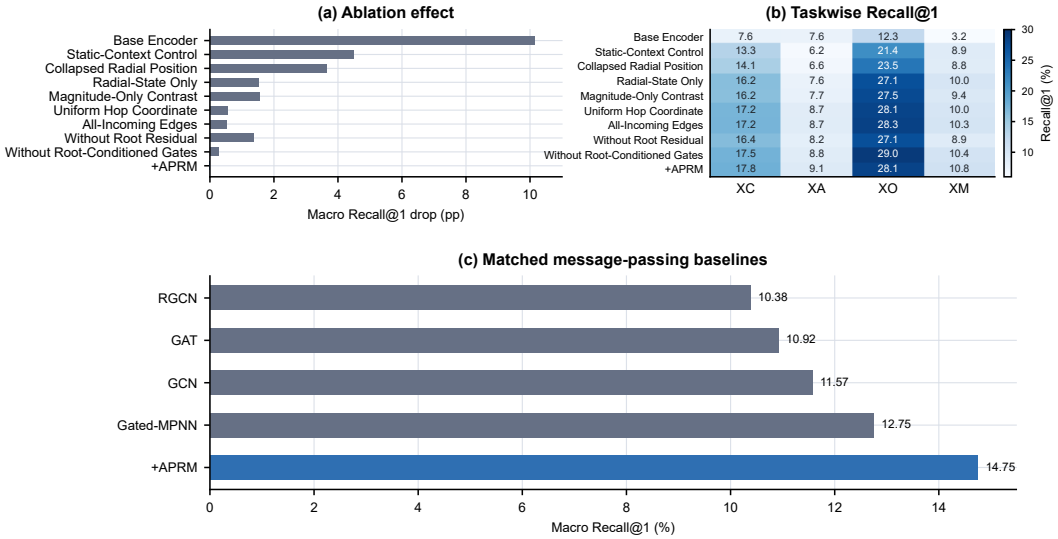

Figure 5: Mechanism controls and matched message-passing baselines. Panel (a) shows Macro Recall@1 drops relative to +APRM. Panel (b) shows the taskwise seed means from Table 15 in the standard task order XC, XA, XO, XM. Panel (c) shows the four-encoder mean for the matched graph baselines and APRM. Seed standard deviations are retained in the exact ablation table rather than mixed into the heatmap.

C.4 FVA component factorial

The TREX FVA factorial gives 17.13% without explicit FVA inputs. Its four conditions use the fixed radial-state protocol but a separate checkpoint family, not the primary APRM weights. The CDF radial-state component, topology operator, five-hop budget, optimizer, validation selector, and test package are fixed across conditions. Only the root-to-node and predecessor-edge FVA inputs change. Each condition uses three seeds.

Table 17 includes Recall@10 and MRR@10 in addition to Recall@1. The reference is its own Both-FVA score of 17.54%, not the primary 17.77% score.

Table 17: Held-out-test FVA mechanism control on TREX. Values are means over seeds 42–44 (percent). "Decrease from both" is the positive R@1 difference between the condition with both FVA channels and the listed condition.

Condition	R@1 percent	Decrease (pp)	R@10 percent	MRR@10 percent
Both FVA channels	17.54	0.00	40.21	24.01
No explicit FVA	17.13	0.41	39.19	23.54
Node FVA only	17.18	0.36	39.21	23.55
Edge FVA only	16.95	0.59	39.42	23.46

C.5 Function Pool Size

Figure 6 and Tables 18 and 23 give the trend, secondary metrics, and encoder-level results for the experiment varying function pool size in this appendix. Each nested function pool retains all designated positives. Only the number of additional negatives changes; checkpoints remain fixed. The same frozen checkpoints are evaluated with nested function pools. Recall@1 decreases as the function pool grows, but +APRM remains above both Base Encoder and Radial-State Only at every size. At a function pool size of 1,024, the three scores are 13.82%, 26.52%, and 28.41%; at 10,000 functions, they are 6.48%, 13.93%, and 14.75%, respectively.

