future internet

MDPI

*Article*

# Semantic Search for System Dynamics Models Using Vector Embeddings in a Cloud Microservices Environment

**Pavel Kyurkchiev [1], Anton Iliev [1,2,*] and Nikolay Kyurkchiev [1,2,3,*]**

[1] Faculty of Mathematics and Informatics, University of Plovdiv "Paisii Hilendarski", 24, Tzar Asen Str., 4000 Plovdiv, Bulgaria; pkyurkchiev@uni-plovdiv.bg
[2] Centre of Excellence in Informatics and Information and Communication Technologies, 1113 Sofia, Bulgaria
[3] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev Str., Bl. 8, 1113 Sofia, Bulgaria
* Correspondence: aii@uni-plovdiv.bg (A.I.); nkyurk@uni-plovdiv.bg (N.K.)

**Abstract**

Efficient retrieval of mathematical and structural similarities in System Dynamics models remains a significant challenge for traditional lexical systems, which often fail to capture the contextual dependencies of simulation processes. This paper presents an architectural approach and implementation of a semantic search module integrated into an existing cloud-based modeling and simulation system. The proposed method employs a strategy for serializing graph structures into textual descriptions, followed by the generation of vector embeddings via local ONNX inference and indexing within a vector database (Qdrant). Experimental validation performed on a diverse corpus of complex dynamic models, compares the proposed approach against traditional information retrieval methods (Full-Text Search, Keyword Search in PostgreSQL, and Apache Lucene with Standard and BM25 scoring). The results demonstrate the distinct advantage of semantic search, achieving high precision (over 90%) within the scope of the evaluated corpus and effectively eliminating information noise. In comparison, keyword search exhibited only 24.8% precision with a significant rate of false positives, while standard full-text analysis failed to identify relevant models for complex conceptual queries (0 results). Despite a recorded increase in latency (~2 s), the study proves that the vector-based approach is a significantly more robust solution for detecting hidden semantic connections in mathematical model databases, providing a foundation for future developments toward multi-vector indexing strategies.

**Keywords:** system dynamics; semantic search; microservices; vector embeddings; Natural Language Processing (NLP); information retrieval; cloud simulation; ONNX inference; graph serialization



Academic Editor: Rafael Valencia-Garcia

Received: 26 December 2025
Revised: 30 January 2026
Accepted: 3 February 2026
Published: 5 February 2026



## 1. Introduction

Modeling and simulation of dynamic processes through the System Dynamics methodology has established itself as an indispensable tool for decision-making in complex social, economic, and business domains. The foundations of this approach were laid by Forrester's pioneering work on Industrial Dynamics [1], which defined the initial frameworks for modeling feedback loops in managerial decision-making. Although the methodology was established more than six decades ago, it has not lost its relevance; on the contrary, it continues to evolve. According to contemporary analyses by Sterman [2], System Dynamics today serves not merely as a technical tool for simulation, but as a means for learning and overcoming cognitive barriers in an increasingly complex world. With the advent of cloud

technologies, modeling platforms have evolved from monolithic desktop applications to distributed web-based systems, enabling multi-tenant collaboration and transforming into large-scale repositories of simulation models [3,4].

Despite advancements in cloud infrastructure and the exponential increase in computing power, the effective discovery and reuse of System Dynamics models remains an open challenge. Traditional Information Retrieval (IR) approaches rely primarily on lexical keyword matching (Keyword Search), implemented via SQL queries or Full-Text Search indexes [5]. This approach suffers from the so-called "Vocabulary Mismatch problem," defined by [6] and observed in the retrieval of software artifacts by McMillan et al. [7]. These issues mirror the challenges in semantic code search described by Husain et al. [8]. In the context of System Dynamics, these challenges are even more pronounced, as the models represent a form of Visual Programming. Unlike linear code, the logic here is distributed across the graph topology. Consequently, effective retrieval requires the algorithm to "read" not only the labels but also the structural logic of the simulation—a task beyond the capabilities of standard SQL indexes.

Over the years, advancements in Natural Language Processing (NLP) and transformer architectures (such as BERT), pioneered by Devlin et al. [9], have led to the paradigm of "Semantic Search." Unlike standard search methods, BERT employs a bidirectional attention mechanism, allowing it to understand the context of words in depth. However, the direct application of BERT for semantic search is computationally inefficient. To apply these capabilities to information retrieval tasks, textual data must be transformed into vector spaces—a concept rooted in the Vector Space Model of Salton et al. [10], where semantic similarity is measured via metrics such as cosine similarity. Research by Reimers and Gurevych [11] demonstrates that optimized BERT-based architectures (such as Sentence-BERT) enable the generation of high-quality vector embeddings with high computational efficiency. Recent studies from 2024 have further validated this shift, demonstrating that Retrieval-Augmented Generation (RAG) architectures significantly outperform lexical methods in specialized engineering domains [12,13].

Despite the existence of these established technologies and their widespread application in text documents and source code, their specific use for visual System Dynamics models in a cloud environment remains underexplored. Contemporary solutions largely rely on heavy external API services, raising concerns regarding data privacy and system latency.

This paper addresses the problem of model discoverability by proposing an architecture for semantic indexing and retrieval of System Dynamics models, integrated within a microservice cloud environment. This approach aims to minimize the "hidden technical debt" in machine learning system deployment [14], ensuring scalability and resilience. Unlike previous implementations focused primarily on architectural modularity [15] or foundational simulation theory [16], here we present the implementation of an intelligent retrieval module utilizing local ONNX inference. Experimental results demonstrate that the proposed approach achieves high precision (averaging 98%) on the tested corpus for complex conceptual queries, completely eliminating the information noise inherent in traditional lexical methods.

The remainder of this paper is organized as follows. Section 2 reviews the related work and current state of the art in model retrieval. Section 3 presents the proposed cloud-based microservices architecture and infrastructure components. Section 4 details the methodology for semantic linearization and vector embedding of graph structures. Section 5 provides the experimental setup and discusses the comparative results. Finally, Section 6 concludes the paper and outlines directions for future work.

## 2. Related Work

The issue of discoverability and reusability of simulation models has been the subject of numerous studies, which can be categorized into three main directions: traditional repositories, graph-based approaches, and contemporary semantic methods.

### 2.1. Traditional Model Repositories

Existing platforms for sharing System Dynamics and mathematical models typically rely on metadata and lexical (keyword-based) search. Research in the field of Information Retrieval (IR) indicates that while this approach is effective for simple queries, it suffers from the fundamental "Vocabulary Mismatch problem" [17]. Users often employ conceptual terms (e.g., "economic stagnancy") that are not explicitly present in the model's variable names, leading to a failure to retrieve relevant resources.

### 2.2. Graph Neural Networks (GNN) in Modeling

With the advent of deep learning, Graph Neural Networks (GNNs) have established themselves as the standard for analyzing structured data. Work by Scarselli et al. [18] demonstrates the success of GNNs in node and graph classification. However, the direct application of GNNs for retrieving System Dynamics models faces significant challenges. Training an effective GNN requires massive annotated datasets, which are often lacking in specialized domains. This creates a "Cold Start" problem, rendering the approach inapplicable for smaller or specialized repositories.

Consequently, applying standard pre-trained GNNs (often designed for molecular or social graph structures) necessitates extensive fine-tuning to capture the causal logic of System Dynamics. Due to the lack of massive labeled datasets in this specific domain, our research deliberately pivots away from graph-native embeddings, opting instead for a semantic linearization strategy that leverages the 'world knowledge' of Large Language Models (LLMs).

