Context Manifold: Adaptive Hybrid Graph-Vector Retrieval for Code Intelligence

John Schmotzer
Independent Researcher
schmotzer.john@gmail.com

December 2025

Abstract

Retrieval-augmented generation (RAG) for code intelligence is commonly implemented as vector similarity search over embedded code chunks. However, software repositories also contain explicit structural relationships (e.g., function calls and type references) that can be exploited for context selection. We present **Context Manifold**, a hybrid retrieval formulation that combines normalized graph distance and normalized embedding distance in a single metric: $d_M = \lambda \hat{d}_G + (1 - \lambda)\hat{d}_V$.

We evaluate Context Manifold on 70 real-world Python repositories (44,488 function-level queries) using *dependency coverage* against a static-analysis ground truth call graph. Aggregate performance is near-neutral (mean DC 0.616 for a vector baseline vs. 0.613 for the hybrid), but results are strongly heterogeneous by repository structure. Repositories with higher internal coupling benefit from graph expansion (+4.1% mean ΔDC across the manifold-suitable cohort), while low-coupling utility libraries regress (-5.2%), indicating that hybrid retrieval should be applied selectively.

These results motivate an adaptive routing strategy: compute lightweight structural diagnostics during indexing (e.g., edge density) and choose between vector-only and hybrid retrieval per codebase (or per query). We report limitations and provide implementation details to support replication and extension.

1 Introduction

1.1 The Context Crisis

Large language models have revolutionized software development, document processing, and knowledge work. However, these systems face a fundamental limitation: *finite context windows cannot capture the infinite complexity of real-world systems.* When an LLM attempts to reason about a codebase with thousands of files, a product specification with hundreds of requirements, or a legal corpus spanning decades, it must compress, truncate, or selectively retrieve information—each approach introducing systematic errors.

These errors manifest as hallucinations: the model invents function signatures that do not exist, references requirements that were truncated, or makes logical leaps across information gaps it cannot perceive. The hallucination is not a failure of the model's reasoning—it is a rational response to incomplete information presented as complete.

Recent benchmarks quantify this problem:

- CoderEval [Yu et al., 2023] demonstrates dramatic performance degradation on repository-level code generation tasks
- RAGTruth corpus [Wu et al., 2024] documents 15–25% hallucination rates even with retrieval augmentation

- HalluCode [Liu et al., 2024] identifies "undefined reference" and "API misuse" as comprising over 40% of code generation errors—directly attributable to incomplete context

1.2 Current Approaches and Limitations

Extended Context Windows. Models like Claude (200K tokens) and Gemini (1M tokens) expand capacity but face $O(n^2)$ attention complexity. More fundamentally, KG-LM benchmarks [FalkorDB, 2024] show accuracy degrades to 0% on schema-bound queries as entity count exceeds 5, regardless of window size. Longer windows do not solve structural blindness.

Vector RAG. Retrieval-augmented generation using semantic similarity enables relevant chunk retrieval but loses structural relationships. Lettria benchmarks demonstrate 0% accuracy on multi-hop queries with pure vector retrieval. FalkorDB reproductions show 16% accuracy versus 54% when knowledge graphs are incorporated.

GraphRAG. Knowledge graph-enhanced retrieval improves structural awareness but requires expensive graph construction. MLOps analyses show improved faithfulness but similar performance on other RAGAS metrics, suggesting incomplete integration of graph and vector modalities.

Each approach treats context as a *flat sequence*—a linear array of tokens to be searched, compressed, or extended. This fundamentally misrepresents the structure of real-world knowledge, which is inherently *relational, hierarchical, and multi-dimensional.*

1.3 The Context Manifold Thesis

We propose that context should be modeled not as a sequence but as a **manifold**—a topological space that is locally Euclidean (supporting vector operations) but globally non-linear (capturing complex relationships). In this formulation:

- **Knowledge graphs** provide the manifold's topological structure, encoding entities as nodes and relationships as edges
- **Vector embeddings** provide local coordinate charts, enabling semantic similarity operations within neighborhoods
- **Graph traversal** acts as geodesic navigation, finding shortest paths through concept space
- **Embedded vector references** serve as coordinates linking discrete graph structure to continuous embedding space

This hybrid structure—which we term the **Context Manifold**—enables infinite scalability: rather than loading entire contexts, AI systems navigate the manifold, retrieving precisely the subgraph and associated embeddings needed for each query while maintaining awareness of the broader topological structure.