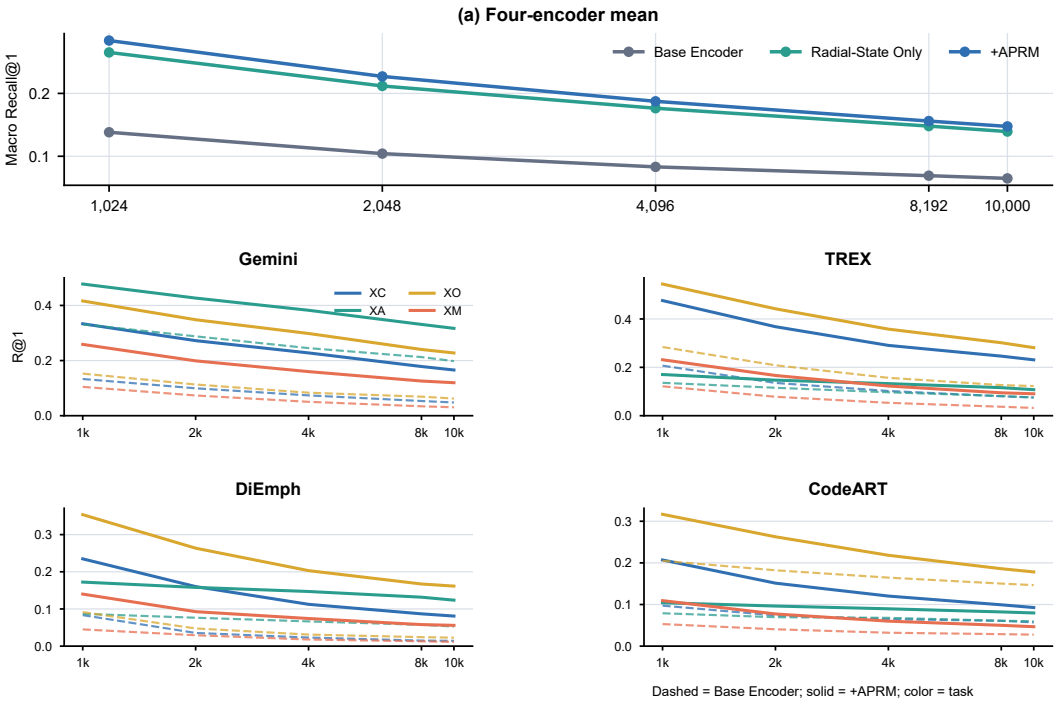

Figure 6: Sensitivity to function pool size across encoders. Panel (a) shows the four-encoder mean for Base Encoder, Radial-State Only, and +APRM. The lower panels show taskwise encoder-level Recall@1; dashed lines are Base Encoder and solid lines are +APRM. Function pools are nested, all designated positives are retained, and checkpoints are not retrained or reselected.

Table 18: Held-out-test sensitivity to function pool size. Every scale retains all 1,000 designated positives and adds negatives in a fixed order. No checkpoint is retrained or reselected. Entries are percentages; each function-pool-size group reports Recall@1 and Recall@10.

Models	1,024		2,048		4,096		10,000	
	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10
Base	13.82	31.56	10.42	24.92	8.31	20.48	6.48	15.91
Base + Radial-State Only	26.52	51.98	21.18	45.20	17.64	38.87	13.93	31.92
Base + APRM	28.41	55.29	22.70	48.43	18.74	41.76	14.75	33.88

Table 19: NOVA held-out-test retrieval comparison at the fixed 8,192-function pool used for the main-text headline. Values are percentages; the NOVA+APRM row reports the mean ± sample standard deviation across the three seeds. The final column is the equal-weight mean MRR over XC and XO.

Models	XC		XO		Mean MRR
	R@1	R@10	R@1	R@10	
NOVA	1.63	6.41	1.83	7.72	3.03
NOVA+APRM	5.32 ± 0.25	21.58 ± 0.51	6.34 ± 0.41	23.90 ± 1.88	10.18 ± 0.42

C.6 Hop-budget selection and sensitivity

Before the primary APRM experiments, we selected K on the radial-state branch using validation data. The development grid used seed 42 and Gemini, DiEmph, and CodeART, with the same 14-epoch schedule. Table 20 records this decision.

