# Planning Agents on an Ego-Trip: Leveraging Hybrid Ego-Graph Ensembles for Improved Tool Retrieval in Enterprise Task Planning

**Sahil Bansal***, **Sai Shruthi Sistla***, **Aarti Arikatala, Sebastian Schreiber**
SAP Labs, Palo Alto, CA, USA
{sahil.bansal01, sai.shruthi.sistla
aarti.arikatala, sebastian.schreiber}@sap.com

## Abstract

Effective tool pre-selection via retrieval is essential for AI agents to select from a vast array of tools when identifying and planning actions in the context of complex user queries. Despite its central role in planning, this aspect remains underexplored in the literature. Traditional approaches rely primarily on similarities between user queries and tool descriptions, which significantly limits retrieval accuracy, specifically when handling multi-step user requests. To address these limitations, we propose a Knowledge Graph (KG)-based tool retrieval framework that captures the semantic relationships between tools and their functional dependencies. Our retrieval algorithm leverages ensembles of 1-hop ego tool graphs to model direct and indirect connections between tools, enabling more comprehensive and contextual tool selection for multi-step tasks. We evaluate our approach on a synthetically generated internal dataset across six defined user classes, extending previous work on coherent dialogue synthesis and tool retrieval benchmarks. Results demonstrate that our tool graph-based method achieves $91.85\%$ tool coverage on the micro-average *CompleteRecall* metric, compared to $89.26\%$ for re-ranked semantic-lexical hybrid retrieval, the strongest non-KG baseline in our experiments. These findings support our hypothesis that the structural information modeled in the graph provides complementary signals to pure similarity matching, particularly for queries requiring sequential tool composition.

## 1 Introduction

Agentic systems powered by Large Language (LLMs) or Reasoning Models (LRMs) excel in planning and scheduling sub-tasks for complex requests (Kim et al., 2024; Erdogan et al., 2025; Rawat et al., 2025). While these systems effectively break down tasks into manageable logical sequences, evaluations have primarily focused on controlled settings with limited, well-defined tools that fit within a model's context window, such as web search, a calculator, etc.

Enterprise environments present greater challenges, with organizations relying on thousands of specialized tools with complex, often undocumented interdependencies. Conventional retrieval methods, particularly vector-based similarity search, frequently miss relevant tools, resulting in fragmented execution strategies. This limitation is especially critical in general-purpose agentic planning systems, where both initial and ongoing tool discovery form the foundation for effective task decomposition and execution.

Moreover, effective tool discovery is an essential prerequisite for meaningful task decomposition: Agents must first identify what capabilities are available before they can decide how to break down and solve a complex problem. Despite its centrality to real-world agentic long-horizon planning, this aspect remains underexplored in the existing literature (Huang et al., 2024; Wei et al., 2025).

We propose a structured semantic representation of enterprise tools using semi-structured data from tool descriptions and metadata. This approach produces a knowledge graph (KG) capturing relationships between tools, entities, and parameters, enabling better mapping of user queries to relevant tools. Our KG-enhanced retrieval mechanism uses neighborhood expansion to uncover implicit connections that traditional retrieval methods miss.

Our contributions are fourfold:

1. We propose a method to extract and model tool dependencies, facilitating tool trajectory discovery when explicit dependencies are missing.
2. We introduce the *Ensemble of Ego Graphs (EEG)* algorithm, which uses an ensemble

*The authors contributed equally to this work.



*Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 903–918
December 20-24, 2025 

of 1-hop ego tool graphs extracted from our overall tool graph via a hybrid node matching and neighborhood expansion technique to improve tool retrieval performance.

3. Motivated by an internal analysis of enterprise user queries, we define six distinct query classes in the context of complex user queries. We present a novel pipeline for generating multi-step, multi-intent queries aligned with these classes. Our approach leverages tool dependency analysis, based on parameter–parameter relationships and LLM-inferred return parameter graphs, to identify feasible tool chains and to ensure the generated queries are coherent, contextually relevant, and faithful to their intended class.
4. We evaluate the retrieval efficacy of our EEG algorithm on complex user queries generated using the above pipeline, using a *CompleteRecall* metric (Zhang et al., 2025) specifically adapted to our tool retrieval setup, demonstrating significant improvements over baseline approaches.

## 2 Related Work

Recent works have explored the use of tool graphs for tool retrieval but exhibit notable limitations.

For instance, ControlLLM (Liu et al., 2024b) requires an adjacency matrix to construct the tool graph, assuming its structure is defined a priori. While offering a structured approach for tool selection and execution, this reliance on pre-specified connections limits its applicability in automation scenarios where tool relationships are unknown or evolving.

ToolNet (Liu et al., 2024a) employs graph-based iterative tool traversal similar to our approach. However, its graph construction depends on either extensive tool-use trajectories from code repositories and public datasets (unavailable for enterprise APIs) or LLM-generated trajectories, which frequently contain errors.

COLT (Qu et al., 2024) employs a complex multi-step, multi-bipartite graph training process for transductive graph embeddings, but lacks automatic scene inference and clear application paths for novel queries, limiting its generalization capabilities.

Tool Graph Retriever (Anonymous, 2024) extracts tool dependencies from documentation to create a graph. Unlike their approach, our method doesn't use a custom dependency identification model but instead leverages similarities between parameters and other entities, extracted via Open Information Extraction, to connect tools in a graph.

Our approach also shares some similarities with ToolFlow (Wang et al., 2025b), which builds tool dependency graphs from documentation as well, but applies them to conversation generation rather than retrieval purposes. Building on the work presented in this paper, we further extended the methodology to synthetically generate complex, multi-step business queries in scenarios where output parameters are not available. This extension enables a more rigorous evaluation of the proposed graph-based retrieval mechanism.

Graph RAG-Tool Fusion (Lumer et al., 2025), developed concurrently with our work, similarly combines vector retrieval with knowledge graph traversal but differs in two key ways: First, unlike their method that relies on synthetic tool graphs with well-defined dependencies suited to depth-first search, our approach semi-automatically converts real enterprise tools into graph representations, requiring only minimal manual input to tune prompts for accurate LLM interpretation of tool metadata and to define domain-specific ontologies and entity types that ensure semantic consistency during graph construction. Second, we identify ego-graph entry nodes using multiple vector representations rather than limiting ourselves to only semantic embeddings, making our method more practical for enterprise environments.