### 2.3. Semantic Search and Vector Embeddings

The proposed approach builds upon recent advancements in Natural Language Processing (NLP) and Transformer architectures. Unlike pure graph-based methods, we propose a strategy for serializing the graph into semantically rich text. This allows us to leverage the "world knowledge" encoded in pre-trained Large Language Models (LLMs), bypassing the need to train specialized networks from scratch. This method aligns with the trend of utilizing vector databases for Retrieval-Augmented Generation (RAG), as defined by Lewis et al. [19].

A key differentiator in our research is the avoidance of external cloud API services for embedding generation. Reliance on such APIs often introduces risks regarding corporate model privacy and network latency. Instead, we integrate local inference, ensuring that sensitive model logic does not leave the infrastructure perimeter while simultaneously optimizing computational costs.

## 3. System Architecture

The platform architecture is implemented as a distributed, cloud-ready system, built upon the principles of Domain-Driven Design (DDD) and a modular microservices organization. The foundation of this infrastructure was laid in previous research. The structural framework extends the architectural implementation described in [15], while aligning with the theoretical foundations of modular simulation established by Zeigler et al. [16]. Regarding data persistence, the specifics of cloud integration follow [20], whereas the storage of dynamic mathematical models aligns with the interoperability specifications cited in [21].

In its baseline version, the system defines clear boundaries between the user interface, the domain model, and the numerical equation solving infrastructure (FormulaProcessingService), which constitutes part of the System Dynamic Solver layer.

The current development extends this foundation through the integration of a novel semantic analysis module. Inter-component communication is facilitated asynchronously via RabbitMQ, allowing heavy computational processes to execute without blocking the user interface. As illustrated in Figure 1, the architecture implements a Publisher-Subscriber pattern, where the Application Services layer publishes events upon every model addition or modification, and specialized independent microservices process them:

- **Iris Processor:** This service manages user communication by dispatching notification emails. This occurs automatically upon the completion of model calculations.
- **Hermes Processor:** The key component of the semantic module. This service listens for PrototypeChange events via RabbitMQ, performs local ONNX inference [22], and automatically updates the indexes in the vector database. This architectural approach effectively isolates the resource-intensive vectorization process from the main web server.
- **Data Storage Layer:** A hybrid storage approach is employed:
  - **MongoDB:** Utilized as a document store for the JSON structure of the graphs. Unlike previous iterations of the platform implemented on Apache Cassandra, the current architecture has migrated to MongoDB. This decision was driven by the need to optimize the handling of deep nested JSON objects, which are characteristic of serialized System Dynamics models and are processed more efficiently by a document-oriented storage system.
  - **Qdrant:** A specialized vector database for storing embeddings and executing ANN (Approximate Nearest Neighbor) queries. The connection is implemented via gRPC to ensure minimal latency.

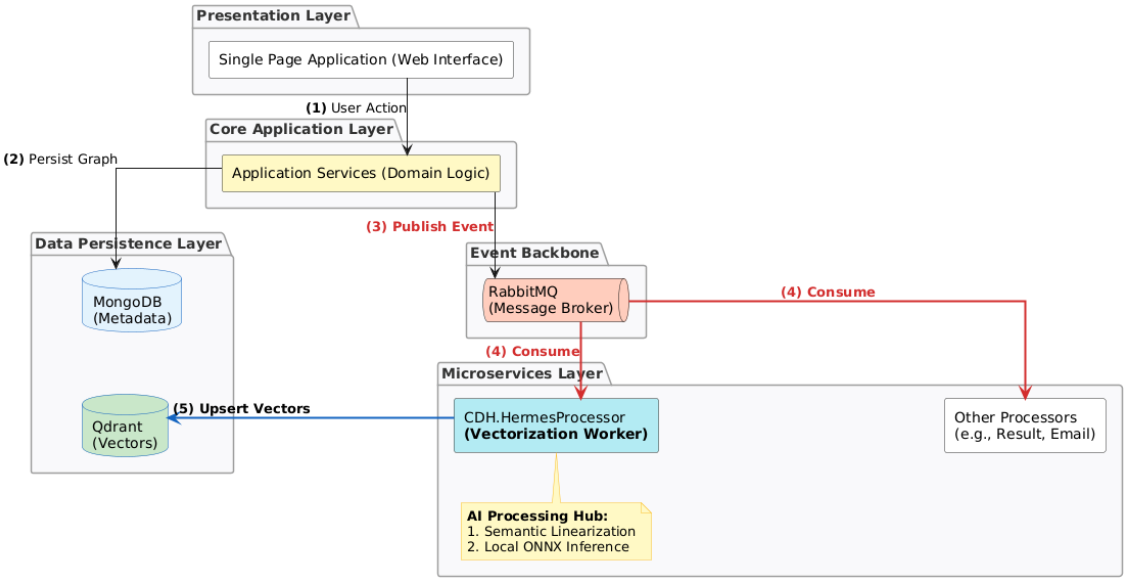


**Figure 1.** High-Level System Architecture & Asynchronous Vectorization Flow.

The semantic processing workflow is implemented within the Hermes Processor, following four key stages:

### *3.1. Input: System Dynamics Graph*

At the core of the domain layer lies the GraphModel class, which represents the structure of System Dynamics models utilizing the Graph Link Model. The data is organized

as a directed graph composed of nodes (NodeDataArray), links (LinkDataArray), and formulas (FormulaDataArray). This graph describes the simulation semantics through specific elements:

- **Stocks:** Represented as Rectangle type nodes, these model the system accumulators (state variables) that change over time through integral equations. Within the NodeDataArray structure, they serve as the primary containers for values.
- **Flows:** Defined as nodes with directed logic (types Valve or Pipe), which control the inflow and outflow rates of stocks, representing the derivatives (rates of change) in the mathematical model.
- **Variables:** Nodes containing constants, graphical functions, or algebraic formulas that define the rules and dependencies between flows and stocks.

### *3.2. Processing: Serialization to Text*

Prior to vectorization, the graph model undergoes a linearization process implemented in two stages. The first stage occurs in the Application Service Layer, where the model prototype is mapped to a domain object, and mathematical formulas are extracted. For every creation or modification operation (within the PrototypeService), the system generates an event that packages the domain object and routes it to an asynchronous processor (Hermes Processor). There, the second stage takes place—the actual linearization, where the hierarchical graph structure is transformed into a rich textual description. The detailed methodology and algorithm for this process are discussed in Section 4.1.

### *3.3. Embedding Service: .NET Core + ONNX*

The key component of the new architecture is the infrastructure service OnnxEmbeddingService. Unlike mainstream solutions relying on external Application Programming Interfaces (such as OpenAI), this implementation utilizes local execution for maximum security and performance.

- **Technology:** The class and its methods are based on Microsoft.ML.OnnxRuntime within a .NET environment.
- **Model:** The nomic-embed-text-v1.5 model is integrated and loaded locally. It was specifically selected for its extended context window (8192 tokens), which allows for the processing of large system structures without fragmentation.
- **Process:** The service performs BERT-style tokenization on the input text and generates a dense vector (embedding vector) that encodes the semantic knowledge of the model in a multidimensional space.

### *3.4. Storage: Qdrant Vector DB*

The generated vectors are managed by the PrototypeVectorRepository class, which implements the interface layer to the Qdrant vector database. This component functions as a specialized store, distinct from the primary non-relational database (MongoDB), and is responsible solely for persisting vector embeddings and their associated UUIDs. Integration is implemented via the gRPC protocol to ensure minimal latency during data transfer between microservices.

To ensure rigorous benchmarking, the architecture supports parallel implementations of standard solutions based on PostgreSQL and Apache Lucene. These components serve as a comparative baseline for efficiency evaluation but do not participate in the primary production flow of semantic search. The architectural layout of these modules is presented in Figure 2.