Table 20: Validation-only radial-state hop-budget grid. Entries are Macro Recall@1 percentages at seed 42 after the same 14-epoch schedule; Mean is the equal-weight mean over the three encoders. The held-out test set was not accessed. Bold denotes the selected hop budget.

K	Gemini	DiEmph	CodeART	Mean
1	10.825	7.950	14.900	11.225
2	11.650	9.525	15.850	12.342
3	11.525	9.850	15.775	12.383
4	12.350	10.000	15.975	12.775
5	12.900	11.400	16.575	13.625
6	12.575	10.825	16.550	13.317
7	12.900	10.950	16.450	13.433

Additional descriptive test results. Figure 7 and Tables 21 and 22 describe a later, independent scan. Each checkpoint was selected on validation data before testing. These results do not change the primary $K = 5$ choice. They use different training and validation runs, not a controlled replication of the primary result, and are not pooled with it.

Table 21: Validation-only APRM hop-budget scan with descriptive test results. Each row aggregates 12 cells from one hop budget across four encoders and seeds 42–44. Entries are percentages with sample SD across these cells ($n = 12$), not seed-only uncertainty. The test columns are included only as a descriptive analysis; they are not used for model, checkpoint, or protocol decisions. This scan is an independent training and validation run and does not replace the primary $K = 5$ result.

K	Validation Macro Recall@1	Test Macro Recall@1	Test Macro Recall@10	Test Macro MRR@10	Δ Test Recall@1 vs. $K = 5$
1	14.35 ± 1.99	11.62 ± 4.48	28.07 ± 10.53	16.26 ± 6.17	-1.92 pp
2	15.87 ± 1.68	13.73 ± 4.82	32.31 ± 10.49	19.01 ± 6.44	$+0.19$ pp
3	15.94 ± 1.47	13.87 ± 4.92	33.38 ± 10.80	19.40 ± 6.61	$+0.32$ pp
4	16.05 ± 1.37	13.31 ± 4.71	32.37 ± 10.54	18.71 ± 6.35	-0.23 pp
5	15.98 ± 1.27	13.54 ± 4.83	32.61 ± 10.81	18.93 ± 6.56	0.00 pp
6	16.11 ± 1.16	13.70 ± 4.68	32.89 ± 10.77	19.13 ± 6.42	$+0.16$ pp
7	16.18 ± 1.06	13.20 ± 4.53	32.37 ± 10.32	18.65 ± 6.19	-0.34 pp
8	16.20 ± 0.93	13.51 ± 4.82	32.79 ± 10.33	18.97 ± 6.43	-0.03 pp
9	16.25 ± 0.96	13.71 ± 4.70	33.43 ± 11.10	19.35 ± 6.56	$+0.16$ pp

Table 22: Per-encoder results from the additional descriptive analysis. R@1 values are percentages, with sample SD over three seeds. The test ranking is descriptive and is not used for model selection. This table recomputes scores using expected exact ties; it is separate from the original scan in Table 21.

Encoder	Val. best K	Descriptive test-max K	Test-max R@1	K=5 R@1	Test order
Gemini	9	9	21.89 ± 0.69	20.85 ± 0.28	9, 2, 3, 6, 8, 4, 5, 7, 1
TREX	2	3	18.18 ± 0.39	17.51 ± 0.33	3, 2, 5, 8, 4, 6, 7, 9, 1
DiEmph	9	9	10.99 ± 0.80	10.38 ± 0.83	9, 2, 6, 3, 8, 7, 4, 5, 1
CodeART	7	2	9.79 ± 0.34	9.58 ± 0.20	2, 3, 4, 9, 5, 6, 7, 1, 8

