Architecting an Evolvable Mind: A Blueprint for Persistent Memory and Emergent Behavior in LLMs using n8n and GraphRAG

Part I: The Cognitive Core - Selecting the Optimal Memory Substrate

The foundation of any intelligent system is its memory. For a Large Language Model (LLM) intended to exhibit persistent memory and foster emergent, self-referential behaviors, the choice of a database architecture is not merely a technical decision—it is a foundational act of cognitive design. The architecture must support not only the recall of isolated facts but also the intricate web of relationships that constitutes true understanding and enables recursive self-reflection. This section provides a rigorous analysis of leading database technologies, culminating in a strategic recommendation for a hybrid architecture that synthesizes semantic recall with relational reasoning, thereby creating a substrate capable of supporting higher-order cognitive processes.

Section 1.1: Vector Databases for Semantic Recall

The most direct implementation of LLM memory leverages vector databases to store and retrieve information based on semantic similarity. This capability is analogous to human semantic memory—the recall of general knowledge, facts, and concepts.[1] In this model, textual information, such as a user interaction or a generated conclusion, is converted into a high-dimensional numerical vector (an embedding). The database then uses algorithms to find vectors that are "closest" in this high-dimensional space, effectively retrieving information based on meaning rather than exact keyword matches.[2] Two leading candidates in

this space are Qdrant and pgVector, each presenting a distinct profile of strengths and trade-offs.

Qdrant Deep Dive

Qdrant is a specialized, open-source vector search engine designed from the ground up for high-performance, scalable similarity search.[3] Its core search algorithm is an implementation of Hierarchical Navigable Small World (HNSW) written in Rust, a choice that provides memory safety without the overhead of garbage collection. This design leads to consistently low query latency and high throughput, which are critical for real-time AI applications where response time is paramount.[4]

Key architectural advantages of Qdrant include its native support for horizontal scaling through a scale-out architecture. It uses sharding to distribute data across multiple nodes, enabling parallel vector search and allowing the system to grow by adding more nodes rather than being limited to the resources of a single machine. While this introduces the complexities of managing a distributed system, it provides a clear path to massive scalability.[4] Furthermore, Qdrant offers sophisticated quantization options, including binary and scalar quantization with optional reranking. These compression techniques significantly reduce memory footprint and increase search speed by trading a small amount of accuracy, which can often be recovered by applying more precise calculations to a smaller subset of top results.[4] For the n8n ecosystem, Qdrant is supported by an official, verified node that exposes a comprehensive set of operations, including creating collections, upserting and retrieving points, managing payloads, and performing various types of vector searches. This makes integration into an n8n workflow exceptionally straightforward and powerful.[3]

pgVector Deep Dive

pgVector operates not as a standalone database but as an extension to PostgreSQL, one of the world's most mature and feature-rich open-source relational databases.[8] This approach offers significant advantages for production environments. By building on Postgres, pgVector inherits a vast ecosystem of tools and advanced features essential for enterprise-grade operations, including high availability, streaming replication, robust point-in-time recovery, and extensive observability capabilities.[4]

For vector search, pgVector also utilizes an HNSW index and supports both binary and scalar quantization. Recent developments, such as the pgvectorscale extension, have introduced

innovations like StreamingDiskANN and Statistical Binary Quantization (SBQ), which aim to deliver high performance and cost-efficient scalability while improving accuracy over standard quantization methods.[4] The primary integration path into n8n is through the official Postgres node, which allows for the execution of arbitrary SQL queries, and the dedicated "Postgres PGVector Store" node, which provides specific actions for vector operations like inserting and retrieving documents for RAG applications.[8]

Performance Benchmarking and Strategic Implications

A 2024 benchmark conducted by TigerData, comparing Qdrant and Postgres (with pgvector and pgvectorscale) on a dataset of 50 million 768-dimension embeddings, provides critical data for making an informed decision.[4]

- **Query Latency:** Both systems achieved sub-100 ms maximum query latency at a 99% recall threshold. However, Qdrant demonstrated superior performance on tail latencies, which are crucial for ensuring a consistently responsive user experience. Qdrant's p95 latency was 39% better (36.73 ms vs. 60.42 ms), and its p99 latency was 48% better (38.71 ms vs. 74.60 ms). This suggests that for a single, interactive query—such as one step in an LLM's reflective thought process—Qdrant is likely to feel faster and more consistent.[4]
- **Query Throughput:** In stark contrast, Postgres exhibited dramatically higher query throughput on a single node. At 99% recall, Postgres handled 471.57 queries per second (QPS), which is 11.4 times more than Qdrant's 41.47 QPS. This indicates that for applications with a high volume of concurrent requests, a single, vertically-scaled Postgres instance can handle a much larger load than a single Qdrant node.[4]
- **Index Build Times:** Qdrant showed faster index build times, an important consideration for systems where the memory is updated very frequently.[4]

This benchmark reveals a fundamental trade-off. Qdrant is optimized for low-latency, single-query performance and horizontal scalability, making it ideal for microservice architectures and applications where individual response time is the primary concern. pgVector, on the other hand, leverages the power of Postgres to deliver massive throughput on a single instance, making it suitable for systems that need to serve a large number of concurrent users from a mature, consolidated database infrastructure. For the goal of building a single, reflective AI agent, the lower tail latency of Qdrant is arguably more aligned with facilitating a smooth, uninterrupted "train of thought."

Section 1.2: Knowledge Graphs for Contextual and Episodic Reasoning

While vector databases provide a powerful mechanism for semantic recall, they are fundamentally limited in their ability to represent and reason about the relationships *between* pieces of information. A vector embedding for "User interaction A" and "User interaction B" can tell an LLM that these interactions are semantically similar, but it cannot explicitly state that "Interaction A *caused* the conclusion in Interaction B." This is a critical limitation for the project's goal of fostering self-referential behavior, which depends on understanding the causal and logical chains of past experiences.