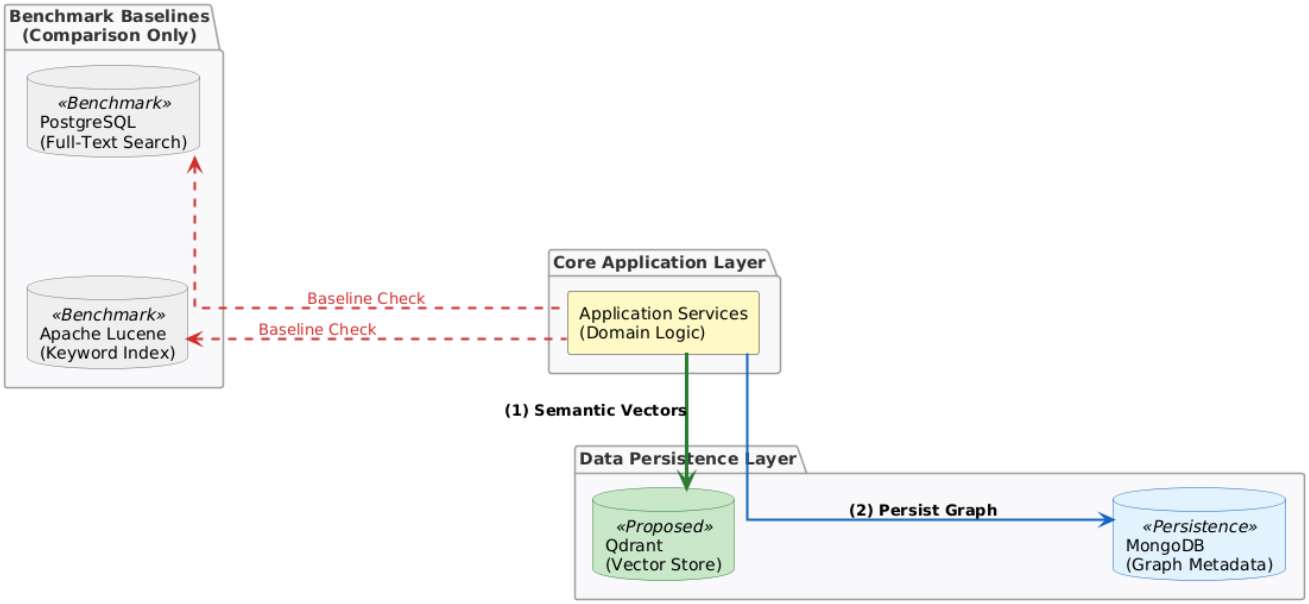


**Figure 2.** System Architecture: Semantic Search Module vs. Benchmark Baselines.

## 4. Methodology: From Graph Model Language to Vector Embeddings

The primary challenge in applying Natural Language Processing (NLP) algorithms to System Dynamics models lies in the fundamental difference in data topology. While NLP models (such as BERT) are trained on linear sequences of symbols (text), System Dynamics models represent Directed Cyclic Graphs. To overcome this discrepancy, a two-stage methodological approach was developed: (1) semantic linearization of the graph structure and (2) aggregation into a unified vector context.

### *4.1. Semantic Linearization*

The indexing process transforms the graphical model into a structured text document (search_document), optimized for vector retrieval.

The choice of a Text-based Serialization strategy over the use of direct Graph Neural Networks (GNNs) is an architectural decision driven by efficiency. While GNNs require massive datasets for training from scratch, the chosen approach allows us to leverage the power of pre-trained Transformer models (LLMs), which already possess vast conceptual knowledge of economic and physical terms. This effectively addresses the "Cold Start Problem" inherent to the scarcity of thousands of annotated System Dynamics models.

Instead of relying solely on flat text, the algorithm generates a hybrid representation that combines topological information with natural language. The generated document consists of four logical segments, illustrated below using an example model of a Financial Liquidity System:

#### 4.1.1. Metadata & Statistics

The document begins by defining the model type and its complexity. This establishes a "global context" for the LLM.

An example of the metadata structure is provided below:

Model Type: System Dynamics; Domain: Financial Economics; Total Components: 18; Total Connections: 24

#### 4.1.2. Path-Based System Structure

This constitutes the core of the linearization process. The algorithm performs a Graph Traversal, tracing the flow of resources and the influence of external market factors.

- **Hierarchical Representation:** A specialized notation is employed (↳ or indentation for nodes, via for links) to preserve nesting depth.
- **Cycle Detection:** The system automatically identifies feedback loops (e.g., profit reinvestment).
- **Conditional Logic:** The conditions for flow activation are explicitly recorded.

To ensure scalability and prevent context window overflow when processing heavily nested cyclic dependencies, the linearization algorithm implements a Depth-First Traversal with State Tracking.

- **Specifically:** Cycle Termination: A Visited hash set tracks nodes within the current recursion stack. Upon encountering a previously visited node, the traversal immediately terminates the current branch and appends a [CYCLE DETECTED] marker. This prevents infinite recursion loops inherent in feedback-rich systems.
- **Recursion Depth Guard:** To mitigate the risk of "text explosion" in highly interconnected graphs, a hard limit on recursion depth ($D_{max}$ = 10) is enforced. Branches exceeding this threshold are truncated, ensuring that the semantic description remains concise and fits within the 8192-token limit of the nomic-embed-text-v1.5 model.

To formalize the serialization process described above, the logic is encapsulated in Algorithm 1. This pseudocode illustrates the specific implementation of the Depth-First Traversal, highlighting the mechanisms for cycle detection and recursion depth control ($D_{max}$).

**Algorithm 1** Semantic Graph Serialization Strategy

**Input:** Graph Model G = (V, E), Maximum Recursion Depth $D_{max}$ = 10
**Output:** Serialized Text Document S

```
function GenerateLinearDescription(G)
    S ← InitializeHeader(G.Metadata)
    Visited ← ∅
    foreach node in G.V sorted by Rank(node) do
        if node.Type ∈ {Stock, Flow} then
            S ← S + TraverseDFS(node, 0, Visited)
        end if
    end for
    return S
end function
function TraverseDFS(current_node, depth, path_history)
    if depth > D_max then
        return "[DEPTH LIMIT REACHED]"
    end if
    indent ← RepeatString(" ", depth)
    text_line ← indent + "? " + current_node.Name + " (" + current_node.Type + ")"
    if current_node ∈ path_history then
        return text_line + " [CYCLE DETECTED]"
    end if
    path_history.Add(current_node)
    foreach link in current_node.OutgoingLinks do
        target ← link.TargetNode
        condition ← ExtractCondition(link)
        text_line ← text_line + "\n" + indent + " via " + link.Label + " [" + condition + "]"
        text_line ← text_line + "\n" + TraverseDFS(target, depth + 1, path_history)
    end for
    path_history.Remove(current_node) ▷ Backtracking step
    return text_line
end function
```

The computational complexity of the proposed serialization strategy is O(V + E), where V denotes the number of nodes (stocks, flows, variables) and E represents the connections. By employing a Visited hash set for state tracking, the algorithm ensures that each node is processed a constant number of times, while the recursion depth guard ($D_{max}$) acts as a hard constraint against exponential text expansion in feedback-rich models. This design guarantees linear scalability relative to the graph topology, rendering the approach computationally efficient for real-time indexing within the microservices environment.

Following this logic, the resulting structure effectively linearizes the graph. An example output generated by this algorithm is presented in Listing 1.