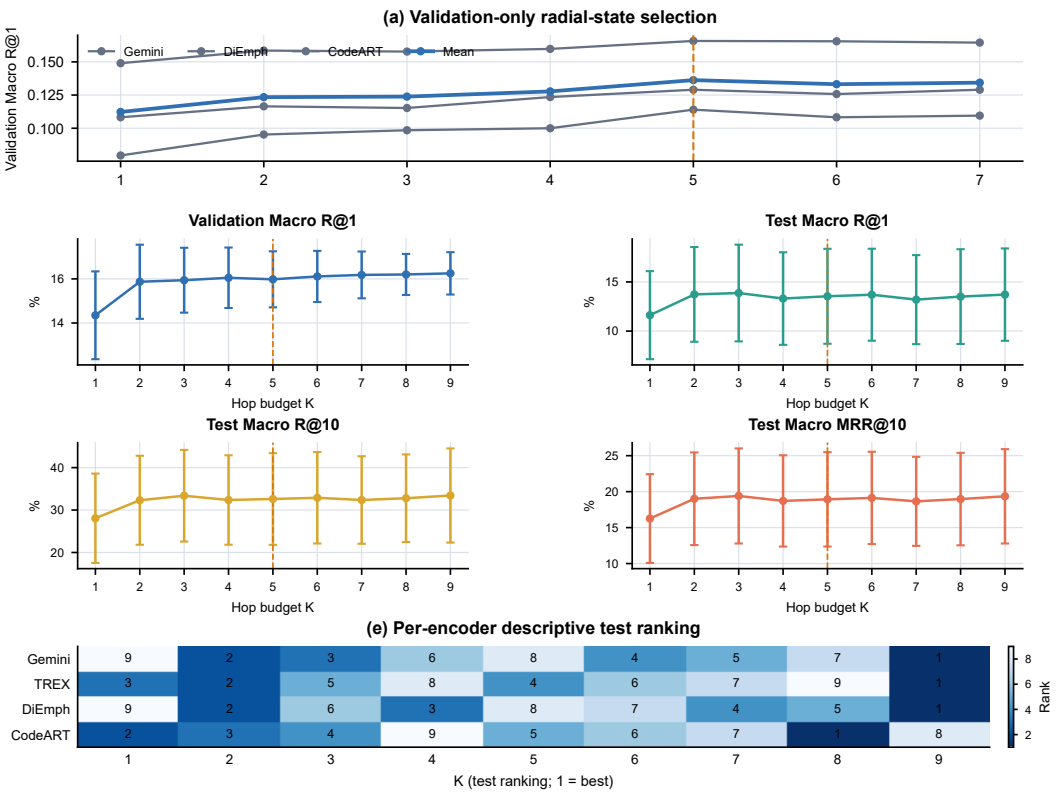

Figure 7: Hop-budget sensitivity and descriptive ranking. The top panel shows the validation-only radial-state selection grid and marks the selected $K = 5$. The four middle panels show the independent scan with mean and sample SD over its 12 encoder–seed cells. The bottom panel displays the descriptive per-encoder test ranking; these test results are not used for model or checkpoint selection.

D External Encoder and Enhancer Comparisons

These two comparisons supplement the transfer and external-comparison results in the main text and remain separate from the four-encoder primary benchmark.

D.1 NOVA x86-64 XC/XO extension

We use the public lt-asset/nova-1.3b foundation checkpoint as the fixed upstream encoder, rather than a BinaryCorp similarity-fine-tuned checkpoint (Jiang et al., 2025). The fixed vector has 2,048 dimensions; the availability bit adds one input. A dimension-matched radial-state component is loaded before contrast training. The optimizer and fixed-parameter protocol follow Appendix A.6.

Validation and testing each use 1,000 queries per task and a shared 8,192-function gallery within each task. XC and XO galleries are separate. The test union contains 17,364 functions. We retain valid functions from the earlier projections, then fill each gallery deterministically under the same structural filter.

The two-task validation selector follows the primary metric order and evaluates all 14 epochs with float64 cosine scores and expected exact ties. Selected epochs are 7, 1, and 7 for seeds

Table 23: Per-encoder Recall@1 across function pool sizes. Rows identify the encoder and APRM setting; columns are grouped by function pool size, with XC, XA, XO, and XM subcolumns. Entries are percentages. +APRM values are means over seeds 42–44, whereas base vectors are deterministic. The table is split into two blocks to preserve the template body font size.

