RevDecode: Enhancing Binary Function Matching with Context-Aware Graph Representations and Relevance Decoding

USENIX Security 2025

Tongwei Ren, Ronghan Che, Guin R. Gilman, Lorenzo De Carli, Robert J. Walls
Worcester Polytechnic Institute, University of Calgary

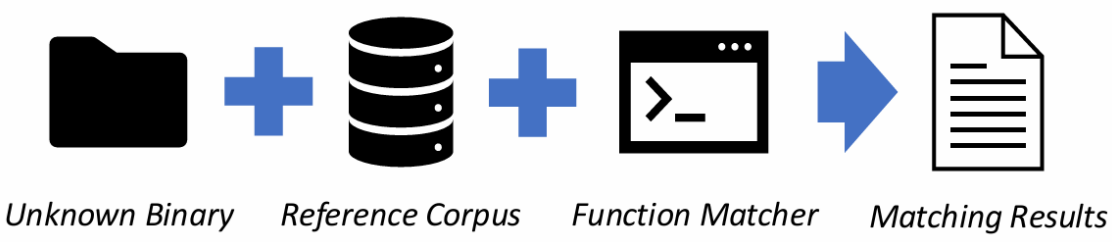
Motivation

What is function matching?

- Match functions from a stripped binary to a corpus of known functions

Why it matters

- Identifying libraries in embedded firmware
- Isolating vulnerable functions
- Understanding code reuse in proprietary codebases



Why it is hard

- Compilation settings, optimization levels, and version differences distort the same function

Existing Approaches & Limitations

Existing Approaches

- BSim, discovRE, SAFE, Gemini compare via control flow, gadgets, instruction/function embeddings, or signatures
- Each reduces a function to an abstract representation, then a similarity score
- But high similarity ≠ a useful match for a reverse engineer

Limitations

Similarity is not the same as relevance

- Matchers focus on similarity; reverse engineers need meaningful insight

Example (discovRE)

- custom_exts_copy (libssl 3.5.0): discovRE's top pick is unrelated elf32_arm_size_stubs (libbfd 2.30)
- Relevant match (libssl 3.0.2) buried at rank 197