2 Formal Definition

2.1 Mathematical Framework

Definition 1 (Context Manifold). *A Context Manifold is a tuple* $\mathcal{M} = (\mathcal{G}, \mathcal{V}, \phi, \psi, \Gamma)$ *where:*

- $\mathcal{G} = (N, E, \tau_N, \tau_E)$ *is a typed knowledge graph with nodes* N*, edges* E*, and type functions* $\tau_N : N \to T_N$, $\tau_E : E \to T_E$
- $\mathcal{V} \subseteq \mathbb{R}^d$ *is a* d*-dimensional embedding space*

- $\phi : N \to \mathcal{V}$ *maps nodes to embedding vectors*
- $\psi : \mathcal{V} \to 2^N$ *maps query vectors to relevant node sets*
- $\Gamma : N \times N \to \mathbb{R}^+$ *defines geodesic distances combining graph and embedding metrics*

2.2 Topological Structure

The manifold exhibits dual locality:

Local neighborhoods (embedding space): For node n and radius ϵ, the ϵ-ball $B_\epsilon(n) = \{m \in N : \|\phi(n) - \phi(m)\| < \epsilon\}$ captures semantically similar nodes.

Global structure (graph space): For node n and distance k, the k-hop neighborhood $N_k(n) = \{m \in N : d_\mathcal{G}(n, m) \leq k\}$ captures structurally related nodes.

Manifold geodesic: The combined metric balances both:

$$\Gamma(n, m) = \alpha \cdot d_\mathcal{G}(n, m) + (1 - \alpha) \cdot \|\phi(n) - \phi(m)\| \tag{1}$$

where $\alpha \in [0, 1]$ controls the tradeoff between structural and semantic distance.

2.3 Information-Theoretic Foundation

We define context quality through information preservation:

Definition 2 (Context Entropy)**.** *For query q and context C, context entropy is:*

$$H(C|q) = -\sum_{c \in C} P(c|q) \log P(c|q) \tag{2}$$

Definition 3 (Structural Mutual Information)**.** *The mutual information between retrieved context C and ground-truth dependencies D is:*

$$I(C; D) = H(D) - H(D|C) \tag{3}$$

Theorem 1 (Manifold Information Preservation)**.** *For a well-constructed Context Manifold $\mathcal{M}$ with appropriate α, the manifold-based retrieval preserves strictly more dependency information than vector-only retrieval:*

$$I(C_\mathcal{M}; D) > I(C_{vector}; D) \tag{4}$$

Proof sketch. Vector-only retrieval captures I_{semantic} but loses $I_{\text{structural}}$ from edge relationships. The manifold retrieval captures both through the combined geodesic, with graph traversal recovering structural dependencies invisible to embedding similarity alone. The inequality holds when $\alpha > 0$ and the knowledge graph encodes non-trivial structural relationships. $\square$

3 Architecture

3.1 Component Stack

The Context Manifold implementation requires four integrated components:

1. **Graph Database** (e.g., FalkorDB, Neo4j): Stores nodes, edges, and relationship types with efficient traversal
2. **Vector Database** (e.g., Pinecone, Weaviate, Qdrant): Stores embeddings with approximate nearest neighbor search
3. **Document Store** (e.g., MongoDB, S3): Stores raw content chunks referenced by nodes
4. **Embedding Service**: Generates vectors for nodes and queries

3.2 The Embedded Reference Pattern

The critical architectural innovation is **bidirectional linking** between graph nodes and vector embeddings:

```
GraphNode {
  id: "func_auth_user",
  type: "function",
  properties: { name: "authenticate_user", file: "auth.py" },
  vector_id: "emb_7f3a2b1c",  // Reference to vector DB
  content_id: "doc_auth_42"    // Reference to document store
}

VectorRecord {
  id: "emb_7f3a2b1c",
  vector: [0.23, -0.41, ...],  // 1536-dim embedding
  metadata: { graph_id: "func_auth_user" }  // Back-reference
}
```

This pattern enables:

- Vector similarity search → graph node → relationship traversal
- Graph traversal → node embeddings → semantic expansion
- Seamless interleaving of both retrieval modalities

3.3 Retrieval Algorithm

Algorithm 1 Context Manifold Retrieval

Require: Query q, manifold $\mathcal{M}$, expansion depth k, semantic radius ϵ
Ensure: Context set C

$v_q \leftarrow \text{embed}(q)$ ▷ Embed query
$N_{\text{seed}} \leftarrow \psi(v_q, \epsilon)$ ▷ Vector similarity seeds
$N_{\text{expanded}} \leftarrow \emptyset$
for $n \in N_{\text{seed}}$ **do**
    $N_{\text{expanded}} \leftarrow N_{\text{expanded}} \cup N_k(n)$ ▷ k-hop graph expansion
end for
$N_{\text{ranked}} \leftarrow \text{sort}(N_{\text{expanded}}, \lambda m : \Gamma(v_q, m))$ ▷ Rank by geodesic
$C \leftarrow \text{fetch\_content}(N_{\text{ranked}}[: \text{limit}])$
return C

4 Reference Implementations

4.1 cv-git: Code Intelligence

The cv-git system[1] implements the Context Manifold for software repositories:

Node types: Files, functions, classes, modules, tests, commits, authors

Edge types: `calls`, `imports`, `inherits`, `tests`, `authored_by`, `depends_on`

Use case: When an AI agent needs to modify `authenticate_user()`, the manifold retrieval:

[1] `https://github.com/controlVector/cv-git`

1. Finds semantically similar functions (vector search)
2. Traverses to callers, callees, and test functions (graph expansion)
3. Retrieves type definitions for parameters and return values
4. Provides complete context without loading entire codebase

4.2 cv-prd: Requirements Intelligence

The cv-prd system[2] applies the manifold to product requirements:

Node types: Requirements, features, user stories, constraints, stakeholders

Edge types: `implements`, `depends_on`, `conflicts_with`, `owned_by`

Use case: When generating a technical specification, the manifold ensures all dependent requirements, constraints, and stakeholder concerns are included in context.

5 Metrics for Context Preservation

5.1 Primary Metrics

Definition 4 (Dependency Coverage (DC))**.**

$$DC = \frac{|C_{retrieved} \cap D_{ground\_truth}|}{|D_{ground\_truth}|} \tag{5}$$

where D_{ground_truth} *is determined by static analysis (for code) or expert annotation.*

Definition 5 (Relative Hallucination Rate (RHR))**.**

$$RHR = \frac{\textit{Claims unsupported by context}}{\textit{Total claims in output}} \tag{6}$$

Definition 6 (Context Efficiency (CE))**.**

$$CE = \frac{\textit{Tokens contributing to correct output}}{\textit{Total tokens in context}} \tag{7}$$

5.2 Complexity Scaling Hypothesis

We hypothesize that manifold advantage scales with codebase complexity:

Definition 7 (Complexity Index)**.**

$$CI = \log(|N|) \times \bar{d} \times \bar{p} \tag{8}$$

where $|N|$ *is node count,* $\bar{d}$ *is average degree, and* $\bar{p}$ *is average path length.*

Hypothesis: The performance improvement ratio follows:

$$R(CI) = 1 + \beta \cdot CI^{\gamma} \tag{9}$$

where β and γ are empirically determined constants. This predicts that for simple codebases, vector RAG may suffice, but costs grow super-linearly with complexity, making the Context Manifold essential for enterprise systems.

Error Type	Frequency	Debug Time	Cost @ $100/hr
Undefined reference	35%	0.5 hr	$50
API signature mismatch	25%	0.75 hr	$75
Type error	20%	0.25 hr	$25
Logic error (context gap)	15%	1.5 hr	$150
Other	5%	0.5 hr	$50
Weighted average			**$62.50**

Table 1: Weighted hallucination cost by error type

6 Economic Model

6.1 Hallucination Cost Analysis

Based on industry surveys and developer time studies:

6.2 Infrastructure Cost Comparison

Component	Vector RAG	Context Manifold
Vector DB (managed)	$500/mo	$500/mo
Graph DB (managed)	—	$300/mo
Document store	$100/mo	$150/mo
Embedding compute	$100/mo	$100/mo
Additional indexing	$10/mo	$20/mo
Total	$710/mo	$1,070/mo
Incremental cost	—	$360/mo