Models / Variant	1,024				2,048			
	XC	XA	XO	XM	XC	XA	XO	XM
Gemini	13.32	33.13	15.27	10.46	9.94	28.79	11.32	7.34
Gemini+APRM	33.35	47.77	41.63	25.87	27.23	42.67	34.82	19.90
TREX	20.70	13.60	28.40	12.15	13.55	11.55	20.85	7.83
TREX+APRM	47.60	17.03	54.43	23.13	36.80	14.70	44.20	16.63
DiEmph	8.34	8.69	9.15	4.50	3.55	7.64	4.76	2.93
DiEmph+APRM	23.48	17.23	35.37	14.00	16.02	15.80	26.35	9.27
CodeART	9.73	7.90	20.50	5.30	7.38	6.95	18.22	4.05
CodeART+APRM	20.70	10.43	31.67	10.93	15.13	9.63	26.27	7.73

Models / Variant	4,096				8,192			
	XC	XA	XO	XM	XC	XA	XO	XM
Gemini	7.37	24.52	8.34	5.04	5.33	21.27	6.91	3.42
Gemini+APRM	22.78	38.25	29.83	16.00	17.85	33.12	24.03	12.58
TREX	10.23	9.68	15.67	5.33	8.03	8.15	12.65	3.71
TREX+APRM	29.07	13.27	35.80	12.27	24.63	11.60	30.17	9.47
DiEmph	2.31	6.65	3.11	1.78	1.49	5.74	2.42	1.30
DiEmph+APRM	11.25	14.70	20.31	7.43	8.69	13.17	16.72	5.80
CodeART	6.49	6.80	16.46	3.22	6.08	6.10	15.03	2.88
CodeART+APRM	12.03	8.97	21.83	6.00	9.93	8.22	18.60	5.00

Table 24: NOVA sensitivity to function pool size on the complete XC/XO test set. The table reports XC and XO Recall@1 at each function pool size; the final column is the equal-weight mean MRR over the eight XC/XO–pool-size cells. Values are percentages; +APRM Recall@1 entries report the three-seed mean ± sample standard deviation. The table is split into two blocks to preserve the template body font size.

Models	1,024		2,048	
	XC	XO	XC	XO
NOVA	7.49	9.03	3.50	4.39
NOVA+APRM	17.89±0.99	20.63±2.38	12.23±0.48	13.56±0.74

Models	4,096		8,192		Mean MRR
	XC	XO	XC	XO	
NOVA	2.44	2.86	1.63	1.83	6.94
NOVA+APRM	7.98±0.47	9.04±0.10	5.32±0.25	6.34±0.41	18.68

42–44. Table 19 reports the held-out-test retrieval comparison. Table 24 varies function pool size without checkpoint reselection.

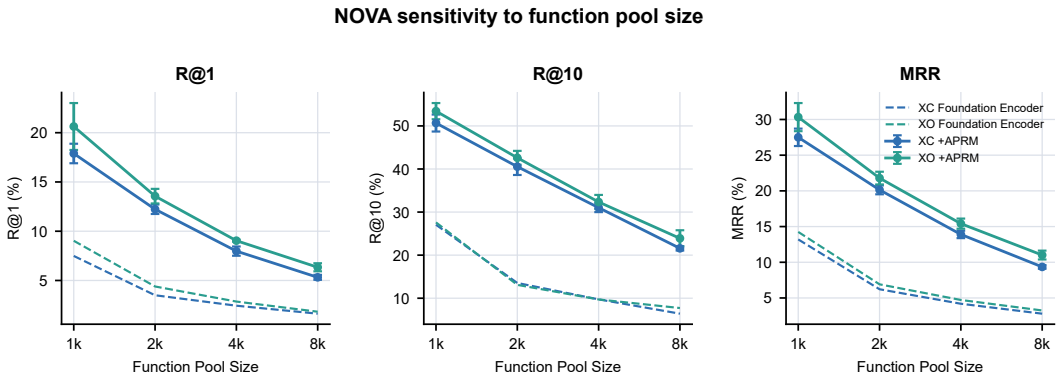

Figure 8: NOVA sensitivity to function pool size. Dashed lines show the deterministic foundation encoder and solid lines show the three-seed +APRM mean with sample-SD error bars. XC and XO are shown separately for R@1, R@10, and MRR.

