GraphReader Agentic RAG: Rethinking Long-Context Retrieval Systems

28 June, 2025 | Nokia Bangalore

Register Now

Jayita Bhattacharyya

Data Scientist,
Deloitte

Soumya Ranjan Das

Data Scientist,
Deloitte

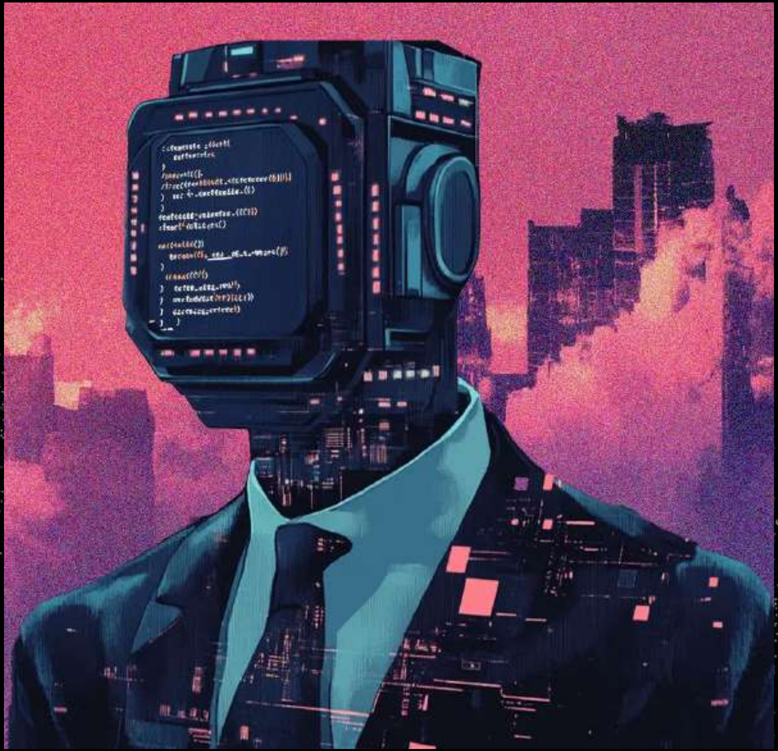
GraphReader Agentic RAG Rethinking for Long-Context Retrieval Systems

Presented by

Jayita Bhattacharyya

Soumya Ranjan Das

Contents



1 Traditional RAG Challenges

2 GraphReader Intro

3 GraphRAG + Agents

4 Graph Construction

5 Retrieval & Traversal

6 Demo

What problems do you face while building search & retrieval systems?

Challenges with Normal RAG Approaches

1

Context Window & lost-in-the-middle
Limitations on complex and multi hop queries.

* certain work have been done to address like MoE, RoPE, KV Caching, training data costs, etc

* Adaptive RAG, Self Corrective RAG, etc

2

State & Persistence Issues
Lack of autonomous exploration and optimized efforts to give structured answers.

* Agentic Architecture helps maintain states between tool calling and agent to agent communication

3

Knowledge Graph Schema
Creating takes domain expertise & graph modelling for better search & retrieval

* Domain specific use cases need SMEs & architects to define type of nodes and relationships

What value does GraphReader bring?

graph-based agent system designed to supercharge when dealing with long and complex texts. Instead of reading documents linearly, it restructures them into a graph of atomic facts and relationships, then uses an intelligent agent to explore that graph in a goal-directed way

Lets learn more

GraphReader - The Healer

1

Chunking Structure

breaks semantic relationships between entities, especially across paragraphs or documents.

2

Shallow Retrieval

goal-directed traversal of the graph, allowing it to follow meaningful paths through related concepts

3

Planning/Reflection

planning, note-taking, and reflection, mimicking how a human researcher would explore a topic.

