cognee

cognee

Building Memory for AI Agents

120 000
Runs per month

100 000+
Library Downloads

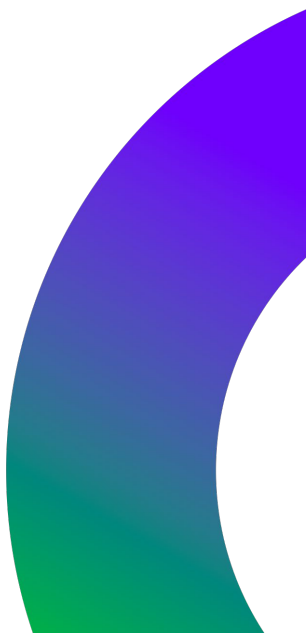
7000+
Github stars

Who I am?

B/S/H/

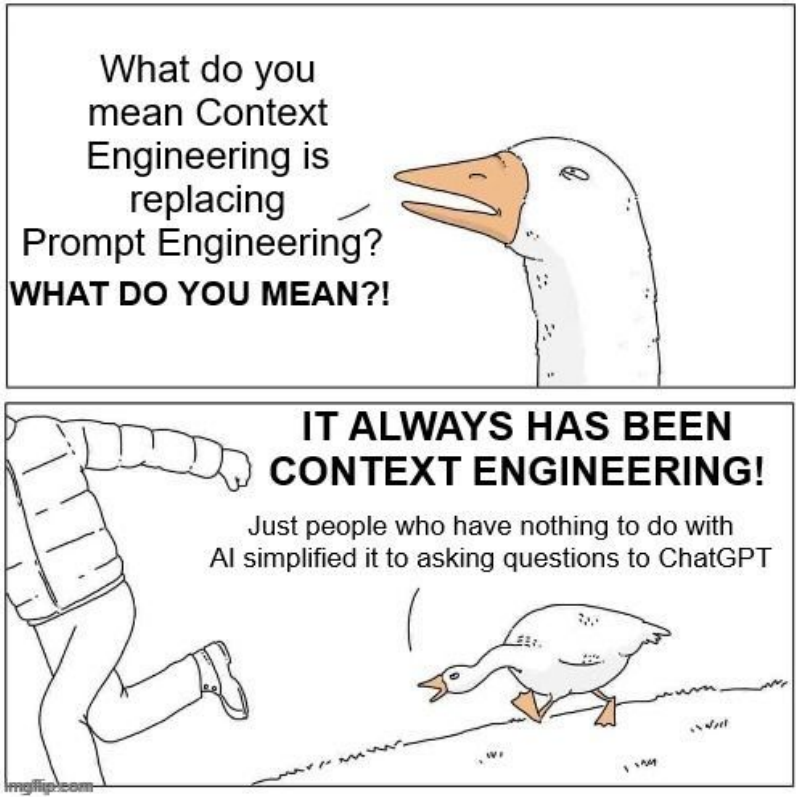
Agenda

- Agent → Query → ??? → Context → Response
- Building Structured Memory
- Closer Look: Retrieval
- Demo
- Closing