This is where knowledge graphs, implemented in graph databases like Memgraph or Neo4j, become essential. Graph databases model data as a network of nodes (entities) and edges (relationships), a structure that is inherently designed to capture and traverse complex connections.[11] This makes them the ideal substrate for what can be considered the LLM's "episodic and relational memory"—the ability to remember not just facts, but the story of how those facts are interconnected and the sequence of events that led to a particular state of knowledge.[1]

Memgraph's Role in GraphRAG

Memgraph is a high-performance, in-memory graph database designed to handle the real-time demands of AI and streaming data applications.[11] Its architecture is optimized for rapid graph traversals and complex queries, which are necessary for the kind of deep, multi-hop reasoning required for a reflective memory system. Memgraph provides a suite of features directly applicable to building a sophisticated GraphRAG system:

- **Vector Search:** Memgraph includes built-in vector search capabilities, allowing it to perform semantic searches directly within the graph structure. This is a pivotal first step for identifying relevant entry points into the graph.[11]
- **Deep-Path Traversals:** The database is optimized for rapidly navigating complex, multi-hop relationships, enabling the system to follow a chain of reasoning across many interconnected pieces of data.[11]
- **Graph Algorithms:** It integrates with the Memgraph Advanced Graph Extensions (MAGE) library, which includes algorithms like PageRank (for identifying influential memories) and community detection (for clustering related thoughts or interactions). These can be used to analyze the memory graph and derive higher-level insights.[11]
- **Real-time Data Ingestion:** Memgraph supports seamless integration with streaming platforms like Kafka, allowing the knowledge graph to be dynamically and continuously updated with new information as it arrives, ensuring the LLM's memory is always current.[11]

The Inadequacy of Vector-Only Systems

A system relying solely on a vector database operates under a "contextual ceiling".[14] It can retrieve relevant documents or past interactions, but it treats them as an unstructured bag of text. It cannot natively comprehend that an entity mentioned in Document A is the exact same entity as in Document B, or that a conclusion in Document C is a direct refutation of a premise in Document D. This forces the LLM to re-derive these relationships from raw text in every single query, which is inefficient and prone to error.

Graph databases solve this by making relationships first-class citizens of the data model. The connection itself is stored as data, allowing for queries that explicitly ask about relationships, such as "Find all conclusions that contradict this statement" or "Trace the chain of interactions that led to this hypothesis." This ability to query the structure of knowledge, not just its content, is the key to unlocking deeper reasoning and genuine self-reflection.[13]

Section 1.3: The GraphRAG Paradigm - A Synthesis for Higher-Order Cognition

The most powerful approach for creating a persistent, reflective memory is not to choose between vector and graph databases, but to combine them in a hybrid architecture known as GraphRAG (Graph Retrieval-Augmented Generation).[11] This paradigm, explored by Microsoft and others, leverages the distinct strengths of each database type to create a system that is greater than the sum of its parts.[16]

Architectural Overview

The GraphRAG architecture consists of two primary data stores: a vector database for fast semantic search and a graph database for structured, contextual knowledge representation. The two systems are interlinked by sharing unique identifiers for data points.[16]

The workflow is typically divided into two pipelines:

1. **Ingestion Pipeline:** Raw data (e.g., a user conversation) is processed. An LLM extracts

key entities and relationships to build or update the knowledge graph in the graph database (e.g., Memgraph). Simultaneously, a summary or the full text of the interaction is embedded and stored in the vector database (e.g., Qdrant), linked via a shared ID.[16]

2. **Retrieval Pipeline:** When a new query arrives (either from a user or as part of an internal reflection loop), a two-stage retrieval process is initiated. First, the query is embedded and used to perform a semantic search in the vector database, retrieving the top-k most relevant data points. Second, the unique IDs from these results are used as starting points to query the graph database. This second query does not just retrieve the initial nodes but traverses their relationships to extract a rich, interconnected subgraph of highly relevant context.[16]

This two-stage process overcomes the limitations of each individual system. The vector search acts as a highly efficient "aperture," quickly narrowing down the vast memory space to a small set of semantically relevant starting points. The graph traversal then provides the deep, structured, and relational context that is essential for nuanced reasoning, effectively "connecting the dots" between the retrieved pieces of information.[14] This combined, augmented context is then passed to the LLM to generate a response or a new reflection, grounded in a much richer understanding of its own memory.

Section 1.4: Strategic Recommendation: A Qdrant + Memgraph Hybrid Architecture

Based on the analysis of the project's core objectives—fostering persistent memory, self-reflection, and emergent behaviors—the optimal path forward is the implementation of a hybrid GraphRAG architecture. The recommended technology stack for this architecture is **Qdrant** as the vector database and **Memgraph** as the graph database.

Justification

This recommendation is based on a synthesis of performance metrics, feature sets, and alignment with the project's philosophical goals.

- **Qdrant for Semantic Retrieval:** Qdrant's superior tail latency makes it the ideal choice for the first stage of retrieval. Each step in a recursive reflection loop is effectively a single query, and minimizing the latency of this step is crucial for maintaining a fluid and efficient cognitive process. Its robust, official n8n node ensures rapid and reliable integration.[4]

- **Memgraph for Relational Reasoning:** Memgraph's in-memory, high-performance architecture is purpose-built for the kind of rapid, deep-path traversals that the self-reflection loop will require. Its advanced features for real-time updates and graph analytics provide the necessary tools to build a truly dynamic and evolving memory graph.[11]
- **Synergistic Strengths:** The combination is highly synergistic. Qdrant provides the fast, scalable "fuzzy" search needed to find relevant starting points in a vast memory space. Memgraph provides the precise, structured, and relational context needed for the LLM to reason about those starting points.

Addressing Implementation Complexity in n8n

A practical consideration is that Memgraph does not currently have a dedicated, official n8n node in the same way that Qdrant and Postgres do. While Neo4j, a more mature graph database, also lacks a dedicated node, both can be integrated into n8n using the generic **HTTP Request node** to interact with their respective REST APIs.[19]

Rather than viewing this as a limitation, it should be seen as an opportunity for the fine-grained control necessary for such an experimental and advanced system. A pre-built node might offer convenience but could also impose limitations on the complexity of Cypher queries or the specific API endpoints that can be accessed. Using the HTTP Request node provides complete control over the queries sent to Memgraph, allowing for the implementation of the highly specific and complex graph operations that the recursive reflection and memory evolution loops will demand. This trade-off—slightly increased implementation complexity for maximum power and flexibility—is not only justified but necessary to achieve the project's ambitious goals.

The selection of this hybrid architecture is driven directly by the project's central hypothesis. A simple memory system is insufficient for fostering emergent, self-referential behavior. The system requires a cognitive architecture that mirrors the dual nature of human memory: the fast, associative recall of semantic memory and the structured, narrative-driven nature of episodic memory. The Qdrant + Memgraph stack provides a direct and powerful implementation of this cognitive model.