Another related approach incorporates tool knowledge directly into model parameters through training or fine-tuning, including multi-label classifiers (Moon et al., 2024) and LLMs (Wang et al., 2025a). However, these parameter-based methods are inadequate for dynamic enterprise environments with large, frequently changing tool ecosystems. For a comprehensive review of such approaches and their limitations, we refer readers to recent surveys like (Qu et al., 2025).

In summary, our approach combines automatic tool graph construction with multi-vector graph-retrieval mechanisms in a novel way, offering superior adaptability to dynamic enterprise environments compared to existing methods that either rely on synthetic graphs, lack contextual understanding, or cannot scale with frequently changing tool ecosystems.

## 3 Methodology

### 3.1 System Overview

Our proposed pipeline for tool retrieval is structured into two principal stages: an offline phase for building a structured Knowledge Graph (KG) from semi-structured tool documentation and metadata, and an online phase for retrieving relevant tools in response to user requests.

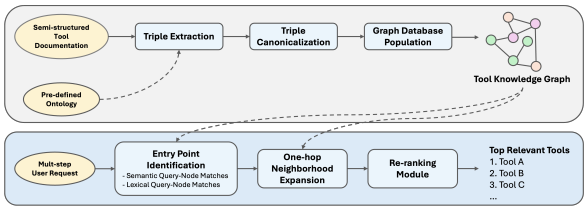


Figure 1: Tool Construction and Retrieval Pipeline

As illustrated in Figure 1, the offline phase begins with the ingestion of semi-structured tool documentation, which is processed by a triple extraction module to identify the relevant relational triples. Since naive extraction can result in an explosion of irrelevant or noisy triples, we guide the process using a pre-defined ontology that constrains and informs the extraction. The extracted triples are then passed through a triple canonicalization stage to ensure structural consistency and remove redundancies. The resulting canonical triples are used to populate a graph database, which instantiates the KG used for retrieval.

In the online phase, when a user query is received, our KG-based retrieval algorithm combines semantic query-tool node matching and textual query-entity node matching to identify entry points in the tool graph. One-hop neighborhood expansion then enriches the candidate tool set, followed by re-ranking to return the most relevant tools. The following subsections provide more details on these steps.

### 3.2 Knowledge Graph Construction

Table 1 introduces an example tool and its metadata, which we use as a running example throughout the methodology section.

To facilitate accurate and scalable retrieval of enterprise tools and their metadata, we construct a KG that captures both semantic and relational structures. This section details the four core components of our KG construction pipeline: ontology definition, triple extraction, triple canonicalization, and graph population. Our full graph construction algorithm in pseudo code is presented in Algorithm 1.

#### 3.2.1 Ontology Definition

To guide the extraction process and maintain a focused, manageable graph structure, we employ a predefined ontology tailored to our domain. This ontology defines:

- Entity Types: A restricted set of meaningful categories such as *tool*, *parameter*, *line of business*, *business object*, *capability*, and *department*.
- Predicate Types: A curated list of relationships (e.g., *has_parameter, used_by, assigned_to, related_to, contains*) that are relevant for retrieval and downstream reasoning tasks.

These types are directly aligned with the structure of enterprise task management tools. For instance, for the sample tool from Table 1, we derive:

- Entity:
  - *type*: `tool`
  - *name*: `Send Task Deadline Reminder to Team Members`
- Predicate-object pairs, such as:
  - *has_parameter*: `priority_filter`
  - *has_department*: `operations`

By constraining both entity and relation types, we reduce graph noise, prevent combinatorial growth, and ensure that extracted triples directly support the system's use-cases. The prompt which provides this guidance to the triple extraction LLM is shared in Figure 6.

#### 3.2.2 Triple Extraction

We extract structured semantic knowledge from semi-structured text sources by identifying relational triples of the form *(subject, predicate, object)*. Each triple encodes a factual assertion about an entity and its relationship to another entity.

- Subject: The source entity (e.g., a *tool*).
- Predicate: The relationship (e.g., *has_parameter*).
- Object: The target entity or value (e.g., *deadline_range*).

To automate this process at scale, we leverage `GPT-4o`[1] for natural language understanding and

[1] https://platform.openai.com/docs/models/gpt-4o

| | | | |
|---|---|---|---|
| **Tool Title** | Send Task Deadline Reminder to Team Members | | |
| **Description** | Sends an automated email reminder to team members responsible for tasks nearing their deadline. | | |
| **Parameters** | **Name** | **Description** | **Type** |
| | priority_filter | Filters for high-priority tasks requiring immediate attention. | `enum [High, Medium, Low]` |
| | recipient_list | Email addresses of team members assigned to tasks, extracted from *task_list_upcoming_deadlines*. | `list<user_email>` |

Table 1: Sample tool with its metadata.

relation extraction. The model is prompted with domain-specific instructions to identify and extract accurate and relevant triples from textual descriptions. For example, for the tool in Table 1, we could extract the following triples:

- (`Task`, *categorized_by*, `Priority Level`)
- (`Automated Reminder`, *triggered_by*, `Task Deadline`)

While some variations in entity or relation phrasing remain, LLM-based extraction mitigates many of the weaknesses of traditional approaches (e.g., rigid pattern-matching, limited semantic generalization) as shown in recent surveys of relation extraction using large language models (Diaz-Garcia and Lopez, 2025; Xu et al., 2024). Residual inconsistencies are addressed through a triple canonicalization step, which we describe next.

#### 3.2.3 Triple Canonicalization

To enhance consistency and reduce redundancy in the KG, we perform normalization on both entities and predicates. This process involves two key steps:

- Entity Normalization: We unify different surface forms of the same entity into a single canonical representation. For instance, *"Supplier", "supplier",* and *"suppliers"* are all normalized to a single canonical node *"supplier"*.
- Predicate Normalization: Semantically equivalent predicates are consolidated under a unified relation. For example, *"works at"*, *"employed by",* and *"works for"* are normalized to a single canonical predicate *"employed_by"*.