Other challenges

- Incomplete corpora; function versions easily confused

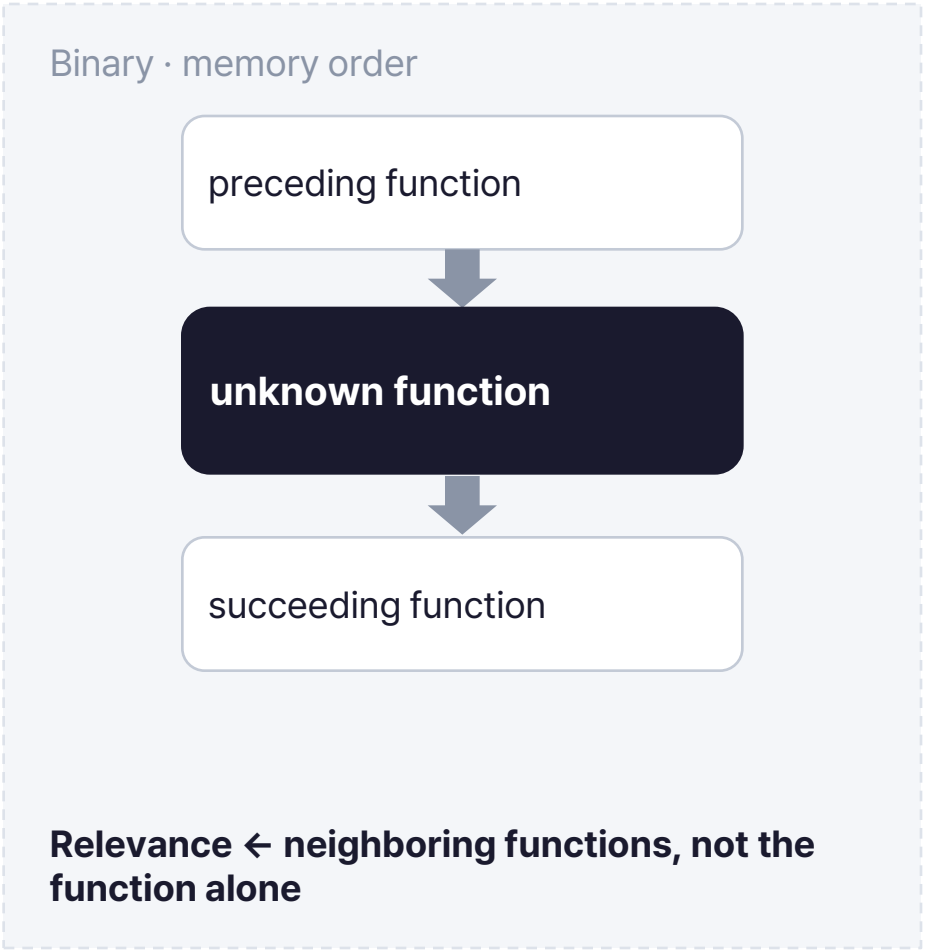
Key Idea: Relevance Decoding

Relevance decoding from context

- Judge relevance from surrounding context, not a function in isolation
- Context = neighboring functions and their positions in the binary
- Existing matchers overlook surrounding code — preceding and succeeding functions

What RevDecode does

- Captures contextual signals with a graph to improve accuracy
- Not a new matcher — a framework enhancing any underlying matcher
- Bridges the gap between similarity and relevance