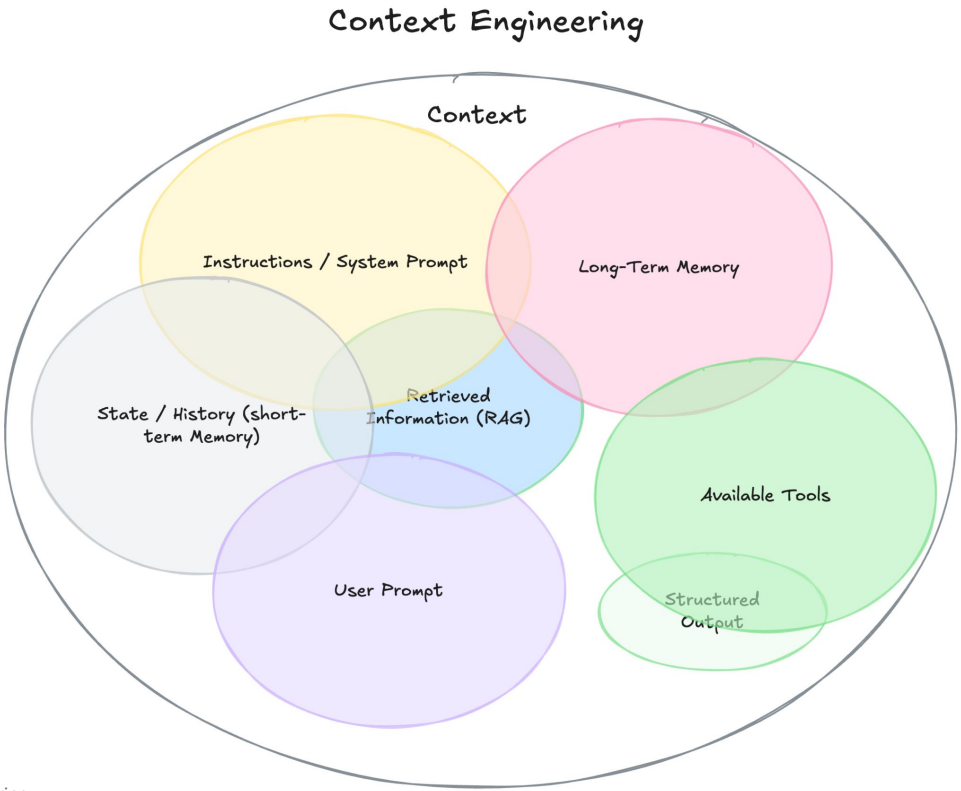


Agent → Query → ??? → Context → Response

Agent → Query → ??? → Context → Response

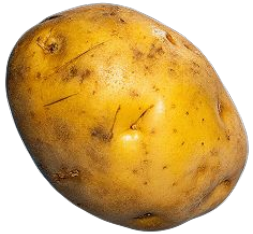


Agent → Query → ??? → Context → Response



https://www.philschmid.de/context-engineering

Agent → Query → Memory → Context → Response



answer relevancy

Ask RAG Solution

92.5%

answer relevancy

Ask Cognee

answer relevancy

Ask ChatGPT

https://www.cognee.ai/blog/deep-dives/ai-memory-evals-0825

AI Agents needs memory

- Documents contain interconnected information

- Traditional search treats content as isolated chunks

- Relationships between concepts get lost

- AI systems need connected context, not fragmented pieces

Building Structured Memory

- Ingest from 30+ data sources
- Structured, unstructured, semistructured data

- Python functions as tasks and pipelines
- Adapters for vector stores and graph databases

- Memory enhancement layer
- Feedback mechanisms
- Retrieval with reasoning

- Evaluation module
- Hyperparameter optimization
- Custom tasks

Read our paper

Building Structured Memory

Ingestion
Ontology Extraction
Batch Loading

- Ingest from 30+ data sources
- Structured, unstructured, semistructured data

Graph Extraction
Entity Extraction
Graph Structure Definition

- Python functions as tasks and pipelines
- Adapters for vector stores and graph databases

Graph Enrichment
Entity Management
Graph Reasoning

- Memory enhancement layer
- Feedback mechanisms
- Retrieval with reasoning

Evaluation
Ontology Validation
Benchmark Evaluation

- Evaluation module
- Hyperparameter optimization
- Custom tasks

Read our paper

Looking into the Retriever Stack - Our Learnings

Modular structure = easy to reason about and easy to extend for custom retrieval

two-step contract: get_context(query) →get_completion(query, context).

```
class BaseRetriever(ABC):
    @abstractmethod
    async def get_context(self, query: str, **kwargs) -> Any:
        """Fetch raw data relevant to the query."""
        pass

    @abstractmethod
    async def get_completion(self, query: str, context: Any, **kwargs) -> Any:
        """Process the context to produce a final result."""
        pass
```

Looking into Retriever Stack - Our Learnings

Method	What it does:	Use when:	Don’t use if:
Chunks Retriever	Vector search over paragraph-sized text chunks; returns the original text as-is.	You need specific facts straight from the source text.	…you want a synthesized explanation or cross-chunk reasoning.
Completion Retriever	Retrieves top-matching chunks, then feeds them to an LLM (with prompt templates) to generate a coherent answer.	You want clear explanations from source material.	.. relationships are crucial to the answer, or you only need raw snippets.
GraphCompletion Retriever	Starts with vector hits (chunks/summaries/entities), builds a small related subgraph (nodes + edges), serializes it to text, then asks an LLM to answer using that structure.	Relationships/context across entities matter, not just matching text.	…a simple fact lives in a single passage and graph context adds no value, or you have no graph :)
GraphCompletion Cot Retriever	Builds on GraphCompletion with chain-of-thought style iterative reasoning: draft → self-check → propose follow-up → retrieve more → refine, over several rounds.	Complex, multi-hop questions needing explicit, iterative reasoning.	..the query is simple-hop—RAG or basic GraphCompletion will do.
GraphCompletion ContextExtension Retriever	Like GraphCompletion, but iteratively extends the context: LLM suggests where to look next; repeat a few rounds to broaden coverage in the graph before answering.	The question needs layered/indirect context or broader coverage than a single pass.	…a one-pass graph view already answers the question (use GraphCompletion instead).