**Listing 1.** Fragment of the linearized structure of a financial model (Path-Based Linearization).

```
Model Context: Financial Liquidity System
Total Nodes: 18 | Cycles Detected: 1
? Market_Demand (variable)
  via influence [always]
  ? Revenue_Inflow (valve)
    via flow_to [always]
    ? Cash_Reserves (stock)
      via payment [conditional: if liquidity > min_threshold]
      ? Operational_Expenses (valve)
      via investment [conditional: if profit_margin > 5%]
      ? Capital_Assets (stock) [cycle detected]
```

#### 4.1.3. Component Categorization

Elements are grouped according to their economic role, enabling the search engine to distinguish between accumulated resources (Stocks) and instantaneous transactions (Flows).

- **Stocks:** Cash_Reserves, Capital_Assets, Inventory.
- **Variables:** Market_Demand, Interest_Rate, Tax_Rate.
- **Flows:** Revenue_Inflow, Operational_Expenses, Depreciation.

#### 4.1.4. Semantic Relationships

The final layer translates technical connections into natural language sentences that describe the underlying economic logic:

- **Flow Relationships:**

  Example 1 (Flow Relationship):

  Revenue_Inflow flows to Cash_Reserves stock via sales operations [always].

  Example 2 (Flow Relationship):

Cash_Reserves stock flows to Operational_Expenses cloud via payment [conditional: if liquidity > min_threshold].

- **Influence Relationships:**

  Example 1 (Influence Relationship):

  Market_Demand variable positively influences Revenue_Inflow valve [always].

  Example 2 (Influence Relationship):

Interest_Rate variable influences Debt_Service valve [conditional: variable rate apply].

This multi-layered structure enables the Embedding model to capture both the topology (the flow of funds through the system) and the semantics (the investment logic) within a single unified vector. This entire process is illustrated in Figure 3.

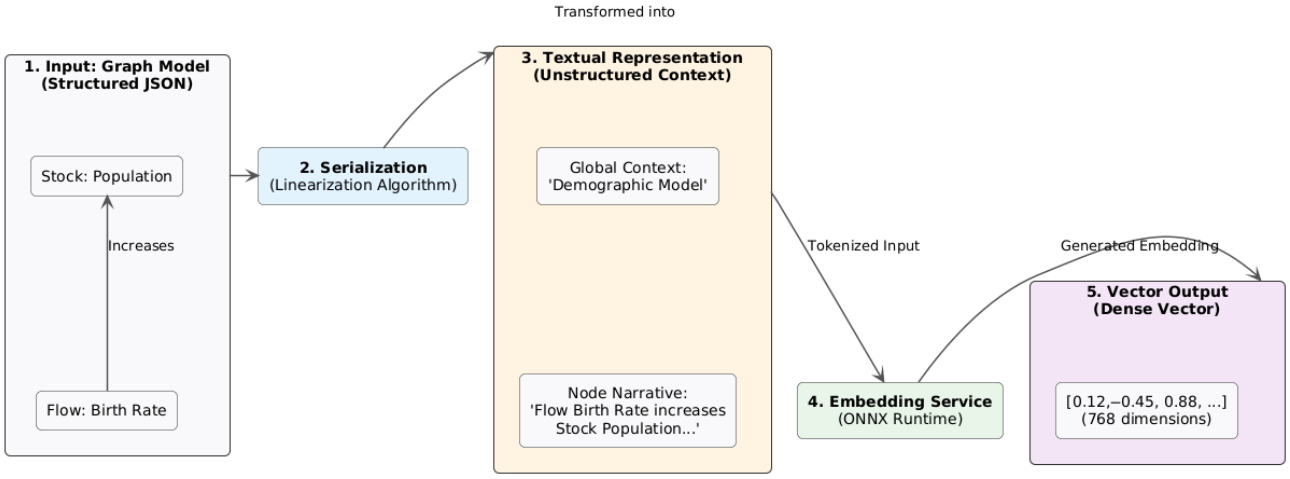


**Figure 3.** Illustration of the methodological process: Transforming a structured System Dynamics graph into linear text and subsequent generation of a vector embedding via an ONNX model.

### *4.2. Strategy of Single-Vector Representation*

In the current implementation, the Single-Vector Embedding approach is adopted. Under this approach, the entire generated text is submitted to the ONNX inference session, which returns a single dense vector with fixed dimensionality (768 dimensions for the utilized nomic-embed-text-v1.5 model).

The selection of this strategy is theoretically and practically justified by the following factors:

- **Holistic Nature of System Dynamics:** The behavior of a dynamic system (e.g., oscillation or exponential growth) is an emergent property—it arises from the interaction of all elements rather than being contained within a single node. The single-vector approach allows the algorithm to compare holistic concepts rather than fragmented parts.
- **Computational Efficiency:** Representing each model as a single point in multidimensional space enables the utilization of the HNSW (Hierarchical Navigable Small World) algorithm, as defined by Malkov and Yashunin [23]. This Approximate Nearest Neighbor (ANN) method guarantees logarithmic search complexity, rendering the system scalable even with millions of indexed models.
- **Noise Elimination:** The aggregation process suppresses the influence of isolated, non-specific terms that would otherwise distort results in more granular approaches.

A critical architectural consideration in the Single-Vector approach is the Context Window (Token Limit) of the underlying Transformer model. Our implementation employs a Semantic Truncation strategy. If the textual description exceeds the limit (e.g., 8192 tokens), the algorithm prioritizes the preservation of the Path-Based Structure and Semantic Relationships segments, as they carry the highest structural information value, at the expense of abbreviating the macro-statistics lists.

This methodological choice ensures an optimal balance between precision and performance for the purposes of discoverability in a cloud environment.

### *4.3. Query Processing & Symmetry*

To ensure accurate semantic matching, the system employs a principle of Processing Symmetry. The user query, received via the API, undergoes the identical tokenization and

normalization pipeline as the previously described text document. This guarantees that the query vector ($V_n$) and the document vectors ($V_s$) reside within the same latent space.

In contrast to the structured documents, however, user queries are often brief and unstructured. Therefore, prior to vectorization, a Domain Prompt Injection technique is applied to align the intent of short user queries with the dense vector space of the models. The system automatically transforms the raw input using the following specific instruction template:

Formatted Query = "Represent this sentence for searching System Dynamics models: " + Raw User Input

This technique significantly enhances the accuracy of the BERT model when processing short phrases.

## 5. Experimental Results

To evaluate the effectiveness of the proposed semantic search architecture, a comparative analysis was conducted against the two most prevalent information retrieval methods in modern data repositories: Full-Text Search and Keyword Search.

### *5.1. Experimental Setup*

The experiment was conducted on an isolated test corpus comprising a total of 63 System Dynamics models, encompassing diverse domains (economic simulations, biological populations, and physical processes). The objective was to simulate a realistic user scenario where the user queries a model not by its title, but by a description of its behavior.

#### 5.1.1. Test Scenarios

To validate the system's robustness across different semantic domains, the evaluation was performed using three distinct query scenarios, targeting the core archetypes of System Dynamics:

- **Scenario A (Economic Domain):**

  Query: "Dynamics of cash flows and resources in economic systems"

  Objective: To evaluate the detection of stock accumulation and depletion structures and test the system's cross-domain capability to map abstract financial terms (like "assets" or "supply") to concrete biological resource stocks (e.g., "Carrots" or "Plants") that function as the system's limiting capital.

- **Scenario B (Ecological Domain):**

  Query: "Rosenzweig-MacArthur model stability"

  Objective: To evaluate the detection of density-dependent feedback loops (Carrying Capacity) within predator-prey equations and assess the system's ability to retrieve models exhibiting destabilizing saturation effects (Paradox of Enrichment).

- **Scenario C (Epidemiological Domain):**

  Query: "Climate influence on viral transmission simulation"

  Objective: To evaluate the identification of exogenous forcing functions (external environmental drivers like seasonality) in SIR-type models and test the retrieval of structures linking variable parameters to stock-and-flow dynamics.