D.2 BinEnhance-Pro comparison

Table 25 uses the common pool defined for the BinEnhance-Pro comparison in the main text. BinEnhance-Pro is reported with its FCG-only and Full-EESG representations, using the corresponding frozen checkpoints selected on validation data. The static rows use full-gallery cosine ranking, whereas the Full-EESG reranking row and the APRM reranking row use the matched two-stage contract below. The rerankers share initialization, training pairs, calibrator seed 1234, and 10 negatives per positive from 500 hard-negative candidates. They use InfoNCE temperature 0.07, Adam with learning rate 0.005, weight decay 5×10^{-4}, at most 20 epochs, and validation-loss patience 3. Static retrieval uses float64 cosine scores. Pairwise features use float32 followed by tanh. Both protocols use expected exact ties.

The APRM representation uses Gemini seed 44 with the same matched training and checkpoint-selection protocol as the primary evaluation.

Table 25: Fair APRM–BinEnhance-Pro comparison on the exact common test pool. Entries are four-task macro percentages over XC/XA/XO/XM, with 169 queries and 2,424 gallery functions per task. Static rows use full-gallery cosine ranking. Pairwise rows use Annoy top-500 retrieval with 10 trees and separate linear rerankers trained under the same pair schedule and optimization settings. ANN misses count as failures.

Models	Retrieval contract	R@1	R@10	MRR@10	Candidate recall
BinEnhance-Pro (FCG-only)	Full-gallery cosine	13.61	37.72	20.25	–
BinEnhance-Pro (Full-EESG)	Full-gallery cosine	16.72	33.73	21.47	–
BinEnhance-Pro (Full-EESG) + pairwise reranking	ANN top-500 + pairwise	42.90	59.91	48.93	80.92
APRM	Full-gallery cosine	25.15	54.73	33.83	–
APRM + pairwise reranking	ANN top-500 + pairwise	48.08	72.63	55.78	94.08

E Dataset-1 Cross-Dataset Zero-Shot Transfer

E.1 Protocol and checkpoint selection

We evaluate the released Dataset-1 benchmark (Marcelli et al., 2022) under the pair-scoring protocol specified by its release. "Zero-shot" here means no training, checkpoint selection, or adaptation on Dataset-1. All 12 checkpoints use the primary APRM method, with a fixed radial-state component and a trainable predecessor-contrast branch. We use validation-only epoch selection and no Dataset-1 use for training or selection.

The source validation split determines checkpoint selection. Six of the 12 checkpoints match the primary benchmark; the other six are selected on that split. The method is unchanged, but the transfer does not reuse all primary-table weights. Table 26 summarizes the encoder-level Macro Recall@1 and Macro MRR comparison.

Each task contains 1,000 official positive queries. XC, XA, and XO provide 8,192 negatives per query; XM provides 10,000. We use the released pairs without resampling. At pool size N, scoring uses the positive and first $N-1$ negatives with rank_method=max. Its optimistic exact ties differ from the primary benchmark's expected-tie rule.

All 919 binaries and 522,003 functions remain in scope. The models use the same 4,000 positive and 34,576,000 negative pair rows. APRM consumes the complete five-hop context: every official root and every required FCG node must have an exact canonical vector. Missing functions are not replaced with zeros, cropped, or silently excluded. The reproducibility record verifies this coverage.

E.2 Aggregate and project-level results

Tables 26 and 27 report aggregate and project-stratified results. The project table separates nmap and z3 rather than treating their query counts as balanced. It supports descriptive comparisons, not a causal explanation of project-specific gains.