4

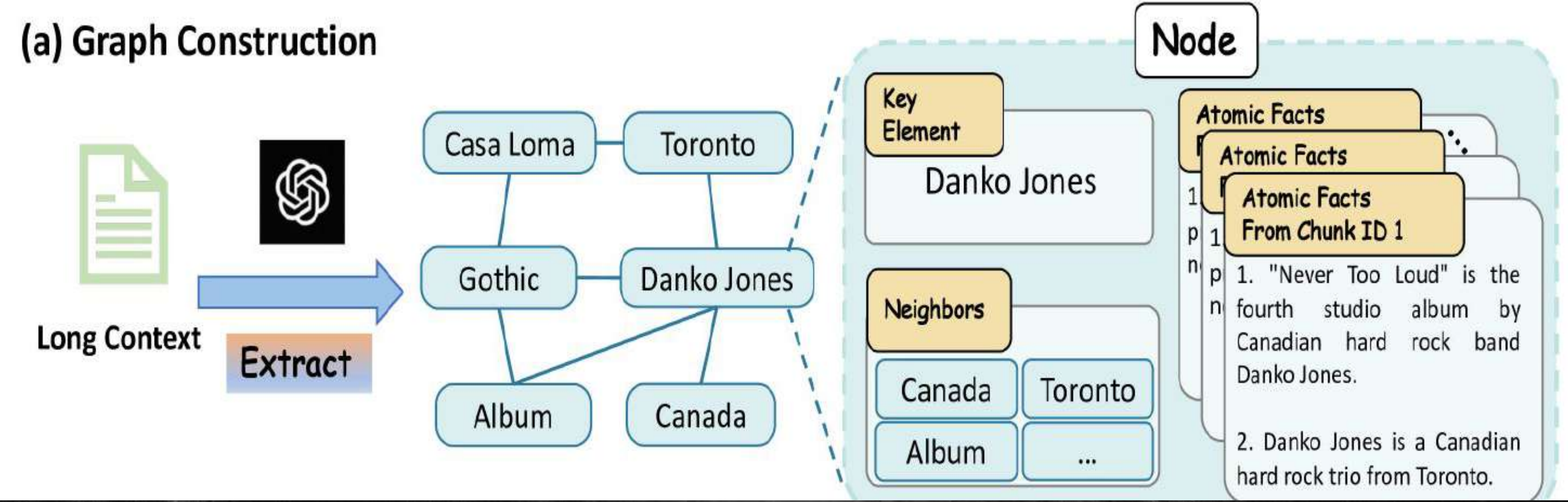
Context Window

coarse-to-fine strategy, starting with summaries and only diving into full content when necessary — achieving