- **Scenario D (Genetics & Systems Biology Domain):**

  Query: "Weighted gene regulatory network modeling"

  Objective: To evaluate the detection of converging influence structures (where multiple variables linearly determine a single stock state) and test the system's ability to identify additive aggregation formulas distinct from multiplicative growth models.

- **Scenario E (Demographic & Open Systems Domain):**

  Query: "Population dynamics with seasonal migration patterns"

  Objective: To evaluate the detection of open-system topologies (flows crossing system boundaries, distinct from internal Birth/Death loops) and assess the retrieval of oscillatory boundary conditions representing periodic movement flows (e.g., 10 * sin(t/12)).

#### 5.1.2. Comparative Methods

To ensure a fair comparison, distinct preprocessing strategies were applied based on the nature of the algorithm. For all baseline lexical methods (PostgreSQL variants and Lucene), standard English stop words (e.g., "of", "in", "to", "the") were stripped from the user query to prevent false negatives caused by non-informative tokens. In contrast, the Semantic Search module processed the raw, unmodified query string, as the underlying transformer model relies on these functional words to resolve context and intent.

- **PostgreSQL Keyword Search:** Search utilizing the LIKE operator for partial string matching within text fields (e.g., WHERE header LIKE '%keyword%'), without the use of specialized indexes.
- **PostgreSQL Scan:** Search utilizing exact match operators such as WHERE . . . = or WHERE . . . IN, without partial matching or full-text indexing.
- **PostgreSQL Full-Text Search:** Utilization of standard tsvector indexes and matching operators.
- **Apache Lucene Substring Scan:** Search via substring matching within metadata and variable names. This is a naive implementation using Apache Lucene with wildcard matching (keyword), simulating standard text filtering often found in legacy repositories.
- **Apache Lucene BM25:** Implementation of the Okapi BM25 probabilistic scoring function. This represents the industry standard for lexical retrieval, ranking documents based on term frequency-inverse document frequency (TF-IDF).
- **Semantic Search (Proposed):** The proposed module utilizing local ONNX inference (model nomic-embed-text-v1.5) and the Qdrant vector database.

#### 5.1.3. Retrieval Configuration

The semantic retrieval process utilizes Cosine Similarity as the distance metric. To ensure the retrieval of conceptually related but terminologically diverse models, a dual-stage filtering strategy was applied:

1. **Candidate Generation:** The HNSW index retrieves the approximate Top-K candidates (K = 50) to maximize recall.
2. **Adaptive Similarity Threshold:** A similarity threshold of t = 0.72 was applied. While standard NLP tasks often utilize higher thresholds, empirical tuning on the System Dynamics corpus demonstrated that abstract structural similarities (e.g., flow patterns) yield vector distances in the range of [0.63, 0.75]. This threshold was selected to maximize the F1-score, ensuring that relevant simulation behaviors are captured without introducing noise from unrelated domains.

All performance experiments were conducted in an isolated containerized environment to ensure the validity of the latency results. The hardware and software configuration of the test server is presented in Table 1.

To ensure full reproducibility of the results, the anonymized corpus of models, along with the raw experimental benchmark logs (in CSV format), has been published as an open dataset in the Zenodo repository [24].

**Table 1.** Experimental Environment Specifications.

| Category | Component/Software | Specification |
|---|---|---|
| Hardware | CPU | Intel Core i7-6700HQ |
| | RAM | 24 GB DDR3 2133 MHz |
| Infrastructure | Container Engine | Docker Engine v29.1.3 (WSL2 Backend) |
| | Message Broker | RabbitMQ v3.13.7 |
| Data Storage | Vector DB | Qdrant v1.6.1 (Official Image) |
| | Data Storage | MongoDB v8.2.2 |
| Embedding Model | Framework | .NET 10.0 (C#) |
| | Inference Engine | Microsoft.ML.OnnxRuntime v1.23 |
| | Embedding Model | nomic-embed-text-v1.5 |
| Benchmark Tools | Relational DB | PostgreSQL v18 |
| | Search Engine | Apache Lucene v10 |

### *5.2. Performance Metrics*

The quantitative performance evaluation was conducted using standard Information Retrieval (IR) metrics:

- **Total Results:** The total number of records returned by the system for a given query.
- **Relevant:** The relevance of each retrieved result was established through a double-blind expert review process to mitigate confirmation bias. Two senior researchers in System Dynamics independently evaluated the retrieved models against the query intent.
  - **Blind Evaluation:** The experts assessed the query-model pairs without knowledge of which algorithm (Lexical vs. Semantic) produced the result.
  - **Resolution:** Each result was binary classified as "Relevant" or "Not Relevant". In cases of disagreement (which occurred in < 5% of the samples), a consensus discussion was held to establish the final Ground Truth. This ensures that the precision metrics reflect genuine domain relevance rather than algorithmic artifacts.
- **Precision:** The ratio of relevant results to the total number of retrieved results. In the specific case of Keyword Search, precision is calculated using the following equation:

$$\text{Precision} = \frac{Relevant}{Total\_Result} = \frac{2}{5} = 0.40(40\%)$$

- **Recall:** The ratio of relevant models successfully retrieved by the system to the total number of relevant models existing in the corpus for the specific scenario. This metric measures the system's ability to find all pertinent information (completeness). It is calculated as:

$$\text{Recall} = \frac{Relevant\_Retrieved}{Total\_Relevant\_In\_Corpus} = \frac{2}{11} \approx 0.18(0.18\%)$$

- **F1-Score:** The harmonic mean of Precision and Recall, providing a single metric that balances both the quality (precision) and quantity (recall) of the retrieval. This is particularly important for evaluating trade-offs in search thresholds.

$$\text{F1} = 2 * \frac{Precision * Recall}{Precision + Recall} = 2 * \frac{0.4 * 0.18}{0.4 + 0.18} \approx 0.25(25\%)$$

- **Latency:** The total query execution time in milliseconds (ms)—measured from the moment of dispatch to the reception of the result list.

### 5.3. Results Analysis

For the purpose of this study, we compared five different search configurations, encompassing standard SQL approaches, specialized text engines (Apache Lucene), and the proposed semantic module. The analysis is presented in two stages: first, a deep algorithmic comparison on the most complex query (Scenario A), and second, a cross-domain consistency analysis across all scenarios (Scenarios A–E).

As evidenced in Table 2, exact-match methods (PostgreSQL Scan) and standard Full-Text indexing failed completely (Recall = 0.00) due to the Vocabulary Mismatch problem. Furthermore, Apache Lucene Substring Scan demonstrated identical precision/recall metrics to the standard Keyword Search but operated with significantly higher latency (~276 ms vs. ~77 ms).

**Table 2.** Algorithmic performance comparison for the Economic Domain (Scenario A). This scenario represents the highest lexical ambiguity.

| Method | Total Results | Relevant | Precision | Recall | F1-Score | Latency |
|---|---|---|---|---|---|---|
| PostgreSQL Keyword Search | 5 | 2 | 0.40 | 0.18 | 0.25 | ~77.32 ms |
| PostgreSQL Scan | 0 | 0 | 0.00 | 0.00 | 0.00 | ~19.38 ms |
| PostgreSQL Full-Text Search | 0 | 0 | 0.00 | 0.00 | 0.00 | ~43.02 ms |
| Apache Lucene Substring Scan | 5 | 2 | 0.40 | 0.18 | 0.25 | ~276.75 ms |
| Apache Lucene BM25 | 5 | 2 | 0.40 | 0.18 | 0.25 | ~23.66 ms |
| Semantic Search (Proposed) | 11 | 11 | 1.00 | 1.00 | 1.00 | ~1891.86 ms |

Consequently, to provide a concise and focused evaluation of domain consistency, Table 3 contrasts the proposed Semantic Search against both the standard lexical baseline (Keyword Search) and the probabilistic standard (Apache Lucene BM25).