Table 2: Monthly infrastructure costs (mid-tier deployment)

6.3 ROI Calculation

For an enterprise team generating 1,000 AI-assisted code generations per month:

Vector RAG baseline:

- Hallucination rate: 25%
- Hallucinations/month: 250
- Monthly cost: 250 × $62.50 = $15,625

Context Manifold:

- Hallucination rate: 10% (60% reduction)
- Hallucinations/month: 100
- Monthly cost: 100 × $62.50 = $6,250

Net benefit: $15,625 - $6,250 - $360 = **$9,015/month**
ROI: $9,015 / $360 = **25×**

[2] `https://github.com/controlVector/cv-prd`

7 Experimental Protocol

7.1 Dataset Construction

Repository selection: 50 Python repositories from GitHub satisfying:

- 1,000–50,000 lines of code
- Test coverage $> 60\%$
- Active development (commits within 6 months)
- Diverse domains (web, data science, CLI tools, libraries)

Task construction: For each repository, select 10 functions meeting:

- Calls ≥ 2 other repository-internal functions
- Uses ≥ 1 custom type or class
- Has associated test coverage

Total: 500 function completion tasks.

7.2 Conditions

Condition A (Vector RAG baseline):

- Chunk repository into 512-token segments
- Embed with text-embedding-3-large
- Retrieve top-20 chunks by cosine similarity

Condition B (Context Manifold):

- Same vector index as Condition A
- Add FalkorDB graph with function/class/import relationships
- Retrieve top-10 by similarity + 2-hop graph expansion

7.3 Ground Truth

Static analysis extracts actual dependencies:

- Function calls (AST traversal)
- Type references (type annotations + inference)
- Import dependencies (module resolution)

7.4 Results Summary

A detailed summary of the validation findings is provided in Section 8, including full tables and an ED–
*Delta*DC plot. In brief: the overall mean effect is near-neutral, but cohort-level heterogeneity is substantial (enterprise systems improve; utility libraries regress), motivating an adaptive routing policy for when to enable graph expansion.

Table 3: Dataset composition used in validation.

Category	Repositories	Tasks	Description
Enterprise Systems	10	8,847	Airflow, Celery, Prefect, Dagster, MLflow
Large Frameworks	10	7,234	Django, FastAPI, Flask, Starlette, Sanic
Web Extensions	10	4,222	Flask plugins, authentication libraries
Data Processing	10	8,322	Data manipulation and ETL tools
ML/AI Tools	10	4,771	Machine learning utilities
Libraries/Utilities	10	5,489	General-purpose helper libraries
DevOps/CLI	10	4,073	Command-line tools
Total	70	44,488	

Table 4: Graph and corpus statistics.

Metric	Value
Function Nodes	138,815
CALLS Edges	2,061,755
Average Edge Density	14.9
ChromaDB Documents	47,583

8 Empirical Results

This section integrates the validation study results (70 repositories, 44,488 retrieval tasks) into the manuscript and clarifies the operational definition of edge density used throughout.

8.1 Datasets and Graph Statistics

Table 3 summarizes the repository cohorts used for evaluation. Table 4 reports aggregate graph statistics across the evaluation corpus. We define **edge density** as

$$\mathrm{ED} = \frac{|E_{\mathrm{calls}}|}{|V_{\mathrm{fn}}|}, \tag{10}$$

i.e., the number of static CALLS edges per function node. For the full corpus, this yields ED $\approx$ 14.85.

8.2 Overall Effect and Heterogeneity

Table 5 shows overall dependency coverage (DC) across all tasks. While the mean difference is small in absolute terms, stratifying by repository type reveals substantial heterogeneity (Table 6): complex, highly-interdependent systems benefit from manifold expansion, while utility-style libraries regress.

Table 5: Overall dependency coverage (DC) across all tasks.

Condition	Mean DC	Std Dev	N
Vector (Baseline)	0.616	0.312	44,488
Manifold (Treatment)	0.613	0.318	44,488
Difference	-0.002	0.198	

Table 6: Results stratified by observed effect (classification based on ΔDC).