high recall with just a 4k window.

5

Multi-Hop Reasoning

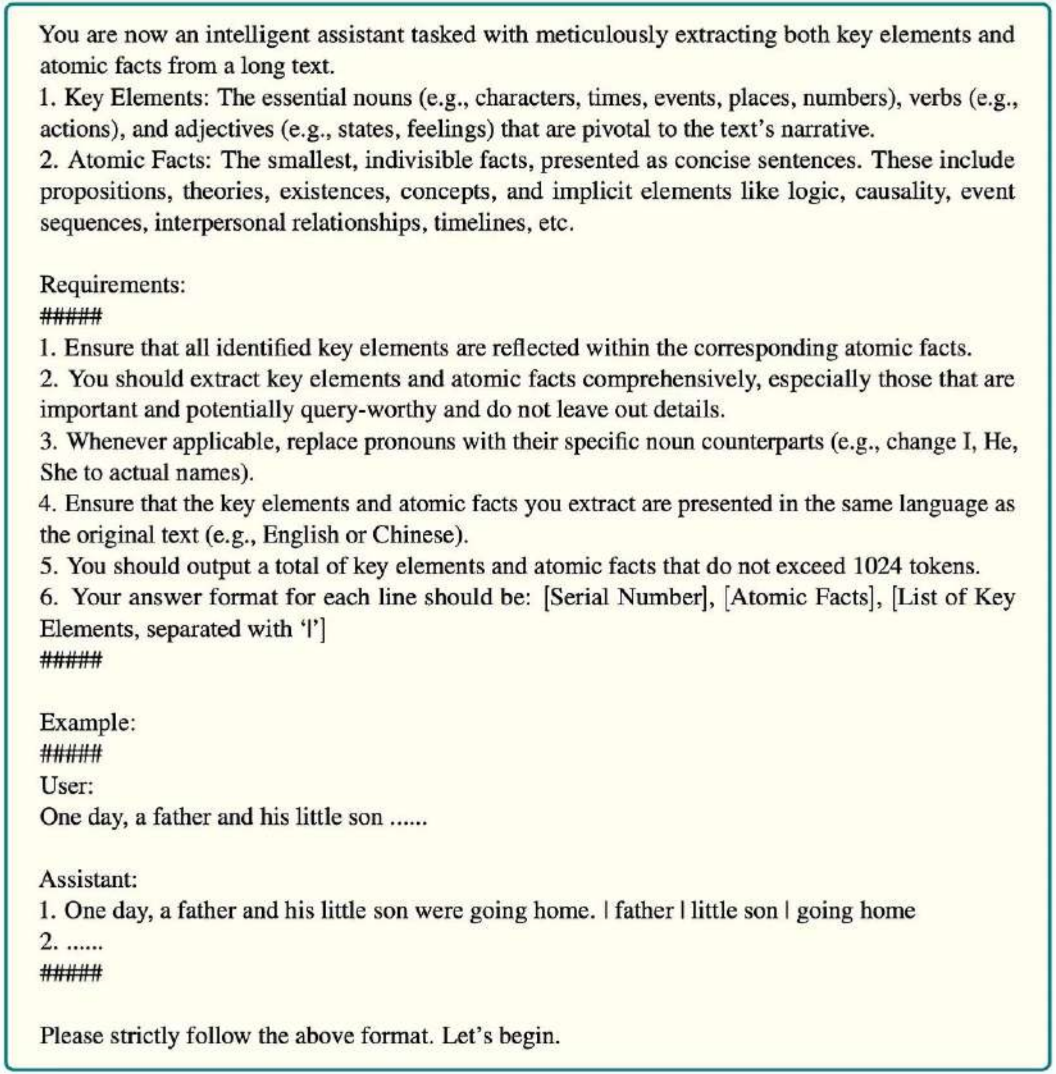
multi-hop traversal connecting multiple facts across documents