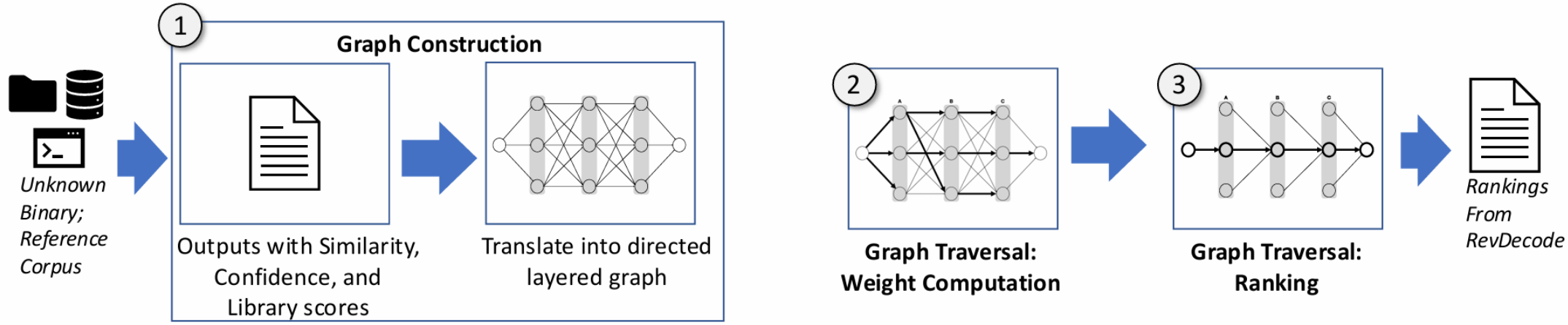
RevDecode Overview

Input and output

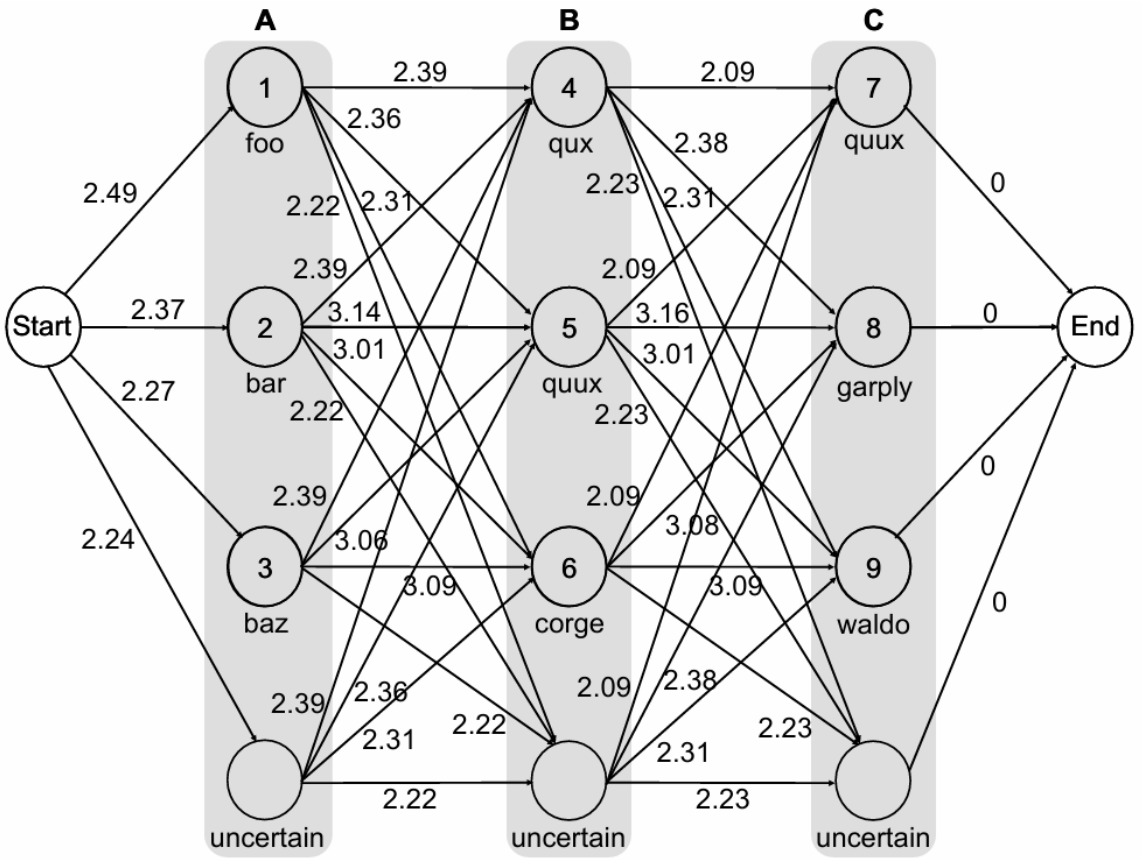
- Unknown binary + reference corpus
- Ranked matches per unknown function

Three phases

- **Graph construction** turns matcher output into a directed layered graph.
- **Graph traversal** runs a Viterbi style forward pass over the graph.
- **Ranking** runs a backward pass to reorder the matches.



How to Read the Graph



(a) Constructed directed acyclic graph.

Element	Meaning
A, B, C	Three unknown functions in the analyzed binary
foo, bar, baz, …	Candidate functions from the corpus
1 – 9	Index numbers on candidate nodes
Gray vertical column	Candidate list for one unknown function (a layer)
Edges (lines)	Transitions linking candidate combinations
Number on an edge	Weight (score) of that candidate pairing
uncertain	No suitable candidate may exist in the corpus
Start, End	Special virtual nodes for path traversal

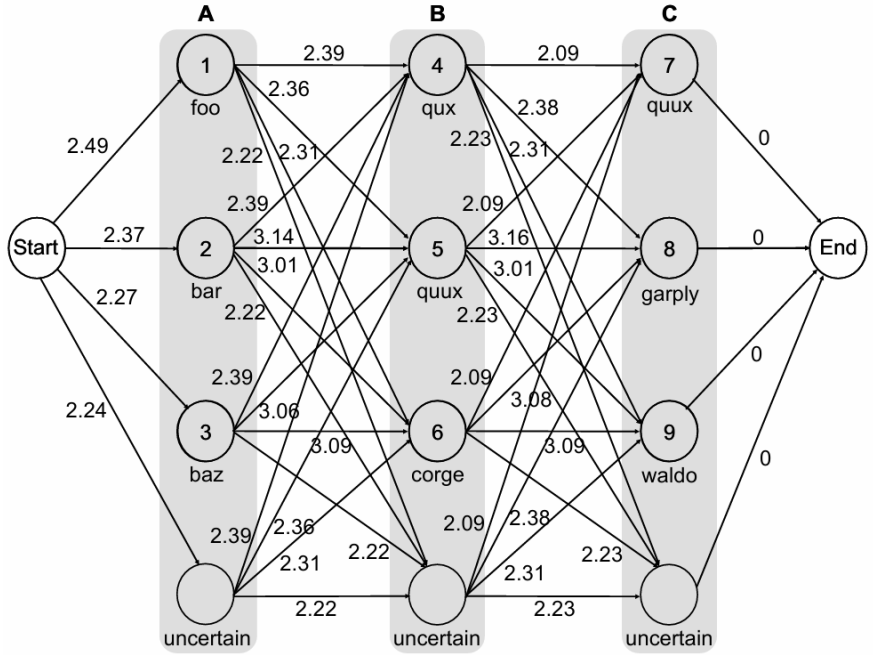
Phase 1: Graph Construction

Directed layered graph

- Columns = unknown functions in binary order
- Nodes = candidate matches
- Uncertain nodes add robustness to incomplete corpora

Edge weights combine four scores

- **Similarity,** from the underlying matcher
- **Confidence,** higher when more unique features match
- **Library,** higher when the match's library is rarer in the corpus
- **Adjacency,** how well two connected matches fit together



(a) Constructed directed acyclic graph.