Table 26: Dataset-1 cross-dataset transfer results without target adaptation, reported in percent. Rows without the +APRM suffix are Base Encoder results; rows with the suffix are +APRM results. Macro MRR is the equal-weight mean over XC, XA, XO, and XM; the final row gives the four-encoder mean.

Models	Macro Recall@1	Macro MRR
Gemini	19.475	24.328
Gemini+APRM	23.350 ± 0.416	28.732
TREX	9.850	13.021
TREX+APRM	15.733 ± 0.364	19.957
DiEmph	12.975	15.153
DiEmph+APRM	14.692 ± 0.088	17.602
CodeART	9.925	11.567
CodeART+APRM	11.392 ± 0.431	13.879
Four-encoder mean	13.056	16.017
Four-encoder mean+APRM	16.292 ± 0.072	20.043

Table 27: Dataset-1 cross-dataset project-stratified Recall@1 comparison without target adaptation. Each cell gives the baseline percentage, followed by the three-seed APRM mean ± sample standard deviation (both in percent) and the absolute change in percentage points. The query counts for nmap/z3 are 103/897 for XC, 101/899 for XA, 90/910 for XO, and 103/897 for XM.

Project	Encoder	XC	XA	XO	XM
nmap	Gemini	34.95 → 50.16 ± 2.02 (+15.21)	38.61 → 45.87 ± 3.48 (+7.26)	24.44 → 44.81 ± 3.57 (+20.37)	19.42 → 30.42 ± 2.02 (+11.00)
nmap	TREX	25.24 → 46.28 ± 1.48 (+21.04)	8.91 → 17.49 ± 1.14 (+8.58)	30.00 → 44.07 ± 4.21 (+14.07)	10.68 → 22.01 ± 1.12 (+11.33)
nmap	DiEmph	32.04 → 40.45 ± 2.02 (+8.41)	1.98 → 7.26 ± 2.29 (+5.28)	23.33 → 37.04 ± 2.31 (+13.70)	7.77 → 12.30 ± 0.56 (+4.53)
nmap	CodeART	22.33 → 28.80 ± 4.04 (+6.47)	0.99 → 3.96 ± 0.99 (+2.97)	14.44 → 29.63 ± 1.28 (+15.19)	5.83 → 11.97 ± 1.48 (+6.15)
z3	Gemini	19.51 → 22.67 ± 0.28 (+3.16)	23.92 → 27.44 ± 1.84 (+3.52)	20.66 → 23.74 ± 0.44 (+3.08)	9.36 → 10.96 ± 0.32 (+1.60)
z3	TREX	13.71 → 21.37 ± 0.84 (+7.66)	3.45 → 7.04 ± 0.78 (+3.60)	13.85 → 20.29 ± 0.84 (+6.45)	4.57 → 6.91 ± 0.11 (+2.34)
z3	DiEmph	19.96 → 21.11 ± 0.45 (+1.15)	2.34 → 3.67 ± 0.40 (+1.33)	23.30 → 24.25 ± 0.44 (+0.95)	4.79 → 5.54 ± 0.28 (+0.74)
z3	CodeART	17.84 → 19.25 ± 0.76 (+1.41)	0.78 → 1.45 ± 0.11 (+0.67)	16.70 → 17.36 ± 0.50 (+0.66)	3.90 → 4.38 ± 0.45 (+0.48)