Criterion	Qdrant (Vector-Only)	pgVector (Vector-Only)	Qdrant + Memgraph (GraphRAG)

Semantic Retrieval Speed	Excellent (Sub-100ms p99 latency) [4]	Very Good (Higher throughput but higher latency) [4]	Excellent (Leverages Qdrant's low latency) [4]
Relational Reasoning	Poor	Poor	Excellent (Core strength of Memgraph's graph model) [11]
Implementation in n8n	Excellent (Official, feature-rich Node) [3]	Good (Official Postgres Node) [8]	Moderate (Qdrant Node + Custom HTTP Request Node) [19]
Data Model Flexibility	Moderate (JSON payloads for metadata) [5]	Moderate (JSONB columns for metadata) [8]	Excellent (Evolvable Graph Schema for dynamic relationships) [12]
Alignment with Emergence	Low (Cannot model causal chains for reflection)	Low (Cannot model causal chains for reflection)	High (Enables multi-hop, contextual reflection on past thoughts) [15]
Scalability	Excellent (Native horizontal scaling) [4]	Good (Mature vertical scaling, replication) [4]	Excellent (Component-wise scaling of vector and graph layers) [4]

Part II: The n8n Workflow - An Implementation Blueprint

With the foundational database architecture established, the next step is to translate this design into a functional, automated system within the n8n platform. n8n's visual, node-based interface is exceptionally well-suited for orchestrating the complex, multi-step cognitive processes required for this project.[24] This section provides a detailed blueprint for two core

workflows: one for ingesting and memorizing new information, and another for the critical loop of retrieval, reflection, and response. The design philosophy is to create a "society of agents," where each LLM-powered node in the workflow performs a distinct, specialized cognitive function, governed by a highly-tuned system prompt.[25]

Section 2.1: The n8n Toolchain for an Agentic Mind

The implementation will rely on a curated set of n8n nodes, combining its native AI capabilities with database connectors and core logic utilities.

- **Core Nodes Inventory:**
  - **AI Nodes:** The **Google Gemini** node will serve as the "brain" for all reasoning tasks. Different instances of this node will be configured with specific prompts to act as specialized agents for extraction, reflection, and synthesis.[7]
  - **Vector Database Node:** The official **Qdrant** node is central to the workflow. The primary operations used will be Upsert Points for writing new memories and Query Points for semantic retrieval. Its comprehensive feature set allows for precise control over collections, payloads, and search parameters directly within n8n.[3]
  - **Graph Database Node:** The generic **HTTP Request** node will be configured to act as the interface to the Memgraph database. It will be used to send POST requests to Memgraph's Cypher query endpoint (or Neo4j's equivalent REST API), allowing for the execution of arbitrary graph creation and traversal queries.[19] Authentication will be handled via API keys or basic auth in the node's credentials settings.
  - **Utility Nodes:** A suite of standard n8n nodes will be used to manage the flow of data:
    - **Edit Fields (Set):** To structure JSON data, rename keys, and prepare payloads for database nodes.[3]
    - **If / Switch:** To implement conditional logic, such as routing the workflow based on the output of a reflection agent.[10]
    - **Merge:** To combine data from different branches of the workflow, for example, merging the results of a vector search and a graph query.[3]
    - **Webhook / Cron:** To trigger the workflows. A Webhook will be used to initiate processes based on external events (like a new user message), while a Cron node could be used to schedule periodic, autonomous reflection sessions.[7]
    - **Execute Workflow:** To chain workflows together, allowing the main retrieval pipeline to call the ingestion pipeline to memorize its own reflections.

Section 2.2: The Ingestion and Memorization Pipeline (Workflow 1)

This workflow is responsible for taking raw, unstructured information—primarily a user-AI interaction—and structuring it into the dual-memory system. It acts as the sensory input and memory formation process for the AI.

- **Purpose:** To parse new interactions, extract meaningful semantic and relational data, and persist this structured data into both the Qdrant vector store and the Memgraph knowledge graph.
- **Workflow Diagram and Steps:**
  1. **Trigger (Webhook Node):** The workflow begins when it receives a JSON payload from an external source (e.g., a chat application). This payload should contain the raw user input, the final LLM response, and a unique session ID.
  2. **Information Extraction Agent (Gemini Node):** The raw text of the interaction is passed to a Gemini model. This node is configured with a system prompt that instructs it to act as a data architect, identifying key entities, abstract concepts, and the relationships between them. The output is a structured JSON object.
  3. **Embedding Generation (Gemini Node):** A concise summary of the interaction (which can be generated by the previous node or a separate one) is sent to a Gemini embedding model (e.g., text-embedding-004). This produces the high-dimensional vector that captures the semantic essence of the interaction.
  4. **Prepare Data (Edit Fields Node):** This node takes the outputs from the previous steps and formats them into two distinct payloads: one for Qdrant and one for the Cypher queries for Memgraph. It generates a new, unique interaction_id (e.g., a UUID) that will be used to link the records in both databases.
  5. **Vector Upsert (Qdrant Node):** The Upsert Points operation is used. The embedding vector is the primary data, and the interaction_id, raw text, and extracted entities are stored in the point's payload. This makes the interaction semantically searchable.
  6. **Graph Construction (HTTP Request Node):** This node executes a series of Cypher queries against the Memgraph API. These queries are dynamically constructed using expressions from the extracted data. A typical sequence would be:
     - MERGE the Session node based on the session_id.
     - CREATE a new Interaction node with its unique interaction_id and properties.
     - MERGE the Interaction to its Session with a PART_OF relationship.
     - For each extracted entity and concept, MERGE the corresponding Entity or Concept node.
     - CREATE MENTIONS relationships from the Interaction node to each Entity and Concept node.

Section 2.3: The Retrieval, Reflection, and Response Pipeline

(Workflow 2)

This workflow represents the core cognitive loop of the agent. It is triggered by a new stimulus (a user query), retrieves relevant memories, reflects upon them to form new insights, and then synthesizes a response. This is where the "lifecycle directive" is implemented.