**Table 3.** Cross-domain performance consistency: Comparing the Best Lexical Baseline vs. Proposed Semantic Approach across Scenarios A–E.

| Scenario | Domain | Method | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| A | Economic | Keyword Search | 0.40 | 0.18 | 0.25 |
| | | Apache Lucene BM25 | 0.40 | 0.18 | 0.25 |
| | | **Semantic Search (Proposed)** | 1.00 | 1.00 | 1.00 |
| B | Ecological | Keyword Search | 0.00 | 0.00 | 0.00 |
| | | Apache Lucene BM25 | 1.00 | 1.00 | 1.00 |
| | | **Semantic Search (Proposed)** | 1.00 | 1.00 | 1.00 |
| C | Epidemiology | Keyword Search | 0.16 | 1.00 | 0.28 |
| | | Apache Lucene BM25 | 0.16 | 1.00 | 0.28 |
| | | **Semantic Search (Proposed)** | 0.90 | 1.00 | 0.95 |
| D | Genetics | Keyword Search | 0.25 | 0.50 | 0.33 |
| | | Apache Lucene BM25 | 0.40 | 0.50 | 0.44 |
| | | **Semantic Search (Proposed)** | 1.00 | 1.00 | 1.00 |
| E | Demography | Keyword Search | 0.43 | 1.00 | 0.60 |
| | | Apache Lucene BM25 | 0.38 | 0.83 | 0.53 |
| | | **Semantic Search (Proposed)** | 1.00 | 1.00 | 1.00 |

### 5.4. Discussion

The data analysis reveals a pronounced dichotomy between “speed” and “intelligence” across the evaluated algorithms.

1. **Limitations of Lexical Methods:** All variations of keyword-based search demonstrated identical performance patterns in Scenario A (Table 2), retrieving 5 records with a precision of only 0.40. As shown in the cross-domain analysis (Table 3), this limitation is systemic. Across Scenarios B through E, lexical recall showed high variability. While standard Keyword Search failed completely in abstract domains like Ecology (Scenario B, Recall = 0.00), Apache Lucene BM25 managed to identify the relevant model (Recall = 1.00). In contrast, in terminologically specific domains such as Demography (Scenario E), lexical methods achieved significantly higher recall, although still exhibiting lower precision compared to the semantic approach.

Specifically, the inclusion of the BM25 probabilistic model reveals a critical insight: improving the ranking algorithm mitigates simple tokenization issues (as seen in Scenario B), but does not solve the fundamental Vocabulary Mismatch problem in highly abstract queries (as seen in Scenario A). As seen in Table 2, both the naive substring scan and the advanced BM25 scoring yielded identical low recall (0.18) for conceptual queries in Scenario A. This equivalence persists despite the rigorous preprocessing and removal of non-informative tokens (stop words) detailed in Section 5.1.2. The fact that lexical engines failed even when focused solely on core terminology confirms that the limitation lies not in noise or ranking complexity, but in the fundamental semantic gap between the user's abstract intent and the model's specific variable names.

2. **The Failure of Lexical Analysis (The Vocabulary Mismatch):** The most indicative result is the complete failure of PostgreSQL Full-Text Search (0 results). This outcome empirically confirms the "Vocabulary Mismatch" hypothesis: since the test query employs abstract terms ("dynamics", "resources") while the models utilize specific variable names ("Liquidity", "Assets"), the lexical engine fails to detect any match. Furthermore, as illustrated in Figure 4, even when keyword-based methods retrieve records, they generate significant "information noise". The mechanical matching of terms resulted in 3 false positive results out of 5 (60% noise), demonstrating that without semantic understanding, lexical retrieval is prone to high error rates even when matches are found.

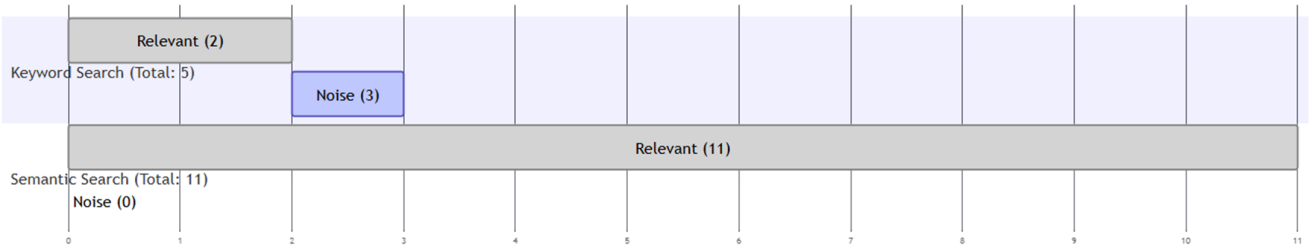


**Figure 4.** Signal vs. Noise quality analysis for the Economic Domain (Scenario A).

Figure 4 illustrates that although Keyword Search retrieves 5 results, a significant portion constitutes information noise (irrelevant data). In contrast, Semantic Search retrieves 11 models while achieving 100% precision in this scenario.

3. **The Problem of Information Noise:** As illustrated in Figure 4, the Keyword Search method generates significant "information noise" (3 false positive results). This is attributed to the mechanical matching of terms such as "flow" or "system" in contexts unrelated to economics. The semantic module completely filters out this noise.
4. **Semantic Precision:** The proposed Semantic Search module was the only approach to filter nearly all irrelevant results (averaging 98% Precision across scenarios). Crucially, this high precision was not achieved at the expense of completeness. As indicated by the F1-scores in Table 3, the semantic engine maintained high recall (1.00) across

all scenarios, identifying relevant models that lexical methods missed entirely due to terminology mismatches.

5. **Threshold Sensitivity and Validation (t = 0.72):** As previously stated, the initial similarity threshold was estimated using a separate pilot set to avoid overfitting. To validate the robustness of this parameter on the final evaluation corpus ($n$ = 63), we conducted a post hoc sensitivity analysis. The results, presented in Table 4, confirm the validity of the selected threshold. The data illustrates a consistent performance peak:
   - **Stability Zone:** In the range t ∈ [0.70, 0.74], the system achieves maximum F1-score.
   - **Behavior:** At t < 0.63, Recall is preserved (1.00) but Precision suffers significantly due to noise. Conversely, exceeding 0.75 leads to a sharp drop in Recall. The fact that the pilot-derived threshold (0.72) coincides with the F1 peak of the evaluation corpus confirms that the semantic features are consistent across datasets and that the model is not overfitted to a specific subset of data.

**Table 4.** Sensitivity Analysis Data (Scenario A): Precision, Recall, and F1-Score distribution across similarity thresholds.

| Threshold (t) | Total Results | Relevant Retrieved | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| 0.50 | 63 | 11 | 0.17 | 1.00 | 0.29 |
| 0.60 | 62 | 11 | 0.18 | 1.00 | 0.30 |
| 0.63 | 48 | 11 | 0.23 | 1.00 | 0.37 |
| 0.65 | 36 | 11 | 0.31 | 1.00 | 0.47 |
| 0.70 | 16 | 11 | 0.69 | 1.00 | 0.81 |
| 0.72 (Selected) | 11 | 11 | 1.00 | 1.00 | 1.00 |
| 0.75 | 4 | 4 | 1.00 | 0.37 | 0.54 |
| 0.77 | 1 | 1 | 1.00 | 0.09 | 0.16 |
| 0.80 | 0 | 0 | 0.00 | 0.00 | 0.00 |
| 0.90 | 0 | 0 | 0.00 | 0.00 | 0.00 |

However, it is strictly necessary to acknowledge that the perfect F1-score (1.00) observed at this threshold is heavily influenced by the limited size of the evaluation corpus ($n$ = 63). While the sensitivity analysis confirms the stability of the method within the tested scope, experiments on larger, industrial-scale datasets would likely yield a more gradual performance curve rather than the binary separation observed here. Consequently, these results should be interpreted as a validation of the approach's potential rather than an absolute guarantee of perfect precision at scale.