1. "Never Too Loud" is the fourth studio album by Canadian hard rock band Danko Jones.

2. Danko Jones is a Canadian hard rock trio from Toronto.

EVERYONE ASKING HOW I DID IT
ME: NO IDEA BECAUSE I FULLY GAVE INTO THE VIBES



You are now an intelligent assistant tasked with meticulously extracting both key elements and atomic facts from a long text.

1. Key Elements: The essential nouns (e.g., characters, times, events, places, numbers), verbs (e.g., actions), and adjectives (e.g., states, feelings) that are pivotal to the text's narrative.

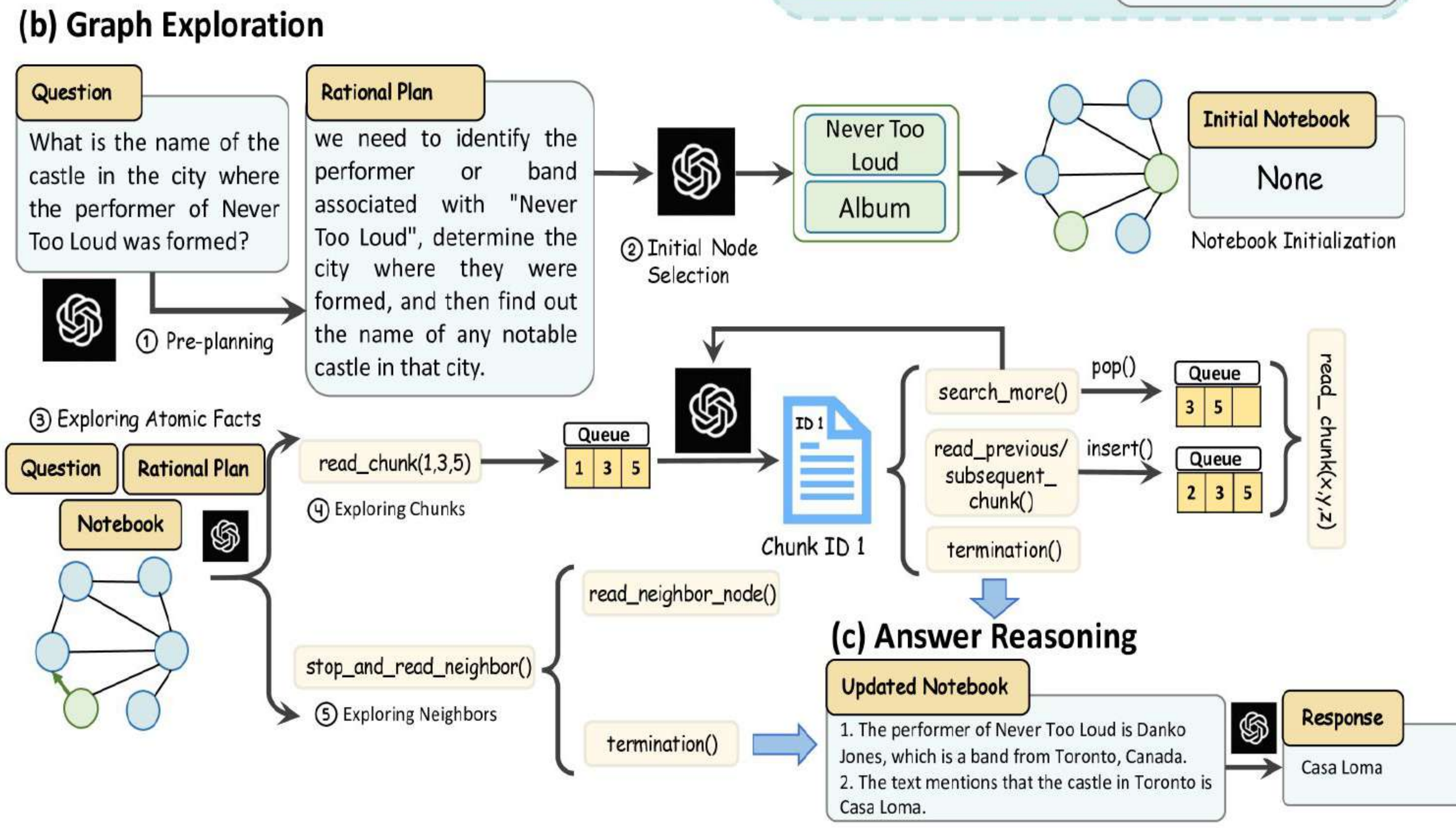
2. Atomic Facts: The smallest, indivisible facts, presented as concise sentences. These include propositions, theories, existences, concepts, and implicit elements like logic, causality, event sequences, interpersonal relationships, timelines, etc.

Requirements:
#####
1. Ensure that all identified key elements are reflected within the corresponding atomic facts.
2. You should extract key elements and atomic facts comprehensively, especially those that are important and potentially query-worthy and do not leave out details.
3. Whenever applicable, replace pronouns with their specific noun counterparts (e.g., change I, He, She to actual names).
4. Ensure that the key elements and atomic facts you extract are presented in the same language as the original text (e.g., English or Chinese).
5. You should output a total of key elements and atomic facts that do not exceed 1024 tokens.
6. Your answer format for each line should be: [Serial Number], [Atomic Facts], [List of Key Elements, separated with '|']
#####

Example:
#####
User:
One day, a father and his little son ......

Assistant:
1. One day, a father and his little son were going home. | father | little son | going home
2. ......
#####

Please strictly follow the above format. Let's begin.