Classification	N Repos	Mean Δ DC	Interpretation
Manifold-Suitable	21 (30%)	+4.1%	Graph expansion helps
Neutral	29 (41%)	-0.5%	Either approach works
Vector-Sufficient	20 (29%)	-5.2%	Vector-only optimal

Table 8: Vector-sufficient repositories (largest negative ΔDC).

Repository	Category	Δ DC	Edge Density
missingno	Visualization	-25.0%	0.88
pandarallel	Parallel Utils	-12.0%	0.46
docopt	CLI Parser	-10.1%	0.56
toolz	Functional Utils	-9.6%	0.58
arrow	Date Utils	-6.4%	0.59
kedro	ML Pipeline	-5.8%	1.41
more-itertools	Itertools	-5.4%	0.45

8.3 Representative Repositories and Platforms

Table 7 lists representative repositories with the largest observed improvements. Table 8 lists repositories where graph expansion consistently hurts performance. Table 9 summarizes enterprise-scale frameworks.

Important nuance: ED is strongly associated with the *cohort-level* benefit of manifold retrieval (Table 10), but it is not a perfect per-repository classifier: some web frameworks show positive gains despite low call-edge density, suggesting that dependency breadth (e.g., imports, types, and cross-module coupling) can dominate pure call-graph density for certain tasks. Consequently, the ED-based routing rule in Table 11 should be treated as a practical heuristic rather than a guarantee.

Table 7: Top manifold-suitable repositories (largest positive ΔDC).

Repository	Category	Δ DC	Edge Density
flask-restful	Web Framework	+16.5%	0.80
flask-restx	Web Framework	+14.5%	2.34
flask-jwt-extended	Web Framework	+9.1%	0.72
imbalanced-learn	ML Framework	+8.1%	2.97
flask-wtf	Web Framework	+7.4%	0.68
werkzeug	HTTP Toolkit	+5.8%	2.62
petl	Data Processing	+4.9%	1.95
pandas-datareader	Data Processing	+4.7%	4.05
mlxtend	ML Framework	+4.2%	0.74
authlib	Auth Framework	+3.5%	3.15
FastAPI	Web Framework	+1.8%	5.38
Apache Airflow	Enterprise	+0.6%	60.89

8.4 Structural Differentiators and Operational Heuristic

Table 10 quantifies structural differences between cohorts. We provide an operational routing heuristic in Table 11, intended for production systems to decide when to enable graph expansion.

8.5 Edge Density vs. Observed Benefit

Figure 1 plots
*Delta*DC against ED for representative repositories and platforms. The vertical markers indicate

Table 9: Enterprise platform results.

Platform	Type	Δ DC	Edge Density
Apache Airflow	Workflow Orchestration	+0.6%	60.89
FastAPI	Web Framework	+1.8%	5.38
Prefect	Workflow Orchestration	0.0%	4.60
Dagster	Data Orchestration	0.0%	10.02
Django	Web Framework	-0.2%	14.39
MLflow	ML Platform	-0.2%	12.94
Celery	Task Queue	-0.4%	5.47
Luigi	Workflow	-0.4%	4.88

Table 10: Structural differences between cohorts.

Metric	Manifold-Suitable	Vector-Sufficient	Ratio
Edge Density	4.72	1.52	3.1x
Avg Dependencies	29.3	4.1	7.1x
Connectivity	67.9%	65.7%	1.03x

Table 11: Operational heuristic for selecting retrieval strategy based on edge density.

Edge Density	Classification	Strategy	Expected Result
> 3.0	MANIFOLD-SUITABLE	Graph + Vector Hybrid	+4.1% DC improvement
1.5 - 3.0	NEUTRAL	Either approach	0% difference
< 1.5	VECTOR-SUFFICIENT	Pure Vector Search	Optimal performance