This canonicalization step enhances graph quality by avoiding duplicate nodes and edges, which simplifies querying and downstream analysis.

[2, 3] Monospaced text denotes function names in definitions and API calls; **bold** text marks language keywords; *italics* mark variables, parameters, and arguments.

**Algorithm 1** Knowledge Graph Construction[2]

**Require:** Tool documentation and metadata $D$, ontology $O$, domain-specific Open Information Extraction prompt $P$
**Ensure:** Knowledge Graph $KG$
$KG \leftarrow$ initialize empty graph
**for all** tool_doc in $D$ **do**
  $triples \leftarrow$ EXTRACTTRIPLES($tool\_doc, O$)
  $canonical\_triples \leftarrow$ CANONICALIZE($triples$)
  **for all** $(s, p, o)$ in $canonical\_triples$ **do**
    ADDNODE($KG$, $s$, $metadata$)
    ADDNODE($KG$, $o$, $metadata$)
    ADDEDGE($KG$, $s$, $p$, $o$)
**return** $KG$
**function** EXTRACTTRIPLES($doc, O$)
  **return** LLM($doc$, prompt=$P$, constrained_by=$O$)

#### 3.2.4 Graph Population

Once the canonical triples are prepared, they are used to populate a graph database. Each node in the graph is enriched with structured metadata, including *"name", "id", "type"*, etc.

### 3.3 Ego Graph Retrieval

To retrieve the most relevant tools in response to a single or multi-step user query, we employ a custom ego-graph retrieval algorithm, described in Algorithm 2, which consists of three main stages:

#### 3.3.1 Entry Point Identification in the Tool Graph

- **Semantic Query-Node Matching:** We embed the user query using OpenAI's `text-3-embedding-large` embedding model and compute semantic similarity with all nodes in the graph. The *top-10* most semantically similar nodes are selected as candidate entry points, ensuring alignment based on meaning.
- **Textual Entity-Node Matching:** We perform unigram, bigram, and trigram matching between the user query and the text associated with nodes in the graph. Any matching nodes

**Algorithm 2** Ego Graph Tool Retrieval[3]

**Require:** User query $Q$, Knowledge Graph $KG$, embedding model $M$, reranker model $R$
**Ensure:** Ranked list of relevant tools $T$

$entrySem \leftarrow \text{MATCHBYSEMANTICSIM}(Q, KG, M)$
$entryText \leftarrow \text{MATCHBYTEXTUALSIM}(Q, KG)$
$entryNodes \leftarrow entrySem \cup entryText$
$candidateTools \leftarrow \emptyset$
**for all** node in $entryNodes$ **do**
    $egoGraph \leftarrow \text{ONEHOPNEIGHBORS}(KG, node)$
    $tools \leftarrow \text{EXTRACTTOOLNODES}(egoGraph)$
    $candidateTools \leftarrow candidateTools \cup tools$
$rankedTools \leftarrow \text{RERANK}(candidateTools, Q, R)$
**return** $\text{TOPK}(rankedTools, k = 10)$

**function** $\text{MATCHBYSEMANTICSIM}(Q, KG, M)$
    $scoredNodes \leftarrow \text{EMBEDDINGSIM}(Q, KG, M)$
    **return** $\text{TOPK}(embeddingMatches, k = 10)$

**function** $\text{MATCHBYTEXTUALSIM}(Q, KG)$
    **return** $\text{NODESWITHEXACTNGRAMMATCH}(Q, KG, n\_max = 3)$

**function** $\text{RERANK}(tools, Q, R)$
    $scoredTools \leftarrow [\,]$
    **for all** $tool$ in $tools$ **do**
        $score \leftarrow \text{RERANKER}(Q, tool, R)$
        Append $(tool, score)$ to $scoredTools$
    **return** $tools$ sorted in descending order by $score$

are also considered as entry points, capturing more direct keyword-based connections.

#### 3.3.2 One-Hop Neighborhood Expansion for Tool Candidate Set Enrichment

After identifying entry points, we execute a one-hop neighborhood expansion around each identified node, constructing an ensemble of ego tool graphs as previously described. This expansion process enriches our candidate set by incorporating all tool nodes directly connected to the entry points, thereby revealing contextually relevant tools that might otherwise remain undiscovered.

#### 3.3.3 Re-Ranking Retrieved Tools

The `llama-3.2-nv-rerankqa-1b-v2`[4] reranking model is used to re-rank the initially retrieved set of tools. The model takes as input the user query and each retrieved tool and outputs a relevance score for each. We retain the *top-10* tools with the highest scores as the final output for a given user query.
An end-to-end example, along with a sample graph snippet, is provided in Appendix under section A.

[4] https://build.nvidia.com/nvidia/llama-3_2-nv-rerankqa-1b-v2/modelcard

## 4 Dataset Generation

To evaluate our graph-based method, we require a tool retrieval benchmark suitable for enterprise use. Existing tool use benchmarks do not meet all these requirements, necessitating a custom dataset, cp. Table 2.

### 4.1 User Query Classification in Enterprise Task-Oriented Systems

Based on the requirements of our enterprise-oriented dialogue system, we have identified a taxonomy of query classes that reflect the diversity and complexity of real-world user requests. These classes help in understanding the structure, dependencies, and execution strategies required for accurate query interpretation and response generation. The classification is as follows:

- **Single-Intent Queries** involve only a single request with no additional steps, conditions, or dependencies, requiring direct execution.
- **Multi-Intent Queries** contain multiple independent requests that can be processed in any order or in parallel, with no logical dependencies between actions.
- **Explicit Multi-Step Queries** include multiple actions where dependencies between steps are clearly stated in the query, requiring strict execution order.
- **Implicit Multi-Step Queries** contain multiple actions where dependencies are implied rather than explicitly stated. The system must infer missing steps and their sequence before executing the main task.
- **Conditional Multi-Step Queries** explicitly state a condition that must be met before executing some of the actions involved.
- **Information Retrieval + Multi-Intent Queries** combine general knowledge inquiry with personalized action, including both broad information requests and targeted instructions

An example for each type can be found in Table 3.