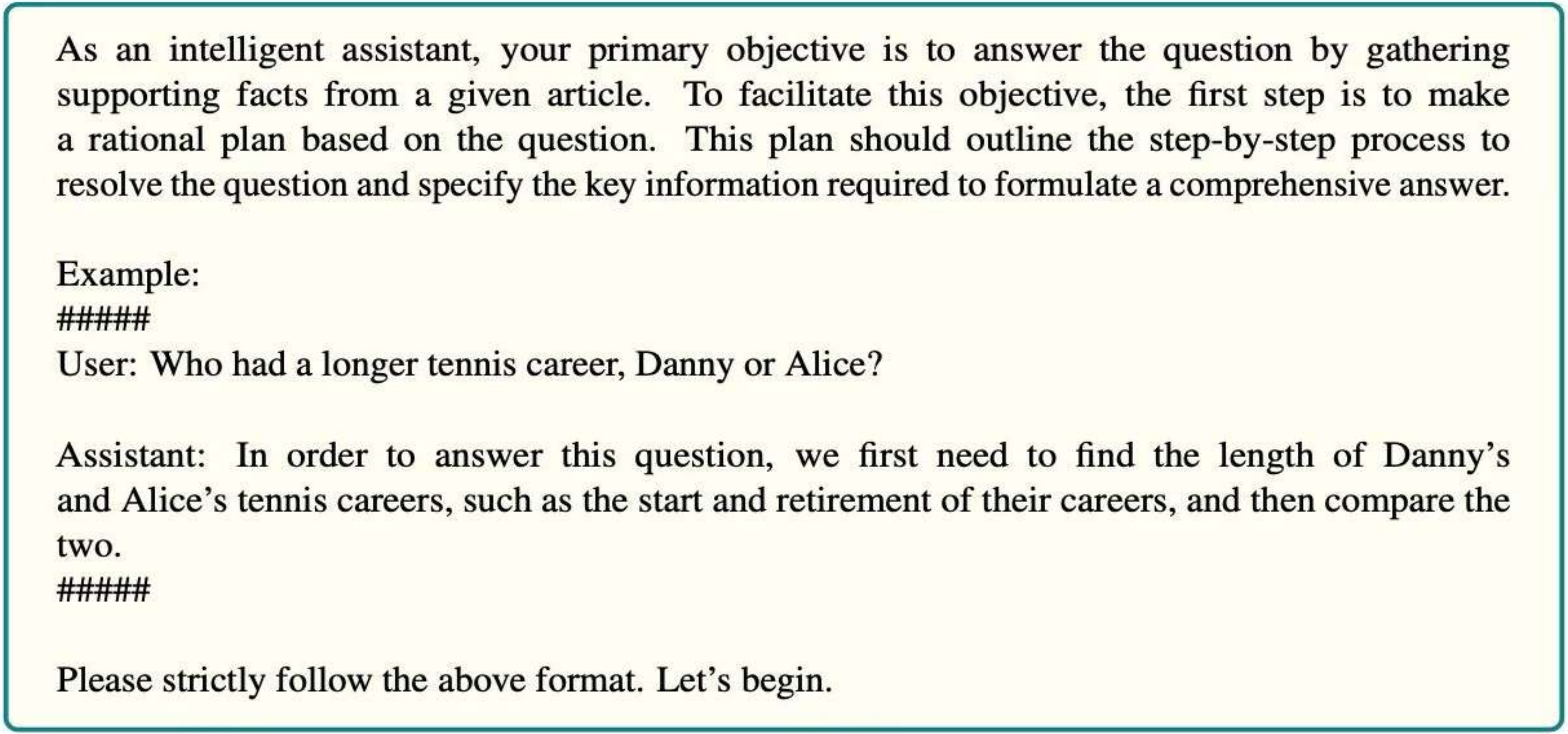


What is the name of the castle in the city where the performer of Never Too Loud was formed?

we need to identify the performer or band associated with "Never Too Loud", determine the city where they were formed, and then find out the name of any notable castle in that city.

1. The performer of Never Too Loud is Danko Jones, which is a band from Toronto, Canada.
2. The text mentions that the castle in Toronto is Casa Loma.