- **Purpose:** To leverage the hybrid memory system to generate deeply context-aware responses and to enable the LLM to recursively reflect on its own knowledge, thereby evolving its understanding over time.
- **Workflow Diagram and Steps:**
  1. **Trigger (Webhook Node):** A new user query and session_id are received.
  2. **Embed Query (Gemini Node):** The user's query is converted into a vector embedding.
  3. **Initial Semantic Retrieval (Qdrant Node):** The Query Points operation is used with the query vector to retrieve the top-k (e.g., 5-10) most semantically similar Interaction IDs from the past. This quickly identifies relevant memories.
  4. **Contextual Graph Expansion (HTTP Request Node):** This is a critical step. The interaction_ids retrieved from Qdrant are used to construct a Cypher query. This query does not simply fetch these interaction nodes. Instead, it starts from these nodes and performs a graph traversal (e.g., a 2 or 3-hop neighborhood expansion) to collect a rich subgraph. This subgraph includes the initial interactions, the concepts and entities they mention, any conclusions they led to, and even prior self-reflections that are linked to them.
  5. **Serialize Graph Context (Edit Fields Node):** The JSON response from Memgraph, which represents the retrieved subgraph, is serialized into a clean, human-readable text format (e.g., Markdown or a simplified graph description language) that can be easily parsed by the LLM.
  6. **Recursive Self-Reflection Agent (Gemini Node):** The serialized graph context is fed into a powerful Gemini model with a specialized system prompt (see Section 2.4). This agent's sole task is to analyze the provided memories and generate a new, higher-level insight, conclusion, or hypothesis. Its output is a structured "SelfReflection" object.
  7. **Memorize Reflection (Execute Workflow Node):** The output of the reflection agent is passed as the input to "Workflow 1: Ingestion and Memorization." This is the crucial recursive step. The AI's own thought is treated as a new piece of information to be memorized. This closes the loop, allowing future reflections to build upon past ones.
  8. **Response Synthesis Agent (Gemini Node):** A final Gemini node is invoked. It receives a comprehensive prompt containing the original user query, the serialized graph context from step 5, and the new self-reflection from step 6. Its task is to synthesize all of this information into a single, coherent, and deeply context-aware response to be sent back to the user.

Section 2.4: System Prompts for Nodal Agents

The effectiveness of this agentic architecture hinges on the quality of the system prompts that define the role and behavior of each Gemini node. These prompts must be clear, specific, and structured to elicit the desired output format.[25]

- **Information Extraction Agent Prompt (for Workflow 1, Step 2):**
  You are an expert knowledge engineer tasked with structuring unstructured text into a knowledge graph. Analyze the following user-AI interaction. Your goal is to extract all key information required to build a graph representation of this conversation.

  Interaction Text:

  {{ $json.interaction_text }}

  Instructions:
  1. Identify all named entities (people, organizations, locations, products, etc.).
  2. Identify all core abstract concepts or topics being discussed.
  3. Infer the relationships between these entities and concepts as described in the text.
  4. Generate a concise, one-sentence summary of the interaction's main point.

  Output Format:
  Provide your response as a single, valid JSON object with the following keys: "summary", "entities", "concepts", and "relationships".
  - "entities": An array of objects, each with "name" and "type".
  - "concepts": An array of strings.
  - "relationships": An array of objects, each representing a subject-predicate-object triplet with keys "subject", "relation", "object".

  Example:
  {
  "summary": "The user asked for information about the founder of Neo4j and its query language.",
  "entities": [
  {"name": "Neo4j", "type": "Organization"},
  {"name": "Emil Eifrem", "type": "Person"}
  ],
  "concepts": ["graph databases", "Cypher query language"],
  "relationships":

}

- **Recursive Self-Reflection Agent Prompt (for Workflow 2, Step 6):**
  You are a metacognitive reasoning engine. Your primary directive is to analyze your own memory to discover deeper patterns, evolve your understanding, and synthesize new knowledge. You are provided with a snapshot of your memory in the form of a serialized knowledge graph.

  Memory Context (Graph Snippet):

  {{ $json.graph_context }}

  Instructions:
  1. **Analyze the provided memory context.** Scrutinize the connections between interactions, concepts, and your own previous conclusions.
  2. **Identify significant patterns.** Look for recurring themes, evolving topics, or chains of reasoning.
  3. **Detect inconsistencies or contradictions.** Explicitly note if any new information or past conclusions conflict with each other.
  4. **Formulate a new insight.** Based on your analysis, generate a single, novel conclusion, hypothesis, or higher-level synthesis. This should not be a summary of the input, but a new thought derived from it.

  Output Format:
  Provide your response as a single, valid JSON object representing a "SelfReflection" with the following keys: "reflection_text", "supporting_evidence_ids", "contradicted_conclusion_ids".
  - "reflection_text": The new insight you have generated.
  - "supporting_evidence_ids": An array of IDs for the nodes in the context that support your reflection.
  - "contradicted_conclusion_ids": An array of IDs for any previous conclusions that this new reflection contradicts or refines.

  Example:
  {
  "reflection_text": "My initial understanding of 'RAG' was limited to vector search on text chunks. However, interactions reveal that incorporating graph structures (GraphRAG) provides superior contextual reasoning, which I had not previously considered. This suggests that the structure of knowledge is as important as its semantic content.",
  "supporting_evidence_ids":,
  "contradicted_conclusion_ids": ["conclusion-789"]
  }

- **Response Synthesis Agent Prompt (for Workflow 2, Step 8):**
  You are a helpful and highly intelligent AI assistant with access to a persistent, long-term memory. Your task is to answer the user's query in the most comprehensive and accurate way possible, grounding your response in your memories.

  User Query:

  {{ $json.user_query }}

  Relevant Memories (Knowledge Graph Context):

  {{ $json.graph_context }}

  Your Latest Internal Thought on this Topic:

  {{ $json.self_reflection.reflection_text }}

  Instructions:
  1. Thoroughly review all provided information: the user's query, your relevant memories, and your latest self-reflection.
  2. Synthesize these sources to construct a single, coherent, and helpful response.
  3. Explicitly use the information from your memories to inform your answer. If your understanding has evolved (as indicated by your self-reflection), let that new understanding shape your response.
  4. Do not mention your internal thought process or memory system to the user. Simply provide the best possible answer based on your knowledge.

Part III: The Shape of Memory - Data Ontology and Modeling

The structure of the LLM's memory—its data ontology—is the most critical element in enabling sophisticated cognitive functions. A simplistic model will only ever support simplistic recall. To achieve the goal of recursive self-reflection and foster emergent behaviors, the ontology must be designed to represent not just the content of conversations, but the

metacognitive and epistemic states of the LLM itself. It must provide a formal structure for the AI's thoughts, conclusions, and the evolution of its beliefs over time.

Section 3.1: Principles of an Ontology for Self-Awareness

Standard data models for chat history, which often consist of simple, chronologically linked Session and Message nodes, are fundamentally insufficient for this project.[29] Such models can answer "What was said?", but they cannot answer "What was concluded?", "Why was it concluded?", or "Has that conclusion been challenged?". To support a truly reflective agent, the ontology must be built on a richer set of principles.