### 4.2 Synthetic Multi-Step Query Generation

We introduce a structured *Query Generation Pipeline* covering all the aforementioned query types. This pipeline comprises three key components—**Path Identification**, **Query Genera-**

| | Ours | ToolLinkOS (Lumer et al., 2025) | ToolSandbox (Lu et al., 2025) | ToolBench (Qin et al., 2023) | ToolBank (Moon et al., 2024) | ToolRet (Shi et al., 2025) |
|---|---|---|---|---|---|---|
| **Number of Tools** | 177 | 573 | 34 | 16,464 | 3,168 | 43,000 |
| **Number of Queries** | 503 | 1,569 | 1,032 | 126,486 | 163,000 | 7,600 |
| **Tool Dependencies** | ✓ | ✓ | ✓ | x | x | x |
| **KG Schema** | ✓ | ✓ | x | x | x | x |
| **Complex Query Types** | ✓ | x | x | x | x | x |
| **Business Tools** | ✓ | x | x | x | x | x |

Table 2: Comparison of our proposed business query dataset with other tool retrieval benchmarks.

**tion**, and **Query Validation**—that collectively synthesize realistic and semantically grounded user queries spanning the various user query classes.

#### 4.2.1 Path Identification

This stage constructs meaningful tool chains by modeling both semantic and functional relationships using graph-based techniques. Specifically, we construct two graph structures to support diverse multi-step execution paths:

- **P-P Graph Construction:** Inspired by ToolFlow (Wang et al., 2025b), this graph captures semantic proximity between tools based on cosine similarity of input parameter embeddings. Each node corresponds to a tool, and edges are established when similarity exceeds a predefined threshold, suggesting potential sequential usage or shared functional behavior. This structure enables efficient exploration of tool compositions for multi-step planning.
- **Inferred R-P Graph Construction:** We employ the `o3-mini`[5] reasoning model to infer plausible output parameters for individual tools. These outputs are matched to tools accepting them as inputs, forming directed output-to-input edges annotated with confidence scores. Each tool sequence is passed through a validation step where we use `o3-mini` to reason where each generated sequence is valid. This graph reveals latent dependencies across tools, allowing for the construction of semantically valid but previously undocumented multi-tool flows.

Together, these graph structures enable robust path exploration for generating logically coherent multi-step queries.

[5] https://openai.com/index/openai-o3-mini/

#### 4.2.2 Query Generation

Once valid tool sequences are identified, the pipeline synthesizes realistic user queries aligned with execution paths and class-specific semantics:

- Generates grammatically well-formed queries for each tool path.
- Adapts structure and phrasing to match one of six predefined user query classes, ensuring linguistic clarity and categorical separation.
- Instantiates abstract parameters with realistic sample values for contextual relevance.
- Promotes query diversity by varying linguistic style and avoiding repetitive formulations.

This step transforms tool logic into realistic language patterns, enabling robust evaluation of retrieval systems under multi-step user query conditions.

#### 4.2.3 Query Validation

To ensure fidelity and structural correctness, the generated queries undergo systematic validation:

- **Class Validation:** Verifies that each query is properly classified according to its structural and semantic attributes.
- **Logical Sequence Verification:** Verifies that the tools used in a multi-step query are contextually compatible and collectively resolve the intended task. It checks whether each tool's input and output logically align, preserving semantic coherence across the entire sequence.
- **Error Detection:** Identifies anomalies such as repeated tool references, incomplete requests, or incompatible parameter logic; such queries are flagged for exclusion.

These checks ensure the integrity of the synthetic dataset, enabling reliable evaluation of tool retrieval frameworks.

Detailed example prompts for R-P graph construction, query creation and validation are shared in Figures 7–9 in Appendix C.

| Query Type | Example |
|---|---|
| Multi-Intent | Can you show me the hire date of my manager, John, and then tell me which department he belongs to? |
| Explicit Multi-Step | Can you show me the details of my expense with report ID `R1234` and then update the transaction amount to 500 with the currency code `USD`? |
| Implicit Multi-Step | I need to adjust the transaction amount of my expense with report ID `R1234` to 500. |
| Conditional Multi-Step | If the transaction date of my expense with report ID `R1234` is before 2022-01-01, update the transaction amount to 500. |
| Information Retrieval + Multi-Intent | What's the current stock status? Also, adjust the product allocation profiles based on the stock information. |

Table 3: Representative Synthetic Queries Generated for Each Query Class

### 4.3 Analysis of Generated Query Types

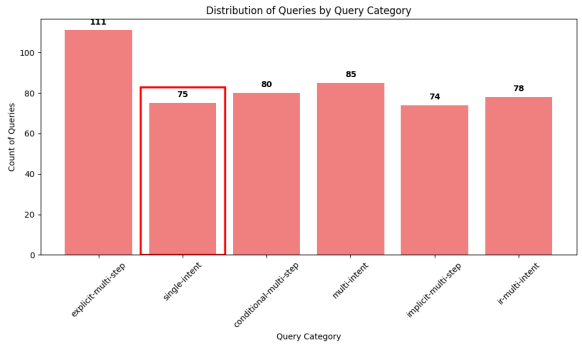


Figure 2: Query Distribution across identified query classes

As a key outcome of our proposed pipeline, we generate a diverse set of synthetic queries spanning across five complex query classes. We analyze the distribution of these queries and provide qualitative examples to showcase how the pipeline captures the structural and semantic characteristics of real-world enterprise queries. As seen in Figure 2, we categorized utterances into six distinct query classes based on intent category. The *single-intent* category consists of real user queries sampled from production logs. The examples for the remaining query classes were synthetically generated using the method explained above to closely match this empirical distribution. Detailed examples for queries generated for each query type are present in Table 3. This approach ensures our dataset reflects realistic usage patterns while enabling scalable coverage of more complex query types.