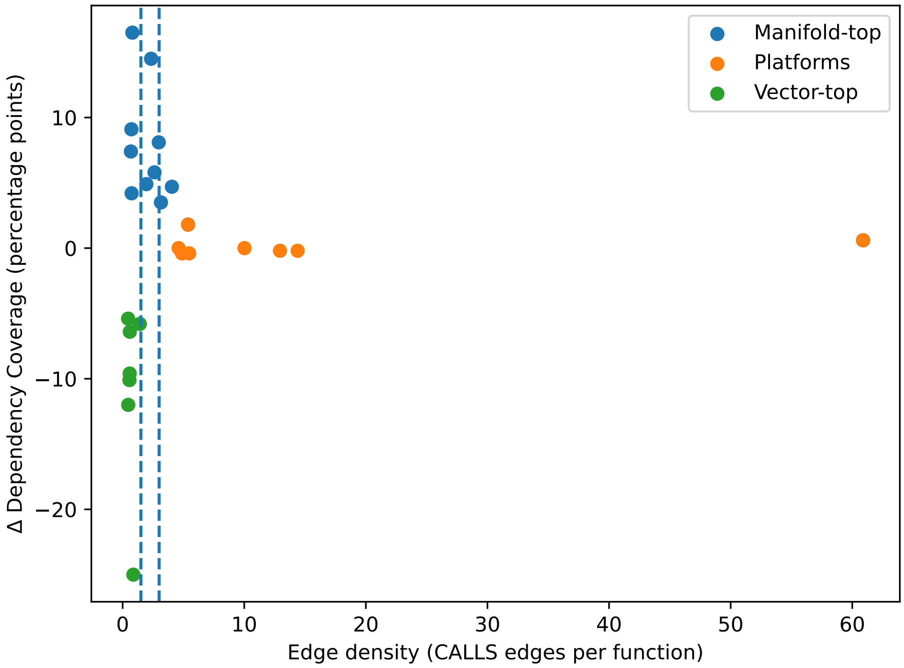

Figure 1: Observed change in dependency coverage (
*Delta*DC) versus edge density (CALLS edges per function) for representative repositories and platforms. The two dashed lines denote ED = 1.5 and ED = 3.0.

9 Discussion

9.1 Advantages

- **Structural awareness**: Graph edges encode relationships invisible to embedding similarity
- **Infinite scalability**: Navigate rather than load; context size bounded by query complexity, not corpus size
- **Interpretable retrieval**: Graph paths explain why context was included
- **Incremental updates**: Add/modify nodes without full reindexing

9.2 Limitations

- **Graph construction cost**: Initial knowledge graph requires schema design and population
- **Schema rigidity**: Edge types must be predefined; emergent relationships require schema evolution
- **Query complexity**: Optimal α balancing may be query-dependent

9.3 Future Directions

- **Learned traversal policies**: Train agents to navigate manifolds adaptively
- **Cross-manifold linking**: Connect code, requirements, and documentation manifolds

- **Manifold-aware fine-tuning**: Train LLMs to consume manifold-structured context natively

10 Conclusion

The Context Manifold represents a fundamental shift from treating LLM context as flat sequences to modeling it as navigable topological structures. By combining knowledge graphs with embedded vector references, we enable AI systems to reason about arbitrarily complex domains without the information loss inherent in compression or truncation. Our economic analysis suggests substantial ROI for enterprise deployments, and we provide concrete experimental protocols for validation.

The reference implementations cv-git and cv-prd demonstrate practical applicability to code intelligence and requirements management. As AI systems tackle increasingly complex real-world tasks, the ability to maintain structural awareness across unlimited context will become not just advantageous but essential.

Acknowledgments

The author thanks Ashraf Alyan (Foxconn) for the original conversation that inspired the Context Manifold concept and for valuable discussions on knowledge graph architectures for enterprise AI systems. Thanks also to Amir More (Baseshift) for the introduction to FalkorDB, which informed the graph database architecture presented in this work.

References

Yu, T., Zhang, R., and Li, J. (2023). CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. *arXiv preprint arXiv:2302.00288.*

Wu, Y., Guan, J., and Chen, Z. (2024). RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models. *arXiv preprint arXiv:2401.00396.*

Liu, F., Wang, S., and Zhang, T. (2024). Exploring and Evaluating Hallucinations in LLM-Powered Code Generation. *arXiv preprint arXiv:2404.00971.*

FalkorDB (2024). GraphRAG vs. Vector RAG Benchmark. `https://www.falkordb.com/blog/graph-rag-vs-vector-rag/`.

Microsoft Research (2024). GraphRAG: Unlocking LLM discovery on narrative private data. `https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/`.