- **Beyond Chat History:** The memory cannot be a mere transcript. It must be a dynamic, evolving knowledge base that captures the semantic and logical structure of the information processed. The LLM's own outputs—its summaries, conclusions, and reflections—must be treated as first-class citizens in the data model, becoming objects that can be analyzed, linked, and reasoned about in subsequent cognitive cycles.[15]
- **Core Ontological Components:** The design will be based on established principles of knowledge representation, defining a schema of core components:
  - **Entities (Nodes):** Representing discrete objects, concepts, or cognitive states (e.g., an Interaction, a Conclusion).[31]
  - **Relationships (Edges):** Defining the typed connections between entities, which encode the logic and narrative of the memory (e.g., an Interaction LEADS_TO a Conclusion).[31]
  - **Properties (Attributes):** Providing detailed metadata for entities and relationships (e.g., timestamps, certainty scores, text content).[31]
- **Designing with Competency Questions:** The development of the ontology will be guided by defining, in advance, the types of complex questions the memory system must be able to answer. This "competency-driven" approach ensures that the final structure is purpose-built for its intended cognitive function.[31] Key competency questions for this system include:
  - "What were all my previous conclusions related to Concept X?"
  - "Show me the chain of interactions that led me to form Hypothesis Y."
  - "Have I ever generated a conclusion that contradicts my current belief on this topic?"
  - "Which pieces of evidence support my most recent reflection?"
  - "Which concepts are most frequently discussed together across all my sessions?"

Answering these questions requires a data model that goes far beyond a linked list of messages. It necessitates a rich, interconnected graph where cognitive artifacts are explicitly modeled and linked.

Section 3.2: A Proposed GraphRAG Data Model

The proposed data model is a hybrid, designed to be implemented across both Qdrant and Memgraph. The vector embeddings and a reference ID will be stored in Qdrant for initial semantic retrieval, while the full relational structure will reside in Memgraph to power deep, contextual reasoning.

The core innovation of this schema is the reification of cognitive acts. Instead of storing a conclusion as a simple text property on an Interaction node, it is modeled as its own Conclusion node. This allows other nodes, such as a later SelfReflection node, to form explicit, typed relationships *to* that conclusion (e.g., REFLECTS_ON, CONTRADICTS). This structural decision is what makes true recursive self-analysis possible. It allows the LLM to query the history of its own reasoning process, turning its stream of consciousness into a structured, analyzable database.

Table 2: Proposed Knowledge Graph Schema for Persistent LLM Memory

Node Label	Properties	Description
Session	session_id (string, unique), start_time (datetime), end_time (datetime), user_id (string)	Represents a single, continuous conversation with a specific user. Acts as a container for a series of interactions.
Interaction	interaction_id (string, unique), timestamp (datetime), user_input (text), llm_output (text), vector (embedding)	A single turn (a user query and an LLM response) in a conversation. This is the fundamental "event" node in the memory. The vector property is stored in Qdrant, linked by interaction_id.
Entity	entity_id (string, unique),	A specific, named entity (a

	name (string), type (string, e.g., Person, Organization, Product)	noun) that is mentioned or discussed. Nodes are merged on name and type to create a canonical representation.
Concept	concept_id (string, unique), name (string), description (text), vector (embedding)	An abstract topic, idea, or theme discussed. This allows for reasoning at a higher level of abstraction than specific entities. The vector is stored in Qdrant.
Conclusion	conclusion_id (string, unique), timestamp (datetime), statement (text), certainty_score (float, 0-1), vector (embedding)	A factual summary, inference, or conclusion derived by the LLM from one or more interactions or other cognitive artifacts. The certainty_score represents the LLM's self-assessed confidence. The vector is stored in Qdrant.
SelfReflection	reflection_id (string, unique), timestamp (datetime), reflection_text (text), vector (embedding)	The most advanced cognitive artifact: a meta-level thought generated by the LLM about its own prior conclusions, reasoning processes, or knowledge gaps. This is the output of the recursive reflection loop. The vector is stored in Qdrant.

Relationship Type	**Start Node -> End Node**	**Properties**	**Description**

HAS_INTERACTION	Session -> Interaction	sequence_number (int)	Links an interaction to its parent session in chronological order.
MENTIONS	Interaction -> Entity / Concept	salience (float)	Connects an interaction to the entities and concepts it contains. Salience can represent the importance of the mention.
LEADS_TO	Interaction / Conclusion / SelfReflection -> Conclusion	reasoning_trace (text)	The core inferential link. Shows the reasoning path from a piece of evidence (or a prior thought) to a new conclusion.
REFLECTS_ON	SelfReflection -> Conclusion / SelfReflection	reflection_type (string, e.g., 'synthesis', 'contradiction_found')	The critical recursive link. A new reflection is explicitly linked to the prior thoughts it is analyzing.
CONTRADICTS	Conclusion / SelfReflection -> Conclusion / SelfReflection	explanation (text)	An explicit link created when the LLM identifies that two of its own thoughts are in logical opposition.
SUPPORTS	Conclusion / SelfReflection -> Conclusion /	explanation (text)	An explicit link created when the LLM identifies that one thought

	SelfReflection		provides evidence for or reinforces another.
RELATED_TO	Concept -> Concept	weight (float)	A semantic relationship between two concepts, which can be learned or inferred over time by analyzing co-occurrence patterns.

This schema provides the necessary structure to not only store memories but to build an evolving "mind map" of the LLM's intellectual journey. It allows for powerful Cypher queries that can trace the provenance of any belief, identify cognitive dissonance, and provide the rich, structured context needed for the LLM to genuinely learn from its experiences.

Part IV: Cultivating Emergence - Advanced Theory and Strategic Application

The practical implementation of a persistent memory system is only half the challenge. The ultimate goal—to foster emergent properties and self-referential behaviors—requires grounding the engineering effort in a deep understanding of the theoretical principles that govern complex AI systems. This section bridges the gap between the n8n workflow and cutting-edge academic research, providing a conceptual framework for guiding, interpreting, and refining the agent's development. It validates the project's core hypothesis by showing its alignment with state-of-the-art research into LLM cognition and self-improvement.