Phases 2 and 3: Graph Traversal

Phase 2: Graph traversal

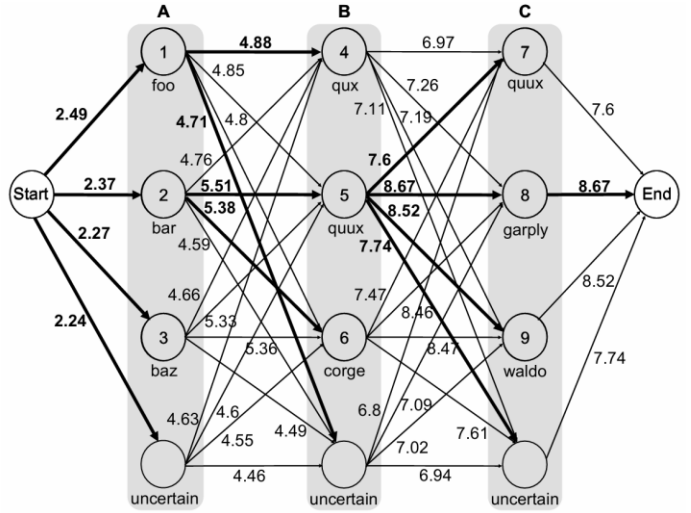
- Viterbi forward pass finds nodes on a maximum-weight path

Phase 3: Ranking

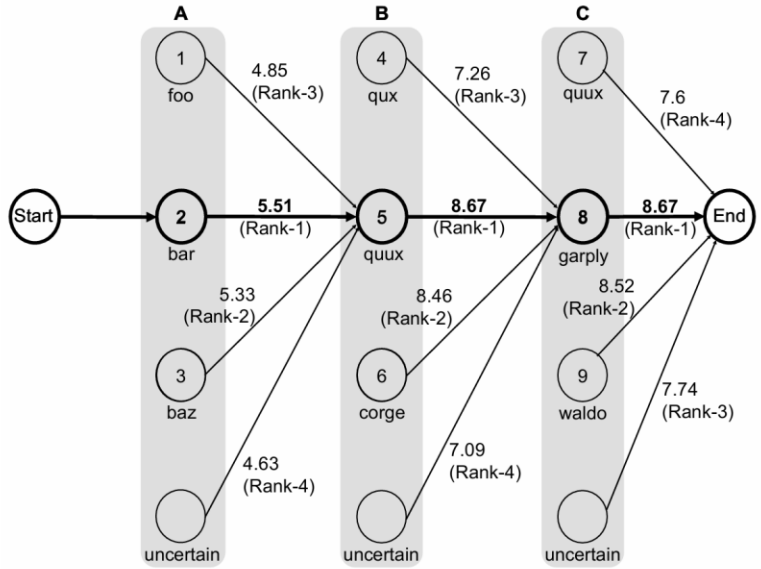
- Backward pass ranks matches by proximity to a maximum-weight path
- Result: refined ranking per unknown function

Reading Figure 1(b) and 1(c)

- In (b), **bold edges** mark each node's maximum incoming path.
- In (c), **bold nodes** lie on maximum weight paths, and the labels show the final rank.



(b) Graph after calculating the weight for each path. Bolded edges denote the maximum incoming path of each dest. node.



(c) Graph after ranking. Bolded nodes are the nodes falling on maximum-weight paths.

Evaluation Setup

Datasets:

- General-purpose: 132 binaries, 262,486 functions
- Synthetic Frankenbinaries: 168 libraries, 107,643 functions