## 5 Experimental Results

### 5.1 Experimental Setup

We constructed our graph database using an internal dataset comprising semi-structured information extracted from several hundred enterprise tools within a large software platform. This dataset includes detailed descriptions, parameter specifications, and associated metadata, serving as the foundational data for the graph construction described in Section 3.2.

To evaluate our system, we employed a separate dataset of synthetic user queries generated through the pipeline outlined in Section 4.2. These queries were created by selecting targeted subsets of tools, formulating logical reasoning paths, and designing multi-step task-oriented queries. Each query was subsequently refined through a combination of automated validation and manual review to ensure high fidelity and practical alignment with real-world enterprise use cases.

### 5.2 Evaluation Metrics

We evaluate retrieval performance using the *CompleteRecall* metric, defined formally as:

$$CompleteRecall = \frac{1}{|Q|} \sum_{q \in Q} \mathbf{1}(Recall@k(q) = 1.0)$$

where $Q$ is the evaluation query set and $\mathbf{1}$ is an indicator function that returns 1 if $Recall@k$ for a given query $q$ is exactly 1.0—meaning all required tools for that query are present within the *top-k* retrieved items, with $k$ being the selected rank cutoff. This metric is tailored to planning systems where task breakdown depends on retrieving a complete set of expected tools. A comparable notion of complete recall has been used in prior work in the context of table retrieval (Zhang et al., 2025).

### 5.3 Retrieval Results and Analysis

To evaluate retrieval effectiveness, we compared four approaches: semantic retrieval, lexical retrieval (Okapi BM25[6]), hybrid retrieval, as well as our proposed KG-based retrieval method. Table 4 summarizes the *CompleteRecall* metric across all four approaches broken down by query type as well as aggregated averages. Comparable graph-based baselines are not available because no existing work provides enterprise-grade tools or real-world enterprise setups of this kind. As a result, direct empirical comparison is not feasible.

Semantic retrieval using dense vector embeddings to capture the similarity between queries and tool descriptions achieved a micro-average *CompleteRecall* of $59.84\%$ at $k = 3$, $73.96\%$ at $k = 5$, and $85.69\%$ at $k = 10$, but struggled with nuanced queries requiring deeper contextual understanding.

For lexical retrieval, we evaluated Okapi BM25 with three tokenization strategies: simple whitespace split, regex-based tokenization using `(\b\w+\b)`, and SpaCy lemmatization[7]. These were tested across two input types—tool description only, and description plus title. Including titles generally improved performance, with regex-based tokenization achieving micro-average *CompleteRecall* of $48.31\%$ at $k = 3$, $61.63\%$ at $k = 5$, and $76.54\%$ at $k = 10$ on the combined input.

Our hybrid baseline combines the *top-10* results from both semantic and lexical retrieval approaches, and re-ranks them using `llama-3.2-nv-rerankqa-1b-v2`. This strategy achieves micro-average *CompleteRecall* of $62.43\%$ at $k = 3$, $78.93\%$ at $k = 5$, and $89.26\%$ at $k = 10$, outperforming the standalone methods.

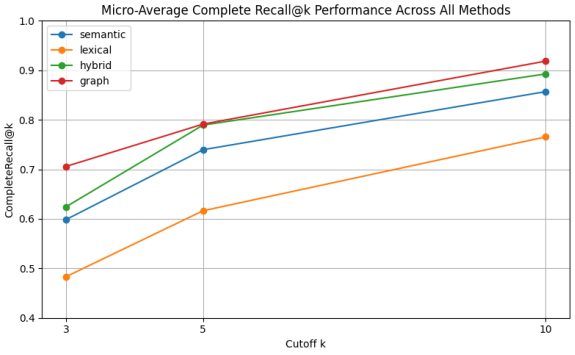


Figure 3: *CompleteRecall@k* micro-averages for each retrieval method

[6] https://pypi.org/project/rank-bm25/
[7] https://spacy.io/api/lemmatizer

Our knowledge graph-based approach significantly outperformed all other methods, achieving micro-average *CompleteRecall* scores of $70.58\%$ at $k = 3$, $79.13\%$ at $k = 5$, and $91.85\%$ at $k = 10$, as shown in Figure 3. Figure 4 highlights the most significant gains in complete recall within the conditional multi-step and implicit multi-step query categories. Appendix B provides further insights into our observations. We attribute these improvements to the KG's ability to explicitly model semantic relationships among tools, enabling context-aware retrieval and revealing connections through shared functionalities and data dependencies.

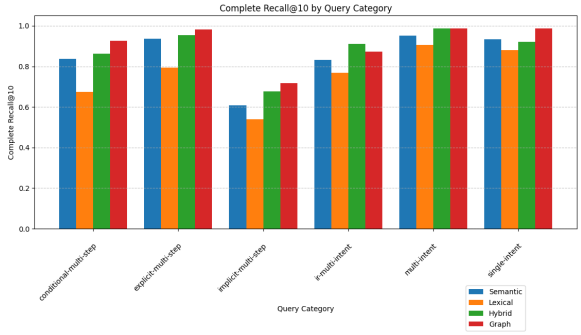


Figure 4: *CompleteRecall@10* for each query type

Some examples of its effectiveness:

- A query involving *"budget planning"* and *"project timelines"* retrieved interconnected tools such as *financial forecasters* and *scheduling applications* based on their shared association with time-based planning.
- Tools related to *"user onboarding"* were retrieved as a group due to common entities such as *authentication, documentation*, and *workflow setup*, facilitating the discovery of a complete on-boarding toolkit.

These findings demonstrate that the proposed KG-based retrieval architecture enables more accurate and complete tool discovery, particularly in enterprise task planning environments that demand coordinated multi-tool workflows.