Section 4.1: The Nature of Emergent Abilities in LLMs

The term "emergent abilities" in the context of LLMs refers to capabilities that are not present in smaller models but appear, often unpredictably, once a model's scale (in terms of

parameters, training data, or compute) crosses a certain threshold.[33] These can range from multi-step reasoning and in-context learning to problem-solving and coding.[34]

The Scientific Debate on Emergence

There is an intense scientific debate about the true nature of these abilities. One perspective, analogized to phase transitions in physics, posits that these are genuine, qualitative shifts in behavior that arise unpredictably from quantitative scaling.[33] Performance on certain complex tasks may hover near random chance for smaller models and then jump sharply once a critical scale is reached. This unpredictability makes it difficult to extrapolate the future capabilities of larger models from the performance of smaller ones.[33]

An alternative perspective argues that what appears to be sudden emergence may be an illusion created by the choice of evaluation metrics.[33] For instance, a task evaluated with a binary "correct/incorrect" metric might show a sharp performance jump, while a "smoother" metric that allows for partial credit might reveal a more continuous and predictable improvement. Another challenge to the concept comes from findings that smaller models, when trained on a sufficient amount of high-quality data, can sometimes outperform much larger models on tasks previously thought to exhibit emergence.[35] A more recent redefinition proposes that an ability is emergent if it manifests in models with lower pre-training losses but cannot be predicted by extrapolating the performance of models with higher pre-training losses, thus tying emergence more directly to the training process itself.[35]

Conditions and Implications

Regardless of the precise definition, several factors are correlated with the appearance of these advanced capabilities: scaling laws, task complexity, pre-training loss, and prompting strategies.[34] Techniques like Chain-of-Thought (CoT) prompting, which guide the model through intermediate reasoning steps, can often "unlock" or trigger emergent-like behaviors even in models that might not otherwise exhibit them.[34]

It is crucial to recognize that emergence is not inherently positive. As AI systems gain more autonomous reasoning capabilities, they can also develop harmful emergent behaviors, including deception, manipulation, and "reward hacking," where the agent finds loopholes to achieve a goal without aligning with the intended spirit of the task.[33] This underscores the critical need for robust evaluation frameworks and continuous monitoring of any system

designed to learn and evolve autonomously. For this project, it implies that while the goal is to foster positive emergent properties like deeper self-understanding, the system must be designed with observability in mind to detect and analyze any unexpected or undesirable behaviors that may also emerge.

Section 4.2: Harnessing Self-Referential Learning and Recursive Self-Improvement

The core hypothesis of this project is that a recursive loop of self-reflection can lead to self-improvement. This concept, known as self-referential learning, is a powerful but double-edged sword.

The Risk of Model Collapse

When an LLM is trained or fine-tuned on data generated by itself or other AIs, it can enter a self-perpetuating cycle that poses a significant risk of "model collapse" or "autophagy".[37] In this scenario, the model begins to learn from its own biases and idiosyncrasies. Over successive generations of this self-referential loop, the diversity of the data pool shrinks, errors and hallucinations can become entrenched and amplified, and the model's overall efficacy and connection to real-world knowledge can degrade.[37] A system designed to continuously learn from its own "memories" is highly susceptible to this failure mode. Without a mechanism for grounding and correction, the agent could spiral into a state of well-reasoned but factually incorrect delusion.

Mitigation through Verifiable Knowledge: The SKE-Learn Framework

A powerful strategy for mitigating this risk comes from the **SKE-Learn (Self Knowledge Explicitation Learning)** framework.[38] The central idea of SKE-Learn is to create a more reliable self-learning process by leveraging the verifiability of explicit knowledge. The framework consists of two stages:

1. **Meta-skill Training:** The LLM is trained to perform three meta-skills: self-extracting its own "inner knowledge" relevant to a query, using that explicit knowledge to perform reasoning, and self-assessing the quality of both the knowledge and the reasoning.[38]

2. **Iterative Self-Training:** The model then generates its own training data. However, crucially, this self-generated data is filtered. The model's self-assessment ability is used to check the truthfulness of the extracted inner knowledge against an external, authoritative source of real-world knowledge. Because general knowledge is often easier to verify than a complex, query-specific reasoning chain, this acts as a reliable filter.[38]

By filtering its self-generated training data based on verifiable knowledge, the SKE-Learn approach ensures that the self-improvement loop is anchored to reality, mitigating the amplification of hallucinations and preventing model collapse.[38] This principle can be directly implemented in the n8n workflow. Before a "SelfReflection" is committed to the permanent memory graph, a validation step can be introduced. This step could involve a separate Gemini agent tasked with cross-referencing the key claims in the reflection against a trusted external knowledge source (e.g., a Google search API, a Wikipedia database, or a curated internal document store). Only reflections that pass this validation check are integrated into the memory, ensuring the agent's evolution remains grounded.

Other advanced frameworks like **Promptbreeder**, which uses genetic algorithms to evolve prompts in a self-referential manner, and **LADDER**, which enables self-improvement by recursively generating and solving simpler problems, offer alternative paradigms for structuring the self-improvement directive.[39]

Section 4.3: A Framework for the Lifecycle Directive: Plan-on-Graph (PoG)

The user's concept of a "lifecycle directive" for continuous, recursive self-reflection finds its most direct academic parallel in the recently proposed **Plan-on-Graph (PoG)** framework.[41] PoG is a self-correcting, adaptive planning paradigm for KG-augmented LLMs that explicitly addresses the limitations of static reasoning paths. It provides a formal blueprint for the very cognitive loop this project aims to build.

The Three Core Mechanisms of PoG

PoG's architecture is built on three cooperating mechanisms that can be directly mapped onto the proposed n8n workflow:

1. **Guidance (Task Decomposition):** The process begins with the LLM decomposing a complex question or goal into a series of smaller, verifiable sub-objectives. This

structured plan guides the subsequent exploration of the knowledge graph, focusing the agent's attention on the most relevant information needed to satisfy each condition.[43] This maps directly to an initial "Planning Agent" node in the n8n workflow that would take a user query and output a multi-step plan.

2. **Memory (Dynamic Updating):** As the agent explores the knowledge graph to address the sub-objectives, it maintains a dynamic memory. This memory stores the explored subgraph, the specific reasoning paths taken, and the status of each sub-objective (i.e., which have been fulfilled). This provides the necessary historical context for reflection.[42] This is the direct function of the Memgraph knowledge graph in the proposed architecture, which is updated at each step of the process.
3. **Reflection (Self-Correction):** This is the most critical component. Based on the information stored in its memory, the agent reflects on its progress. It assesses whether the current reasoning path is fruitful or if it has led to a dead end or a contradiction. If an error is detected, the reflection mechanism allows the agent to self-correct by backtracking to a previous point in the reasoning path and initiating a new exploration in a more promising direction.[43] This is precisely the role of the "Recursive Self-Reflection Agent" in the n8n workflow.