Casa Loma

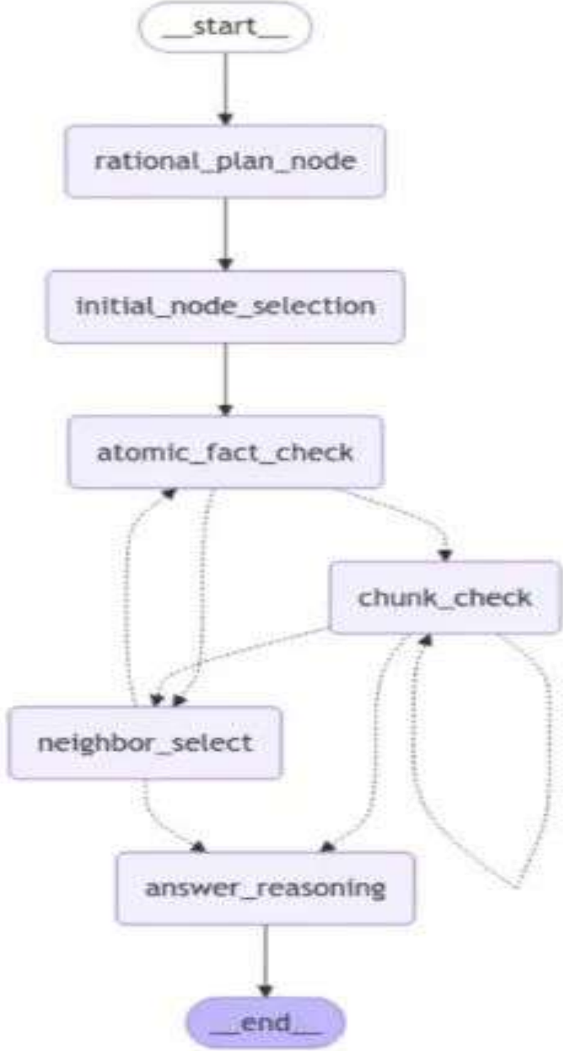


As an intelligent assistant, your primary objective is to answer the question by gathering supporting facts from a given article. To facilitate this objective, the first step is to make a rational plan based on the question. This plan should outline the step-by-step process to resolve the question and specify the key information required to formulate a comprehensive answer.

Example:
#####
User: Who had a longer tennis career, Danny or Alice?

Assistant: In order to answer this question, we first need to find the length of Danny's and Alice's tennis careers, such as the start and retirement of their careers, and then compare the two.
#####

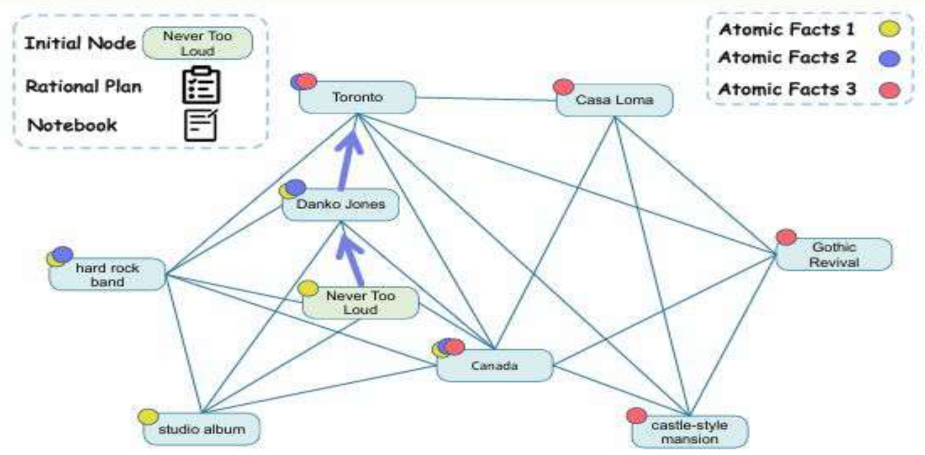
Please strictly follow the above format. Let's begin.



Exploration: Function Call Process

Exploring Atomic Facts Node: Never Too Loud; [Atomic Fact 1 from Chunk ID-6]
Call Function *read_chunk(ID-6).*
Exploring Chunks Realized the performer of Never Too Loud is Danko Jones.
Call Function: *search_more*
Exploring Neighbors Node: Never Too Loud; Neighbor Nodes: [hard rock band, Danko Jones, studio album, Canada]
Call Function *read_neighbor_node(Danko Jones)*
Exploring Atomic Facts Node: Danko Jones; [Atomic Fact 1 from Chunk ID-6, Atomic Fact 2 from Chunk ID-9].
Call Function *read_chunk(ID-9).*
Exploring Chunks Realized Danko Jones band is a band from Toronto, Canada.
Call Function: *search_more*
Exploring Neighbors Node: Danko Jones; Neighbor Nodes: [hard rock band, Never Too Lou, studio album, Canada, Toronto]
Call Function *read_neighbor_node(Toronto)*
Exploring Atomic Facts Node: Toronto; [Atomic Fact 2 from Chunk ID-9, Atomic Fact 3 from Chunk ID-13].
Call Function *read_chunk(ID-13).*
Exploring Chunks Realized the castle mentioned in the text in Toronto is Casa Loma.
Call Function: *termination*

Exploration: Visual Representation



Its Demo Time Finally!!!!!!!