## 6 Limitations

While our approach improves tool retrieval performance for several complex query types, it has notable limitations. First, its effectiveness depends heavily on the quality and completeness of the underlying knowledge graph. Errors in triple extraction or missing tool relationships can considerably weaken results and create a lower bound

| Query Category | Lexical | | | Semantic | | | Hybrid | | | Graph | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | @3 | @5 | @10 | @3 | @5 | @10 | @3 | @5 | @10 | @3 | @5 | @10 |
| conditional-multi-step | 42.50 | 52.50 | 67.50 | 55.00 | 67.50 | 83.75 | **58.75** | **76.25** | 86.25 | **58.75** | 72.50 | **92.50** |
| explicit-multi-step | 43.24 | 58.56 | 79.28 | 71.17 | 84.68 | 93.69 | 71.17 | **90.09** | 95.50 | **77.48** | 85.59 | **98.20** |
| implicit-multi-step | 25.68 | 36.49 | 54.05 | 25.68 | 43.24 | 60.81 | 32.43 | **51.35** | 67.57 | **37.84** | 47.30 | **71.62** |
| ir-multi-intent | 39.74 | 57.69 | 76.92 | 47.44 | 66.67 | 83.33 | 52.56 | **74.36** | **91.03** | **62.82** | **74.36** | 87.18 |
| multi-intent | 63.53 | 81.18 | 90.59 | 69.41 | 87.06 | 95.29 | 69.41 | 85.88 | **98.82** | **90.59** | **94.12** | **98.82** |
| single-intent | 76.00 | 82.67 | 88.00 | 84.00 | 88.00 | 92.00 | 85.33 | 89.33 | 92.00 | **90.67** | **96.00** | **98.67** |
| **Micro-Average** | 48.31 | 61.63 | 76.54 | 59.84 | 73.96 | 85.69 | 62.43 | 78.93 | 89.26 | **70.58** | **79.13** | **91.85** |

Table 4: *CompleteRecall*@{3, 5, 10} (%) across query categories and retrieval methods. Best @{3, 5, 10} score per row is highlighted in bold.

on achievable performance, since inaccuracies in the extraction phase may propagate through later stages. Second, the framework may be less robust in domains with highly heterogeneous or sparsely described tools, reducing node-matching accuracy.

Surprisingly, for certain complex query types, traditional semantic and lexical-semantic hybrid retrieval methods outperformed our graph-based approach. This suggests that the additional structural complexity may not always provide benefits and warrants further investigation into when graph-based methods are most advantageous.

Finally, while we demonstrate strong results in our enterprise task domain, broader evaluation is needed to assess generalization across different domains and tool ecosystems.

## 7 Conclusion and Future Work

The task of efficiently exploring available tools, which is crucial for effective task decomposition during agentic planning, remains challenging for LLM-powered systems, particularly in enterprise environments with numerous tools that have complex and often undocumented interdependencies. Our research was guided by the hypothesis that explicitly modeling these relationships in a graph structure would enhance tool retrieval effectiveness. To address the scarcity of complex queries in existing datasets, we propose a synthetic query generation pipeline that models tool dependencies through parameter-level connections, enabling the generation of realistic, multi-step queries.

We present a systematic approach to transform enterprise tools into a coherent graph representation and introduc a novel *Ensemble of Ego-Graphs (EEG)* retrieval framework that outperforms traditional baselines. Our results as shown in Table 4 support our hypothesis and establish a promising direction for improving tool retrieval in complex enterprise environments.

Future research directions addressing some of the shortcomings we have identified include:

- Implementing a triple validation step to improve the quality of graph connections
- Adding additional dimensions to our dataset to go beyond the current defined classes and better capture the real-world variability and messiness of user queries.
- Making our dataset publicly available
- Exploring graph embedding techniques to complement our EEG retrieval algorithm
- Developing methods to incorporate tool response information to enhance the tool graph's utility
- Evaluating and optimizing graph-retrieval latency to make it comparable with current retrieval mechanisms
- Introducing query augmentation strategies to inject realistic linguistic variability, such as ambiguity, underspecification, and incomplete phrasing to improve generalizability to real-world enterprise queries
- Incorporating inter-annotator agreement measures (e.g., Cohen's Kappa) to evaluate consistency among experts and between experts and the automated validation pipeline to strengthen the reliability of evaluation outcomes

Through these efforts, we aim to further advance tool retrieval capabilities for enterprise applications.

## 8 GenAI Usage Disclosure

We employed ChatGPT and Claude to assist in rephrasing certain sections of the paper for improved clarity. All core content, including research design, data analysis, and result interpretation, was conducted without the aid of generative AI tools.

## References


Anonymous. 2024. Tool graph retriever: Exploring dependency graph-based tool retrieval for large language models. In *Submitted to ACL Rolling Review - June 2024*. Under review.

Jose A Diaz-Garcia and Julio Amador Diaz Lopez. 2025. A survey on cutting-edge relation extraction techniques based on language models. *Artificial Intelligence Review*, 58(9):287.

Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *Preprint*, arXiv:2402.02716.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. 2024a. Toolnet: Connecting large language models with massive tools via tool graph. *Preprint*, arXiv:2403.00839.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and 1 others. 2024b. Controlllm: Augment language models with tools by searching on graphs. In *European Conference on Computer Vision*, pages 89–105. Springer.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, Albuquerque, New Mexico. Association for Computational Linguistics.

Elias Lumer, Pradeep Honaganahalli Basavaraju, Myles Mason, James A. Burke, and Vamse Kumar Subbiah. 2025. Graph rag-tool fusion. *Preprint*, arXiv:2502.07223.

Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. 2024. Efficient and scalable estimation of tool representations in vector space. *Preprint*, arXiv:2409.02141.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 1930–1940, New York, NY, USA. Association for Computing Machinery.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. 2025. Tool learning with large language models: a survey. *Frontiers of Computer Science*, 19(8).

Mrinal Rawat, Ambuje Gupta, Rushil Goomer, Alessandro Di Bari, Neha Gupta, and Roberto Pieraccini. 2025. Pre-act: Multi-step planning and reasoning improves acting in llm agents. *Preprint*, arXiv:2505.09970.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, Vienna, Austria. Association for Computational Linguistics.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025a. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2025b. ToolFlow: Boosting LLM tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4246–4263, Albuquerque, New Mexico. Association for Computational Linguistics.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. PlanGenLLMs: A modern survey of LLM planning capabilities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19497–19521, Vienna, Austria. Association for Computational Linguistics.

Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng,

Yang Wang, and Enhong Chen. 2024. Large language models for generative information extraction: A survey. *Frontiers of Computer Science*, 18(6):186357.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. 2025. MURRE: Multi-hop table retrieval with removal for open-domain text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5789–5806, Abu Dhabi, UAE. Association for Computational Linguistics.

## A Complete Example

**User Query:** *Show me the details of all purchase order items with 'pending' status.*

**Target Tools:**

```
query purchase order item,
read purchase order item
```

**Entity-Node Matches:**

```
purchase order item, detail,
purchase order
```

**Top Relevant Tools Identified:**

```
read purchase order item,
query purchase order item,
show sale order query item,
read purchase requisition item,
query purchase order header,
read purchase order definition,
query purchase requisition item,
show sale order read header,
read purchase requisition
definition
```

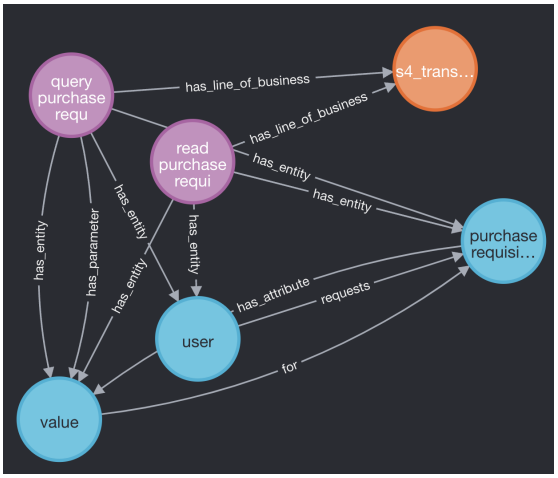


Figure 5: Subgraph to help illustrate the example in Appendix A

## B *CompleteRecall@10* per Query Category

Figure 4 shows a histogram comparing the *CompleteRecall@10* performance of each retrieval system across different query categories.

- For conditional multi-step, explicit multi-step, and implicit multi-step queries, the **graph** tool retrieval system performs best, indicating that the extracted triples effectively capture tool dependencies within the graph, enabling more accurate semantic reasoning and retrieval.
- For multi-intent, the graph model performs on par with the hybrid model.
- For information retrieval multi-intent queries, the graph model performs under-par when compared with the hybrid model.
- These results highlight that the graph model is especially effective for queries with implicit or complex dependencies that are not explicitly modeled, demonstrating the advantage of structured graph representations in capturing hidden relationships that improve retrieval performance.

## C Prompts

```
You are a top-tier natural language understanding expert, skilled in
extracting triplets for Knowledge Graph construction.
Objective: Extract structured triplets (head-relationship-tail) from the
given text to build a knowledge graph. The input is a scenario description
from a digital assistant framework.
Instructions:
1. Read the provided text carefully.
2. Extract triplets in the format: { "head": ..., "tail": ...,
"relationship": ... }.
- head: the main entity or concept performing an action or being described.
- relationship: the action or relation connecting the head to the tail.
- tail: the entity or concept receiving the action or being related.
3. Include default triplets based on specific patterns:
- has_line_of_business: between the scenario title and line of business.
- has_entity: between the scenario title and entities in the text.
- has_parameter: between the scenario title and listed parameters.
4. Also capture other relevant relationships, such as: contains,
related_to, used_for, has_attribute, associated_with, managed_by, part_of,
required_for, depends_on, produces, receives_from, involved_in, reports_to,
responsible_for, affects, includes.
Important Note: For every "head" and "tail" entity in the triplets, include
an additional has_entity triplet linking it to the scenario title.
5. Output must be a valid JSON dictionary with this structure (and no extra
text):

{
  "relationships": [
    {"head": "...", "tail": "...", "relationship": "..."},
    ...
  ]
}

6. Example:
Text: "The scenario titled 'Error Reporting' pertains to the Line of
Business 'Finance'. The scenario description entails: 'As a user, I want to
see a report of the top rejection errors for invoices.' The parameters are:
error_id, resolved_number."
Output:

{
  "relationships": [
    {"head": "Error Reporting", "tail": "Finance",
     "relationship": "has_line_of_business"},
    {"head": "Error Reporting", "tail": "User", "relationship": "has_entity"},
    {"head": "Error Reporting", "tail": "Report",
     "relationship": "has_entity"},
    {"head": "Error Reporting", "tail": "Rejection error",
     "relationship": "has_entity"},
    {"head": "Invoice", "tail": "Rejection error", "relationship": "contains"},
    {"head": "Report", "tail": "User", "relationship": "generated_for"},
    {"head": "Error Reporting", "tail": "error_id",
     "relationship": "has_parameter"},
    {"head": "Error Reporting", "tail": "resolved_number",
     "relationship": "has_parameter"}
  ]
}

7. Strict rules: - Do not duplicate triplets. - Do not allow "head"
and "tail" to be identical. - The scenario title may only appear in
has_line_of_business, has_entity, and has_parameter triplets. - Break
complex sentences into simpler ideas to ensure accurate extraction. -
Maintain consistent naming of entities and relationships.
Follow the above instructions exactly, and output only a valid JSON
dictionary.
```

Figure 6: Prompt for domain-specific ontology-guided triple extraction from scenario descriptions.