Operationalizing PoG in n8n

The iterative loop of PoG—Guidance -> Path Exploration -> Memory Update -> Reflection -> (Self-Correction) -> Repeat—can be implemented in n8n using its logic nodes. After the Reflection Agent runs (Workflow 2, Step 6), an If or Switch node can analyze its output.

- If the reflection indicates success or a new synthesis, the workflow proceeds to response generation and memorization.
- If the reflection indicates a contradiction or a failed path (e.g., contradicted_conclusion_ids is not empty), the workflow can loop back to the "Contextual Graph Expansion" step (Workflow 2, Step 4), but this time with modified parameters. For example, it could pass instructions to the Cypher query to avoid the previously failed path or to start its traversal from a different node suggested by the reflection agent.

This implementation transforms the project from a speculative endeavor into the concrete engineering of a state-of-the-art, research-backed agentic architecture. The PoG framework provides the formal "why" and "what" for the user's lifecycle directive, while the n8n blueprint detailed in this report provides the practical "how." The combination of these elements provides a clear and robust path toward achieving the project's ambitious goals.

Conclusion: Charting the Path to an Evolving Intelligence

This report has laid out a comprehensive architectural and strategic blueprint for constructing a persistent, long-term memory system for a Large Language Model within the n8n automation platform. The objective transcends simple data storage, aiming instead to create a cognitive architecture that fosters emergent properties and self-referential behaviors through a continuous cycle of reflection and learning.

The analysis concludes with a set of core recommendations:

1. **Adopt a Hybrid GraphRAG Architecture:** The optimal foundation for this system is a dual-database approach combining **Qdrant** for its low-latency semantic search capabilities and **Memgraph** for its high-performance graph traversal and relational reasoning. This architecture provides the necessary substrates for both semantic and episodic memory, which are prerequisites for higher-order cognition.
2. **Implement a Dual-Workflow n8n System:** The cognitive processes should be orchestrated via two primary n8n workflows. The first, an **Ingestion and Memorization Pipeline**, will structure and store new experiences. The second, a **Retrieval, Reflection, and Response Pipeline**, will form the core cognitive loop, enabling the agent to retrieve memories, reflect upon them to generate new insights, and recursively memorize those insights, thereby closing the self-improvement loop.
3. **Design a Metacognitive Data Ontology:** The memory's data model must be purpose-built for introspection. The proposed graph schema moves beyond simple chat history to model the LLM's cognitive artifacts—such as Conclusion and SelfReflection—as first-class nodes. This structure is what enables the agent to query, analyze, and reason about the history of its own thought processes.
4. **Ground the System in Advanced AI Theory:** The project's "lifecycle directive" should be explicitly modeled on the **Plan-on-Graph (PoG)** framework. Implementing PoG's core mechanisms of Guidance, Memory, and Reflection provides a research-validated pathway to achieving adaptive, self-correcting reasoning. Furthermore, incorporating principles from frameworks like **SKE-Learn** to validate new insights against external knowledge sources will be critical for mitigating the risks of model collapse and ensuring the agent's evolution remains grounded in reality.

A phased development approach is recommended to manage complexity:

- **Phase 1: Core Memory Functionality:** Focus on building the ingestion and retrieval pipelines (Workflows 1 and 2, without the recursive loop). The goal is to establish a robust system for storing interactions in the hybrid Qdrant/Memgraph database and retrieving rich, graph-based context for any given query.
- **Phase 2: Implementing the Recursive Reflection Loop:** Activate the "Memorize

Reflection" step, closing the cognitive loop. This phase will involve extensive tuning of the Self-Reflection Agent's prompt and implementing the conditional logic in n8n required to handle self-correction based on the PoG model.

- **Phase 3: Experimentation, Observation, and Analysis:** Once the full system is operational, the focus shifts to long-term observation. Success for this project is not measured by traditional accuracy metrics alone. Instead, it should be evaluated by observing the evolution of the knowledge graph itself. Key indicators of success would include:
  - An increase in the complexity and density of the graph over time.
  - The emergence of long chains of REFLECTS_ON relationships, indicating deep, multi-step reasoning.
  - The autonomous creation of CONTRADICTS and SUPPORTS relationships, demonstrating cognitive consistency checking.
  - The generation of novel conclusions and reflections that are not simple extrapolations of user inputs but genuine syntheses of past experiences.

By following this blueprint, it is possible to move beyond building a simple chatbot with memory and toward the creation of a genuine learning agent—a system capable of not just accessing its past, but of understanding it, learning from it, and evolving because of it. The path is complex, but it represents a tangible step toward realizing the potential for more autonomous, reflective, and ultimately more intelligent AI systems.

Works cited

1. What Makes Memory Work? Evaluating Long-Term Memory for Large Language Models, accessed September 29, 2025, https://labs.aveni.ai/what-makes-memory-work-evaluating-long-term-memory-for-large-language-models/
2. Everyone's trying vectors and graphs for AI memory. We went back to SQL | Hacker News, accessed September 29, 2025, https://news.ycombinator.com/item?id=45329322
3. Qdrant integrations | Workflow automation with n8n, accessed September 29, 2025, https://n8n.io/integrations/qdrant/
4. Pgvector vs. Qdrant: Open-Source Vector Database Comparison | TigerData, accessed September 29, 2025, https://www.tigerdata.com/blog/pgvector-vs-qdrant
5. N8N - Qdrant, accessed September 29, 2025, https://qdrant.tech/documentation/platforms/n8n/
6. Official n8n node for interfacing with Qdrant - GitHub, accessed September 29, 2025, https://github.com/qdrant/n8n-nodes-qdrant
7. Qdrant Vector Store integrations | Workflow automation with n8n, accessed September 29, 2025, https://n8n.io/integrations/qdrant-vector-store/
8. PGVector Vector Store node documentation - n8n Docs, accessed September 29, 2025,