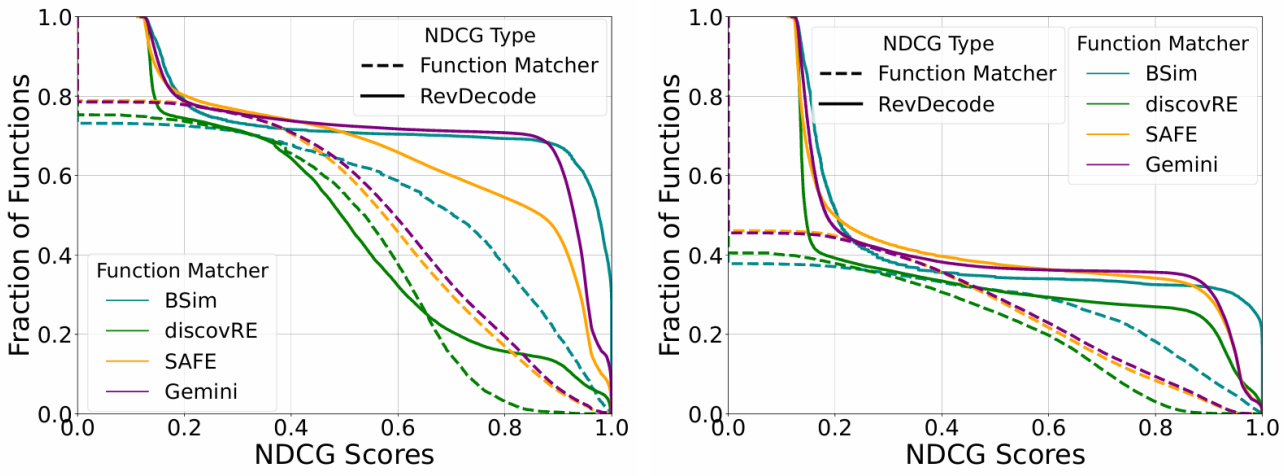
Metrics:

- Tie-aware Normalized Discounted Cumulative Gain (NDCG), a measure of ranking quality

Matchers Evaluated:

- BSim, discovRE, SAFE, Gemini

Results: Ranking Effectiveness

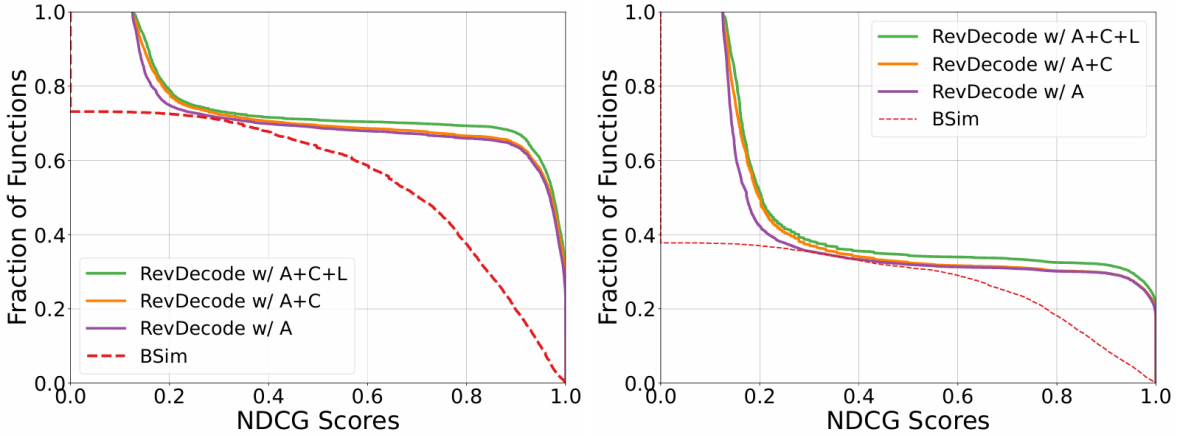


(a) All functions. (b) Without shared libs.

Figure 4: NDCG scores of rankings derived from raw similarity scores (dashed lines) versus rankings produced by REVDECODE using the same matchers (solid lines), evaluated on the general-purpose dataset.

Results: Ablation Study

Every weight component adds value (BSim, general-purpose dataset; baseline NDCG = 0.27)



(a) All functions. (b) Without shared libs.

Figure 5: NDCG scores for REVDECODE variants with different scoring components. Rankings are derived using only Adjacency (**A**), Confidence (**C**), and Library (**L**) scores, or combinations thereof.

Conclusion

Summary

- Relevance from a function's surrounding context
- Match selection as graph exploration over function interdependencies
- GPU parallelizes traversal → scales to large binaries

Personal Thoughts (limitations)

- Corpus incompleteness unsolved; real firmware sets dropped → limited real-world validation
- Fragile memory-adjacency assumption
- Dependent on corpus coverage and underlying matcher quality
- Risk of error propagation through incorrect context chains
- Heuristic, dataset-dependent scoring parameters

Thank You