Others: Summaries, Code, Cypher, NaturalLanguage, ChunksLexical, CodingRules, Feedback, Temporal, FeelingLucky

https://www.cognee.ai/blog/deep-dives/the-art-of-intelligent-retrieval-unlocking-the-power-of-search

Looking into Retriever Stack - Our Learnings

Method	What it does:	Use when:	Don't use if:
Chunks Retriever	Vector search over paragraph-sized text chunks; returns the original text as-is.	You need specific facts straight from the source text.	…you want a synthesized explanation or cross-chunk reasoning.
Completion Retriever	Retrieves top-matching chunks, then feeds them to an LLM (with prompt templates) to generate a coherent answer.	You want clear explanations from source material.	.. relationships are crucial to the answer, or you only need raw snippets.
GraphCompletion Retriever	Starts with vector hits (chunks/summaries/entities), builds a small related subgraph (nodes + edges), serializes it to text, then asks an LLM to answer using that structure.	Relationships/context across entities matter, not just matching text.	…a simple fact lives in a single passage and graph context adds no value, or you have no graph :)
GraphCompletion Cot Retriever	Builds on GraphCompletion with chain-of-thought style iterative reasoning: draft → self-check → propose follow-up → retrieve more → refine, over several rounds.	Complex, multi-hop questions needing explicit, iterative reasoning.	..the query is simple-hop—RAG or basic GraphCompletion will do.
GraphCompletion ContextExtension Retriever	Like GraphCompletion, but iteratively extends the context: LLM suggests where to look next; repeat a few rounds to broaden coverage in the graph before answering.	The question needs layered/indirect context or broader coverage than a single pass.	…a one-pass graph view already answers the question (use GraphCompletion instead).

Others: Summaries, Code, Cypher, NaturalLanguage, ChunksLexical, CodingRules, Feedback, Temporal, FeelingLucky

https://www.cognee.ai/blog/deep-dives/the-art-of-intelligent-retrieval-unlocking-the-power-of-search

Two Parallel Worlds During Data Ingestion

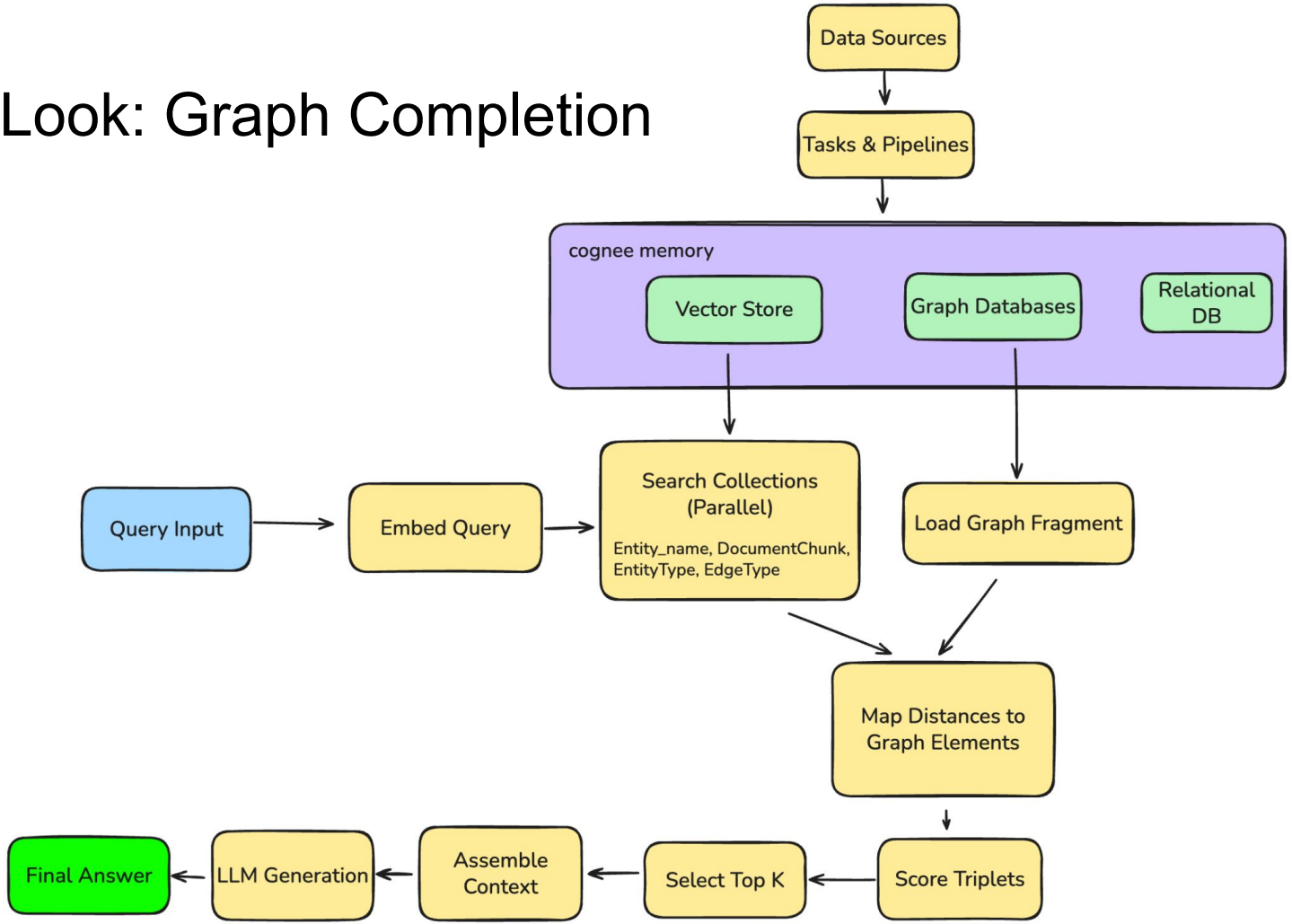
World 1: Graph Space (Relationship Structure)