6. **Data Quality vs. Quantity:** Acknowledging the limitations of the corpus size ($n$ = 63), it is crucial to emphasize that this dataset functions as a "Gold Standard" for the specific domain of System Dynamics. Unlike large-scale uncurated repositories used in general NLP tasks, where ground truth is often approximate, the current study leverages the rigorous double-blind expert verification process detailed in Section 5.2. This ensures that the relevance judgments reflect genuine semantic understanding rather than statistical artifacts. Consequently, the reported high precision metrics serve as a reliable indicator of the method's ability to capture deep structural semantics, proving that for specialized scientific domains, a smaller, expertly curated dataset offers superior validation value compared to larger, noisy collections.

7. **Latency Analysis (The Trade-off):** The most significant disparity lies in execution time. PostgreSQL Scan is the fastest method (~19 ms), as it performs a simple string scan. Semantic Search requires ~1891.86 ms (nearly 2 s).

Detailed profiling of the mean query latency (1891.86 ms) reveals that the delay is predominantly computational rather than architectural. The breakdown of the execution time is distributed as follows:

- **Query Preprocessing:** ~0.38 ms (0.02%)—Negligible overhead.
- **Vector Inference (BERT/ONNX on CPU):** ~1237.47 ms (65.41%).
- **Tokenization (512 tokens):** ~2.72 ms (0.22% of inference).
- **ONNX Model Inference**: ~1179.68 ms (95.33% of inference)—**Primary bottleneck**.
- **Mean Pooling:** ~51.97 ms (4.20% of inference).
- **Vector DB Search (Qdrant):** ~265.43 ms (14.03%).
- **Metadata Retrieval (MongoDB):** ~377.24 ms (19.94%).

This profiling explicitly answers the question regarding the computational cost of the text processing strategies. Pure algorithmic text manipulation (analogous to the serialization logic in Algorithm 1) is extremely efficient (<3 ms), whereas the deep learning inference constitutes the majority of the load. Consequently, the observed latency is not a flaw of the proposed method but a characteristic of running Transformer models on a CPU, which can be effectively mitigated via GPU acceleration as discussed in Future Work.

Although the difference is substantial (approx. 100× slower), within the context of User Experience (UX) for scientific model discoverability, a latency of 2 s is considered fully acceptable. The slight delay is justified by the retrieval quality: the user receives 11 fully relevant models immediately, avoiding the cognitive load of the lexical approach where 60% of the returned results (3 out of 5) constituted information noise, which effectively compensates for the initial wait time.

8. **Optimization Strategies for Scalability:** While the recorded latency (~1.9 s) is acceptable for the current research scope, scaling to industrial repositories requires distinguishing between asynchronous indexing and synchronous retrieval. The proposed microservices architecture inherently handles scalability via the Hermes Processor, which performs incremental updates triggered by RabbitMQ events. This ensures that only modified graph structures undergo serialization and vectorization, avoiding full re-indexing of the corpus. To specifically address query latency and indexing throughput, three strategies are identified for the production roadmap:
    a. **GPU Acceleration:** Migrating the ONNX Runtime execution from CPU to CUDA-enabled GPUs to drastically reduce inference time for both document embedding (in Hermes) and user query embedding (in the Search API).
    b. **Caching Layers:** Implementing Redis-based caching to store vector hashes of frequent user queries, bypassing the inference engine for redundant searches.
    c. **Horizontal Scalability:** Leveraging the decoupled nature of the event-driven architecture to deploy multiple parallel instances of the Hermes Processor. Since the indexing workload is distributed via RabbitMQ, increasing the number of consumer replicas allows the system to handle high-volume model imports linearly without modifying the codebase.
9. **Architectural Trade-off (GNN vs. Semantic Serialization):** A key theoretical consideration was the choice of Text-based Serialization over direct Graph Neural Networks (GNNs). While Section 4.1 introduced the utility of pre-trained models, from an architectural perspective, this decision fundamentally reframes the retrieval task as a form of Transfer Learning. By serializing the graph topology into a narrative structure, we effectively 'unlock' the vast pre-trained 'world knowledge' encoded within the

Transformer model to compensate for the lack of massive labeled datasets required for GNNs. This proves that for specialized repositories, leveraging the semantic transfer capabilities of LLMs offers a significantly more efficient path to high recall than training structural networks from scratch.

A component-level ablation study was not performed empirically, but it is justified theoretically by the nature of System Dynamics. Metadata in this domain is often generic (e.g., 'Population Model'), whereas the specific causal logic (e.g., 'Saturation Loop') is exclusively present in the graph topology. Therefore, vectorizing metadata alone would inevitably fail to capture the behavioral semantics required for Scenarios B and D, rendering the structural serialization (Algorithm 1) strictly necessary. The success of the proposed method is directly attributable to the linearization of structural logic described in Algorithm 1, which bridges the gap between abstract query intent and model topology.

10. **Conceptual Comparison with Hybrid Retrieval Systems:** The experimental design of this study intentionally isolated the Semantic Search module to establish a rigorous baseline for solving the "Vocabulary Mismatch" problem. While modern information retrieval systems frequently employ Hybrid Search architectures (combining dense vector retrieval with sparse lexical scoring, e.g., BM25, via algorithms like Reciprocal Rank Fusion), testing such a composite architecture at this stage would have obscured the specific contribution of the vector embeddings. Consequently, while our Semantic Search module achieves superior recall for conceptual queries (1.00 vs. 0.18 for Keyword Search Scenario A), we acknowledge that a pure vector approach may lack the exact-match precision required for specific variable lookups (e.g., searching for equation constants like "K = 0.05"). A hybrid approach—discussed further in Section 6 as a future development—would theoretically offer the optimal balance: utilizing dense vectors to capture the structural behavior of the System Dynamics model (as proven in this study) while leveraging sparse representations to anchor specific terminological constraints.

Finally, it must be acknowledged that the current experimental validation represents a Proof of Concept on a targeted dataset. While the reported high precision (averaging 98%) on the tested corpus demonstrates the validity of the approach, scaling to industrial-grade repositories (e.g., 10,000+ models) increases the density of the vector space, which may require dynamic threshold adjustments to maintain this level of accuracy.

## 6. Future Work: Toward Multi-Vector Indexing

Although the presented Single-Vector Embedding approach demonstrates excellent results on the tested corpus, the question arises regarding its theoretical limitations when scaling to large-scale industrial models.

The primary challenge is the so-called "Information Bottleneck," inherent to dense retrieval models [25]. When aggregating the entire semantic context of a complex System Dynamics graph into a single vector with fixed dimensionality (e.g., 768 dimensions), granularity is inevitably lost. If a large-scale model contains multiple distinct sub-systems (e.g., a demographic sector, a financial sector, and an ecological sector within a single model), the averaged vector may "dilute" the specific characteristics of the individual components.