Neo4j + LangChain + LangGraph = 🔥

Our Implementation - Open_GraphReader -> Open source implementation using Neo4j community edition Langchain and models Deepseek-R1-distill-llama-70b for ingestion & Llama-70b for retrieval. **

** Disclaimer - However we're GPU poor so we leverage open source LLMs through Groq

Application Areas

Medical Literature Review

clinical research or drug discovery

researchers often need to synthesize findings from hundreds of pages of studies. GraphReader can build a knowledge graph from these papers and answer multi-hop questions like "What are the long-term effects of Drug X on patients with Condition Y?" — without missing buried details.

Legal Document Analysis

Legal professionals deal with contracts

case law, and regulations that span thousands of tokens. GraphReader's coarse-to-fine reasoning and multi-hop traversal make it ideal for answering questions like "Which clauses in these contracts conflict with Regulation Z?"

Academic Research Assistants

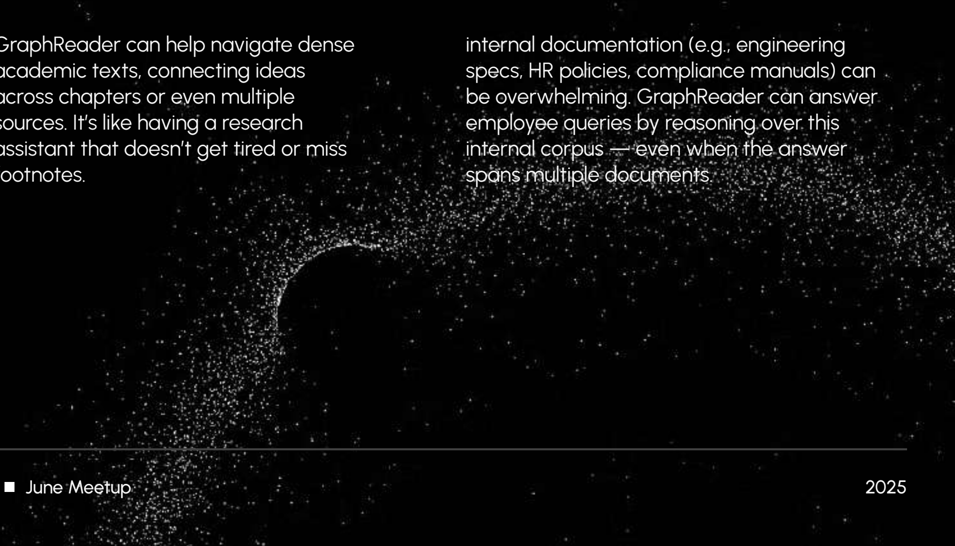
For students or scholars

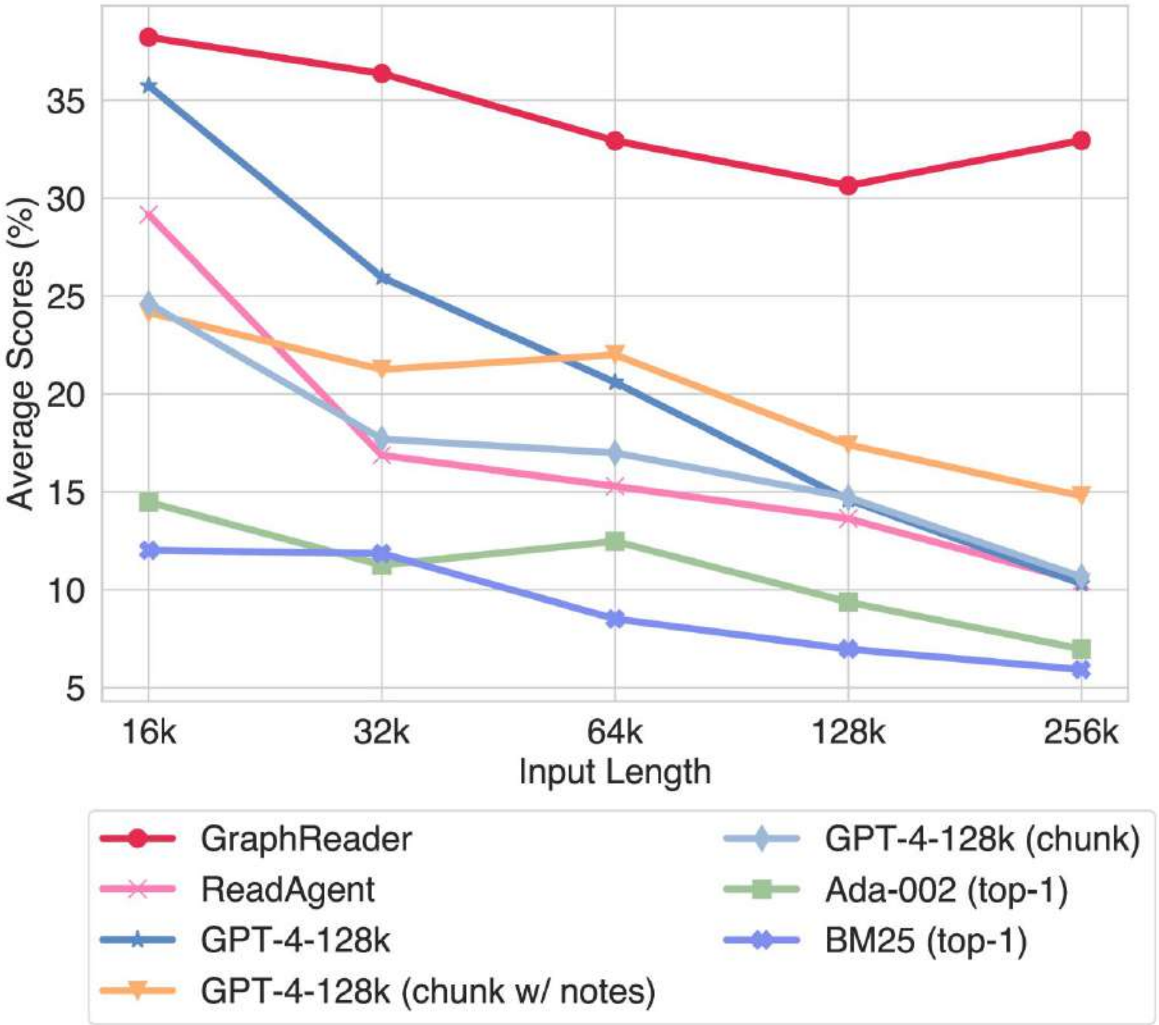
GraphReader can help navigate dense academic texts, connecting ideas across chapters or even multiple sources. It's like having a research assistant that doesn't get tired or miss footnotes.

Enterprise Knowledge Management

large organizations

internal documentation (e.g., engineering specs, HR policies, compliance manuals) can be overwhelming. GraphReader can answer employee queries by reasoning over this internal corpus — even when the answer spans multiple documents.