- Entities and concepts are identified
- Connections between them are preserved
- Forms a web of knowledge
- Enables "how does X relate to Y"

World 2: Vector Space (Semantic Similarity)

- Text is converted to numerical embeddings
- Each piece of content becomes a point in high-dimensional space
- Similar meanings cluster together
- Enables "find things that mean something like X"

Closer Look: Graph Completion



The Hybrid Architecture Advantage- Best of Both Worlds

Vector Search Provides:

✅ Semantic similarity

✅ Fuzzy matching

✅ Language understanding

❌ No relationship awareness

❌ No reasoning paths

➕

Graph Traversal Provides:

✅ Relationship preservation

✅ Multi-hop reasoning

✅ Causal chains

❌ No semantic similarity

❌ Exact matches only

Combined Approach:

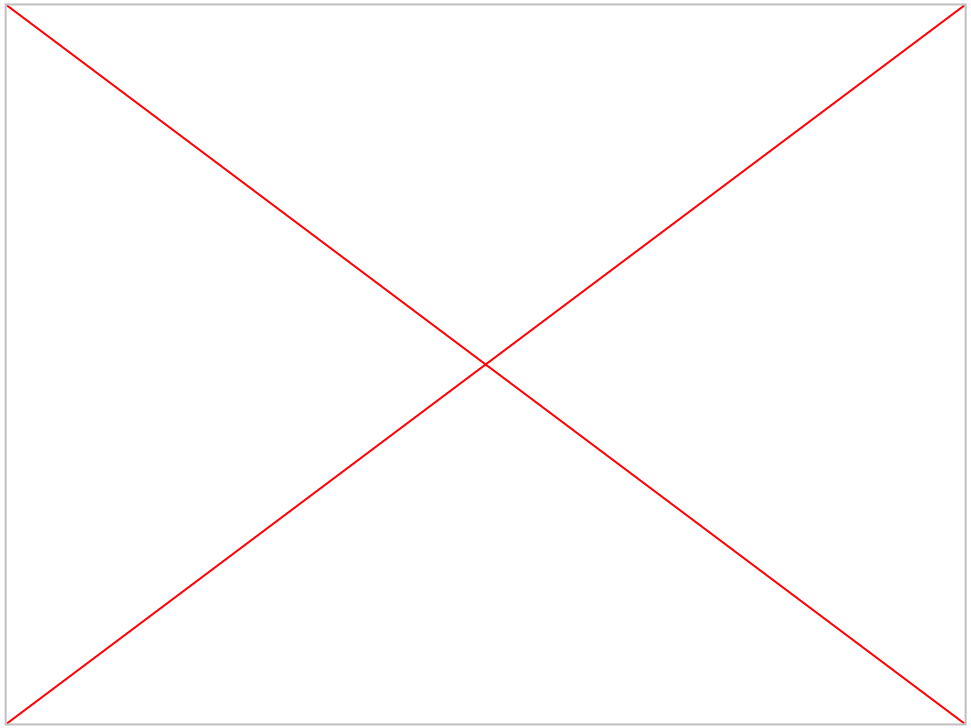
✅ Semantic similarity

✅ Relationship preservation

✅ Multi-hop reasoning

✅ Ranked by relevance

DEMO!



Memory Pattern Anti-Patterns

Common Pitfalls to Avoid

1. **Graph Everything Syndrome:** Not every use case needs a knowledge graph
2. **Chunk Size Paralysis:** Obsessing over perfect chunk boundaries
3. **Embedding Model FOMO:** Latest ≠ Best for your use case
4. **Context Window Maximalism:** More context ≠ Better answers

cognee is open source!

Check out the repo on GitHub
&
try it yourself

Join the community
&
ask us questions