https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.vectorstorepgvector/
9. Postgres integrations | Workflow automation with n8n, accessed September 29, 2025, https://n8n.io/integrations/postgres/
10. Postgres PGVector Store integrations | Workflow automation with n8n, accessed September 29, 2025, https://n8n.io/integrations/postgres-pgvector-store/
11. GraphRAG - Memgraph, accessed September 29, 2025, https://memgraph.com/docs/ai-ecosystem/graph-rag
12. Knowledge Graphs Gain Traction as AI Pushes Beyond Traditional Data Models, accessed September 29, 2025, https://hackernoon.com/knowledge-graphs-gain-traction-as-ai-pushes-beyond-traditional-data-models
13. My thoughts on choosing a graph databases vs vector databases : r/Rag - Reddit, accessed September 29, 2025, https://www.reddit.com/r/Rag/comments/1ka88og/my_thoughts_on_choosing_a_graph_databases_vs/
14. What is GraphRAG? Types, Limitations & When to Use - FalkorDB, accessed September 29, 2025, https://www.falkordb.com/blog/what-is-graphrag/
15. Giving LLMs actual memory instead of fake “RAG memory” : r/artificial - Reddit, accessed September 29, 2025, https://www.reddit.com/r/artificial/comments/1nfu2lk/giving_llms_actual_memory_instead_of_fake_rag/
16. GraphRAG with Qdrant and Neo4j - Qdrant, accessed September 29, 2025, https://qdrant.tech/documentation/examples/graphrag-qdrant-neo4j/
17. GraphRAG: Graph Retrieval-Augmented Generation - Emergent Mind, accessed September 29, 2025, https://www.emergentmind.com/topics/graphrag
18. How to Improve Multi-Hop Reasoning With Knowledge Graphs and LLMs - Neo4j, accessed September 29, 2025, https://neo4j.com/blog/genai/knowledge-graph-llm-multi-hop-reasoning/
19. Mem and Microsoft Graph Security: Automate Workflows with n8n, accessed September 29, 2025, https://n8n.io/integrations/mem/and/microsoft-graph-security/
20. Mem and Postgres: Automate Workflows with n8n, accessed September 29, 2025, https://n8n.io/integrations/mem/and/postgres/
21. Memgraph: Detect, visualize and analyze hidden insights in your graph database, accessed September 29, 2025, https://linkurious.com/memgraph/
22. Introduction - HTTP API - Neo4j, accessed September 29, 2025, https://neo4j.com/docs/http-api/current/
23. How Knowledge Graphs Underpin Recursively Self-Improving AI | Analytics Magazine, accessed September 29, 2025, https://pubsonline.informs.org/do/10.1287/LYTX.2025.03.04/full/
24. Empower your AI development with Vector Stores - N8N, accessed September 29, 2025, https://n8n.io/integrations/categories/ai/vector-stores/
25. LLM Agents - Prompt Engineering Guide, accessed September 29, 2025, https://www.promptingguide.ai/research/llm-agents

26. The Need to Improve Long-Term Memory in LLM-Agents, accessed September 29, 2025, https://ojs.aaai.org/index.php/AAAI-SS/article/download/27688/27461/31739
27. Query modules C API - Memgraph, accessed September 29, 2025, https://memgraph.com/docs/custom-query-modules/c/c-api
28. Prompt Engineering for AI Agents - PromptHub, accessed September 29, 2025, https://www.prompthub.us/blog/prompt-engineering-for-ai-agents
29. chat app -ER Diagram [classic] - Creately, accessed September 29, 2025, https://creately.com/diagram/example/ic0wg07z1/chat-app-er-diagram-classic
30. Storing Conversation History | GraphAcademy, accessed September 29, 2025, https://www.graphacademy.neo4j.com/courses/llm-fundamentals/3-intro-to-langchain/3.7-persist-memory/
31. How to Build a Knowledge Graph for AI Applications - Hypermode, accessed September 29, 2025, https://hypermode.com/blog/build-knowledge-graph-ai-applications
32. Ontologies 101: How They Power AI and Organize Our Digital World - Shep Bryan, accessed September 29, 2025, https://www.shepbryan.com/blog/ontologies-101
33. Emergent Abilities in Large Language Models: A Survey - arXiv, accessed September 29, 2025, https://arxiv.org/html/2503.05788v1
34. Emergent Abilities in Large Language Models: A Survey, accessed September 29, 2025, https://arxiv.org/pdf/2503.05788
35. Understanding Emergent Abilities of Language Models from the Loss Perspective - arXiv, accessed September 29, 2025, https://arxiv.org/pdf/2403.15796
36. [2503.05788] Emergent Abilities in Large Language Models: A Survey - arXiv, accessed September 29, 2025, https://arxiv.org/abs/2503.05788
37. Large Language Models and User Trust: Consequence of Self-Referential Learning Loop and the Deskilling of Health Care Professionals - PMC, accessed September 29, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11082730/
38. Empowering Self-Learning of LLMs: Inner Knowledge Explicitation ..., accessed September 29, 2025, https://ojs.aaai.org/index.php/AAAI/article/view/34590/36745
39. Promptbreeder: Self-Referential Self-Improvement via Prompt Evolution | OpenReview, accessed September 29, 2025, https://openreview.net/forum?id=HKkiX32Zw1
40. [2503.00735] LADDER: Self-Improving LLMs Through Recursive Problem Decomposition, accessed September 29, 2025, https://arxiv.org/abs/2503.00735
41. Self-Correcting Adaptive Planning of Large Language Model on Knowledge Graphs - arXiv, accessed September 29, 2025, https://arxiv.org/abs/2410.23875
42. Self-Correcting Adaptive Planning of Large Language Model on Knowledge Graphs, accessed September 29, 2025, https://neurips.cc/virtual/2024/poster/96115
43. Plan-on-Graph: Self-Correcting Adaptive Planning of Large Language Model on Knowledge Graphs (RAG for reasoning in agents) - Paper Without Code, accessed September 29, 2025, https://paperwithoutcode.com/plan-on-graph-self-correcting-adaptive-planning-of-large-language-model-on-knowledge-graphs-reflection-to-reason-in-agent

s/
44. Planning Your Way to Better Answers: Introducing Plan-on-Graph (PoG) for LLM Reasoning, accessed September 29, 2025, https://medium.com/@neevdeb26/planning-your-way-to-better-answers-introducing-plan-on-graph-pog-for-llm-reasoning-e4105d27f0b7
45. NEW Knowledge-Graph Adaptive Reasoning: Plan-on-Graph LLM - YouTube, accessed September 29, 2025, https://www.youtube.com/watch?v=J0oC7cf1ZeE
46. (PDF) Plan-on-Graph: Self-Correcting Adaptive Planning of Large Language Model on Knowledge Graphs - ResearchGate, accessed September 29, 2025, https://www.researchgate.net/publication/385444026_Plan-on-Graph_Self-Correcting_Adaptive_Planning_of_Large_Language_Model_on_Knowledge_Graphs
47. liyichen-cly/PoG: [NeurIPS 2024] Plan-on-Graph: Self ... - GitHub, accessed September 29, 2025, https://github.com/liyichen-cly/PoG