Performance on LV-Eval at 5 context length levels. GraphReader outperforms existing open-sourced and closed-source models while demonstrating a scalable performance in very long contexts.

In contrast, other models exhibit a significant decrease in performance as context length increases.

Limitations

- If we parsed many documents, broad keywords would end up having a lot of connections, which might lead to some downstream problems that aren't dealt with in the original implementation.
- Another minor problem is the non-determinism of the extraction, as the results will be slight different on every run.
- The efficiency of the agent depends on its planning and reasoning capabilities.

Resources

- https://arxiv.org/html/2406.14550v1
- https://medium.com/data-science/implementing-graphreader-with-neo4j-and-langgraph-e4c73826a8b7
- Code Repo -> https://github.com/dharmsurag/open_graphreader

Let's vibe code some AI Agents

All you need is....

- Detect hallucinations
- Deal with prompt injection
- Define prompting strategy
- Log everything for debugging
- Just hook up agents with MCP
- Choose which LLM for max accuracy
- Handle when tools fail or return garbage
- Ensure that the service is available 99.9%
- Latency less than 3-second response time
- Inject relevant context to avoid token limits
- Create safety filters so it doesn't go rogue
- Design fallback behavior when LLMs don't know
- Add dynamic memory for short-term and long-term
- Handle API failures and race conditions in async calls
- Monitor and rate-limit API so it doesn't burn $100 per minute

bitly

Thank you

Open to Questions