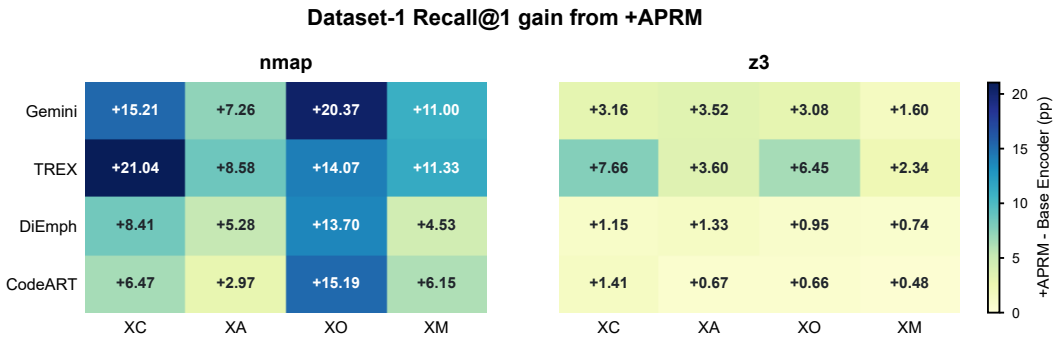

Figure 9: Dataset-1 project- and task-stratified gain. Cells report the Recall@1 gain from +APRM over the Base Encoder in percentage points. The exact base scores, +APRM means, standard deviations, and taskwise values remain in Tables 26 and 27.

F Inference Cost

This section supplements the efficiency note in the main text. We measure enhancement on the 22,600-function validation set using seed 42, batch size 64, and two full-encoding repetitions. "Pure forward" excludes graph reconstruction; "rebuild+forward" includes graph loading and feature assembly. Cached enhanced vectors are not counted as model inference. Table 28 separates these costs.

Table 28: Measured APRM enhancement inference on the complete 22,600-function validation set. Times are means; batch size is 64; all rows use seed 42.

Encoder	Input dim.	Pure forward (ms)	Rebuild+forward (ms)	Full encode (s)
Gemini	65	130.43	426.62	110.31
TREX	769	121.68	963.54	209.74
DiEmph	769	70.77	990.99	194.78
CodeART	769	49.33	749.83	224.61

Table 29 measures query enhancement and search for Gemini seed 42. The p95 times include graph-loading and shared-service variation. The scoring path differs from the primary held-out-test path. These measurements exclude upstream pretraining, APRM training, other seeds, and non-Gemini online queries. They are not a matched end-to-end comparison.

Table 29: Gemini seed-42 measured online inference profile. Each task uses a 10K gallery and 1,000 fixed queries; enhancement is recomputed per query.

Task	Index build (s)	Index MB	Enh. p50/p95 (ms)	ANN p50/p95 (ms)	Online (s)
XC	0.887	20.75	44.34/413.52	0.233/0.311	110.678
XA	3.633	20.80	33.62/94.39	0.232/0.316	49.508
XO	3.423	20.82	9.79/537.10	0.223/0.277	148.996
XM	4.421	20.79	38.51/106.86	0.222/0.272	57.558

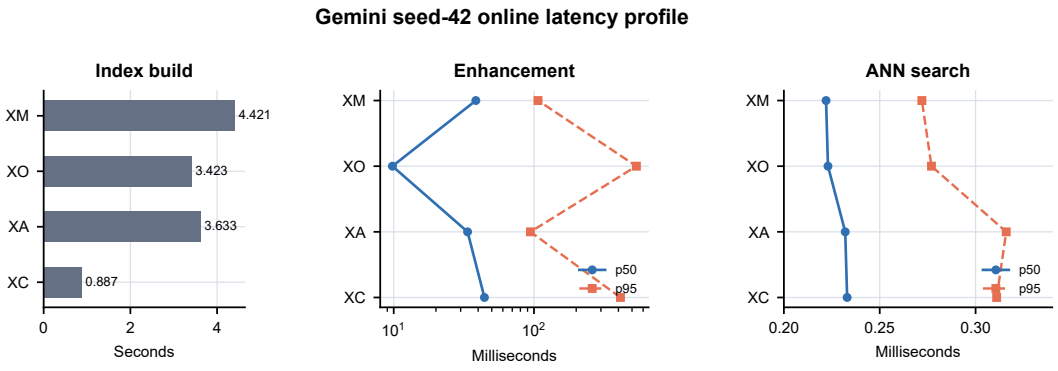

Figure 10: Gemini seed-42 online latency profile. The panels separate index-build time, enhancement p50/p95, and ANN-search p50/p95. Exact online totals remain in the accompanying table; p50 and p95 are not pooled with retrieval uncertainty.