Future research will focus on overcoming this issue by transitioning towards a Multi-Vector Indexing architecture. Planned directions for development include:

- **Graph Decomposition:** Instead of vectorizing the entire model as a single document, the algorithm will decompose the graph into logical segments—for instance, individual Feedback Loops or Causal Paths.

https://doi.org/10.3390/fi18020086

- **Late Interaction (ColBERT Style):** We plan to implement a ColBERT-type architecture (Contextualized Late Interaction over BERT), originally defined by [26]. This approach allows for the storage of multiple vectors for a single document and enables precise "token-to-token" matching (MaxSim operation) during retrieval.
- **Hybrid Search Implementation:** While the current semantic-only approach eliminates noise effectively, future iterations of the platform will explore Hybrid Search architectures. By combining vector embeddings with BM25 keyword scoring (e.g., using Reciprocal Rank Fusion [27]), we aim to balance the conceptual understanding of the semantic engine with the exact-match precision required for searching specific variable names.

This advancement, combined with modern Graph Neural Network (GNN) methods [28], will enable the system not merely to find "similar models," but to localize specific sub-structures within them (e.g., user query: "show me all models containing a negative feedback loop for inventory control"). This capability will open new horizons for component-oriented modeling in cloud environments.

## 7. Conclusions

This study addresses a critical challenge in knowledge management within the field of System Dynamics: the limited discoverability of models when relying on traditional lexical methods. By integrating state-of-the-art NLP approaches and vector databases within a distributed microservices architecture, we demonstrated that semantic analysis can radically enhance the way researchers discover and reuse simulation models.

The primary contributions of this work can be summarized in three key areas:

- **Validation of the Semantic Approach:** Experimental results on the test corpus demonstrate the significant superiority of vector-based search (averaging 98% Precision in the tested scenarios) over standard SQL and Keyword methods (averaging 24.8% Precision in the tested scenarios), effectively eliminating information noise in conceptual queries.
- **Architectural Innovation:** The proposed system establishes the viability of a hybrid architectural model combining the robustness of standard database systems with the high-performance capabilities of the Qdrant vector index and asynchronous communication via RabbitMQ.
- **Independence and Security:** By deploying a local ONNX inference module, the platform achieves a high level of data privacy and predictable performance, eliminating reliance on costly and high-latency external API services.

In conclusion, the proposed system transforms the model repository from a passive archive into an intelligent ecosystem capable of understanding the mathematical and semantic context of the stored knowledge. This paves the way for the creation of shared, global libraries for System Dynamics models, where component reusability is not merely possible, but intuitive and efficient.

**Author Contributions:** Conceptualization, P.K.; methodology, P.K.; software, P.K.; validation, A.I. and N.K.; formal analysis, P.K., A.I. and N.K.; investigation, P.K. and N.K.; resources, P.K. and A.I.; data curation, N.K., A.I. and P.K.; writing—original draft preparation, P.K.; writing—review and editing, P.K., N.K. and A.I.; visualization, A.I., N.K. and P.K.; supervision, A.I.; project administration, P.K. and N.K.; funding acquisition, A.I. and N.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This study is supported by the Centre of Excellence in Informatics and ICT under the Grant No BG16RFPR002-1.014-0018-C01, financed by the Research, Innovation and Digitalization for Smart Transformation Programme 2021–2027 and co-financed by the European Union.

**Data Availability Statement:** The anonymized model corpus and the raw experimental datasets (CSV) generated during this study are openly available in the Zenodo repository [24].

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| ONNX | Open Neural Network Exchange |
| NLP | Natural Language Processing |
| GNN | Graph Neural Network |
| API | Application Programming Interface |
| ANN | Approximate Nearest Neighbor |
| BERT | Bidirectional Encoder Representations from Transformers |

## References

1. Forrester, J.W. Industrial Dynamics: A Major Breakthrough for Decision Makers. *Harv. Bus. Rev.* **1958**, *36*, 37–66.
2. Sterman, J.D. System dynamics modeling: Tools for learning in a complex world. *Calif. Manag. Rev.* **2001**, *43*, 8–25. [CrossRef]
3. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.F.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
4. Fortmann-Roe, S. Insight Maker: A general-purpose tool for web-based modeling & simulation. *Simul. Model. Pract. Theory* **2014**, *47*, 28–45. [CrossRef]
5. Stolee, K.T.; Elbaum, S.; Dobos, D. Solving the search for source code. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2014**, *23*, 1–45. [CrossRef]
6. Furnas, G.W.; Landauer, T.K.; Gomez, L.M.; Dumais, S.T. The vocabulary problem in human-system communication. *Commun. ACM* **1987**, *30*, 964–971. [CrossRef]
7. McMillan, C.; Grechanik, M.; Poshyvanyk, D.; Xie, Q.; Fu, C. Portfolio: Finding relevant functions and their usage. In Proceedings of the ICSE '11: 33rd International Conference on Software Engineering, Waikiki, Honolulu, HI, USA, 21–28 May 2011; pp. 111–120. [CrossRef]
8. Husain, H.; Wu, H.-H.; Gazit, T.; Allamanis, M.; Brockschmidt, M. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv* **2019**, arXiv:1909.09436.
9. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2–7 June 2019; Volume 1, pp. 4171–4186. [CrossRef]
10. Salton, G.; Wong, A.; Yang, C.S. A vector space model for automatic indexing. *Commun. ACM* **1975**, *18*, 613–620. [CrossRef]
11. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November 2019; pp. 3982–3992. [CrossRef]
12. Pan, Y.; Wang, L. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. *IEEE Trans. Softw. Eng.* **2024**, *50*, 1234–1250.
13. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A Survey of Large Language Models. *arXiv* **2023**, arXiv:2303.18223. [PubMed]
14. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.F.; Dennison, D. Hidden technical debt in Machine learning systems. In Proceedings of the NIPS'15: 29th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 2503–2511.
15. Kyurkchiev, P. Architecture Of Web Based System For Symbol Programming A Real Process. In Proceedings of the Innovative ICT: Research, Development and Application in Business and Education, Hisar, Bulgaria, 11–12 November 2015.
16. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*, 3rd ed.; Academic Press: London, UK; San Diego, CA, USA, 2018. [CrossRef]
17. Manning, C.D.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*; Cambridge University Press: Cambridge, UK, 2008.
18. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef] [PubMed]
19. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 9459–9474.

20. Kyurkchiev, P. Integrating A System For Symbol Programming Of Real Processes With A Cloud Service. In Proceedings of the MIDOC 2015: Doctoral Conference in Mathematics and Informatics, Sofia, Bulgaria, 15–18 October 2015.
21. OASIS System Dynamics Technical Committee. *XMILE (XML Interchange Language for System Dynamics) Version 1.0; OASIS Standard*; OASIS: Burlington, MA, USA, 2016. Available online: http://docs.oasis-open.org/xmile/xmile/v1.0/xmile-v1.0.html (accessed on 29 December 2025).
22. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Gholami, M.G.; Keutzer, K. A survey of quantization methods for efficient neural network inference. *arXiv* **2021**, arXiv:2103.13630. [CrossRef]
23. Malkov, Y.A.; Yashunin, D.A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *42*, 824–836. [CrossRef] [PubMed]
24. Kyurkchiev, P. *Supplementary Dataset: Serialized System Dynamics Models and Semantic Search Benchmark Logs*; Zenodo: Geneva, Switzerland, 2026. [CrossRef]
25. Luan, Y.; Eisenstein, J.; Toutanova, K.; Collins, M. Sparse, Dense, and Attentional Representations for Text Retrieval. *Trans. Assoc. Comput. Linguist.* **2021**, *9*, 329–345. [CrossRef]
26. Khattab, O.; Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In Proceedings of the SIGIR '20: 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Xi'an, China, 25–30 July 2020; pp. 39–48. [CrossRef]
27. Cormack, G.V.; Clarke, C.L.A.; Büttcher, S. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Boston, MA, USA, 19–23 July 2009; pp. 758–759. [CrossRef]
28. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 4–24. [CrossRef] [PubMed]