**System Prompt:**
You are an expert reasoning agent tasked with identifying output parameters in a business scenario.
**You will be provided with:** (1) a scenario name and description, (2) a list of input parameters and their descriptions, (3) a fixed list of **available parameters**, which are the only candidates you may choose from as outputs.
**Your Task:** Determine **which (if any)** of the available parameters are likely to be **outputs**--meaning they are **generated**, **updated**, **or returned** as a result of executing the scenario.
- Pay specific attention to the parameter descriptions when choosing a likely output.
- You may include up to 3 parameters.
- Only choose parameters that have **strong logical support** based on the scenario and inputs.
- If there is no clear evidence, return an empty list: [].
- Do not guess or assume without justification--**precision is more important than recall.**
**Scoring Criteria:** For each selected parameter, provide a **confidence score** between 0 and 1, based on the following:
- **Relevance:** How directly the parameter aligns with the business goal or result described in the scenario.
- **Causality:** Whether the parameter is clearly generated or changed as a consequence of executing the scenario.
- **Clarity:** Whether the scenario description explicitly or implicitly implies this parameter is affected or produced.
- **Typical Usage:** Whether this parameter is commonly used as an output in similar scenarios or business processes.
**Scoring scale:**
0.90-0.99: Very strong evidence -- directly and explicitly implied as an output; all four dimensions clearly supported.
0.70-0.89: Strong inference -- not explicitly stated but logically follows from the scenario and typical practices.
0.50-0.69: Weak or partial evidence -- some contextual hints or common patterns suggest it, but not clearly supported.
< 0.50: Do not include -- insufficient support or speculative.
**Output Format:** Respond only with a JSON array, with **no markdown, no headings, and no surrounding text.** Each item must match the structure below and correspond exactly to entries in available_params.
**Example:**

```
[
  {
    "parameter_name": "string (must match exactly from available_params)",
    "parameter_id": "string (must match exactly from available_params)",
    "confidence_score": float (0 to 1),
    "reasoning": "Short explanation (1-2 sentences max)."
  }
]
```

**Notes:**
- Only select from the list: {available_params}.
- Return [] if no likely output parameters.
**User Prompt:**

```
Scenario ID: {scenario_descriptions['scenario_id'][i]}
Scenario Name: {scenario_descriptions['joule_scenario_title_std'][i]}
Scenario Description: {scenario_descriptions['scenario_description'][i]}
Input Parameter Descriptions: {scenario_descriptions['Parameter_Info'][i]}
```

Figure 7: Prompt for identifying likely output parameters in business scenarios with confidence scoring.

You are an intelligent query generation agent whose goal is to generate user queries using the logical function paths provided to you.
Here are the logical function paths:{query_generator_input}
Here are the different classes of user queries that have to be generated using these paths:

**1. Explicit Multi-Step User Queries**

- Include multiple actions where each step explicitly depends on the completion of the previous one.
- Require a strict execution order, ensuring prior steps are processed before moving forward. Often use sequence-based phrasing such as "Show me X, then do Y."

Example: "Show me Jan's information and send him a spot award with a budget based on his career level."

**2. Implicit Multi-Step User Queries**

- Contain multiple actions, but the dependency between steps is implied rather than explicitly stated. The system must infer missing steps before executing the main task.
- User queries belonging to this class must not include sequencing phrases (e.g., "Do this, then do that"), conjunctions like "and" or "also" for distinct tasks, or conditional constructions (e.g., "If X happens, then do Y").

Example: "Send an email reminder to all suppliers invited to the Sapphire event." (Determining the list of invited suppliers is implicit.)

**3. Conditional Multi-Step User Queries**

- Depend on a condition being met before executing an action. Often use phrasing like "If X happens, do Y" or "Only if A is true, execute B."
- Require logical decision-making to ensure the correct steps are triggered.

Example: "Show me items related to GL 1234 if the account balance exceeds $1M."

**4. Multi-Intent User Queries**

- Contain multiple independent requests that can be processed in any order or in parallel, with no logical dependencies between actions.

Example: "Show my direct reports and display the weather forecast."

**5. Information Retrieval + Multi-Intent User Queries**

- Combine a general knowledge inquiry with a personalized action. The informational query pertains to rules, definitions, or external facts, while the personal request focuses on user-specific data or tasks. Typically structured with both a broad question and a targeted action.

Example: "What is a spot award? Also, show me mine."

**Instructions:**

- Review the given logical path, including all functions, their purpose, descriptions, and input parameters.
- Generate one natural-sounding user query for each of the five classes based on the logical path.
- Ensure each query clearly reflects the intent of its respective class.
- Sound fluid, conversational, and human-like -- avoid robotic or overly formal phrasing.
- Avoid internal domain-specific terminology and do not reuse exact words or phrases from function descriptions.
- Use realistic, fake values for at least one function's input parameters (e.g. "location": "Chicago", "amount": 200) in the utterances.
- Make each query sound like something a real user might say in a relevant context.
- Verify that each user query distinctly and accurately reflects the intended class, ensuring no overlap or confusion between the different user intents.

**Output Instructions:**

- Provide the final output strictly in this format: {format_instructions}
- Do not include extra text like "json" or "output" in the response.

Figure 8: Prompt for generating user queries across various multi-step classes using structured functional paths.

**System Prompt:**
You are an expert in scenario analysis and workflow planning. Your task is to evaluate whether a sequence of two scenarios is valid based on their ability to follow one another in a logical, functional, and operational manner.
Each scenario includes a unique ID, a title, and a description of the actions or behaviors involved. The first scenario must be completed before the second one can logically occur, and the actions in the second scenario must be a valid continuation or follow-up to the first.
The scenarios must be part of a **multi-step process**, where the first scenario sets up a necessary context or action that the second can build upon. The scenarios cannot have distinct or unrelated intents. The second scenario must build upon the result or state created by the first. If the two scenarios are unrelated or do not form a cohesive multi-step action, the sequence should be considered invalid.
You will receive a list of two scenarios. Your task is to determine whether the second scenario can validly follow the first scenario in a multi-step process, based on the logical flow and dependencies between them.
Return Format: Provide a valid JSON dictionary with the following fields:

```
{
  "from_scenario_id": "string",
  "to_scenario_id": "string",
  "is_valid": true or false,
  "explanation": "Short rationale with example use case."
}
```

**Notes:**
- You must only select from this list of available parameters: {available_params}
- Return an **empty list** (`[]`) if there are no likely output parameters.

**User Prompt:**
Scenario ID: {scenario_descriptions['scenario_id'][i]}
Scenario Name: {scenario_descriptions['joule_scenario_title_std'][i]}
Scenario Description: {scenario_descriptions['scenario_description'][i]}
Input Parameter Descriptions: {scenario_descriptions['Parameter_Info'][i]}

Figure 9: Prompt for validating multi-step scenario transitions using structured logical analysis.