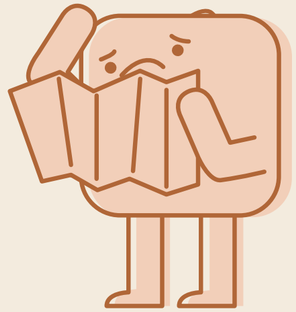
Problem Statement

As we graduate and prepare to move to new cities for jobs or career opportunities, we face significant challenges. Moving to an unfamiliar area means:

- Not knowing much about the neighborhood or its safety.
- Struggling to find housing that fits our budget and preferences.
- Being unaware of essential amenities like restaurants, and public services nearby.

This lack of accessible, centralized information makes relocation stressful and time-consuming, especially when balancing the demands of a new job or settling into a new environment.

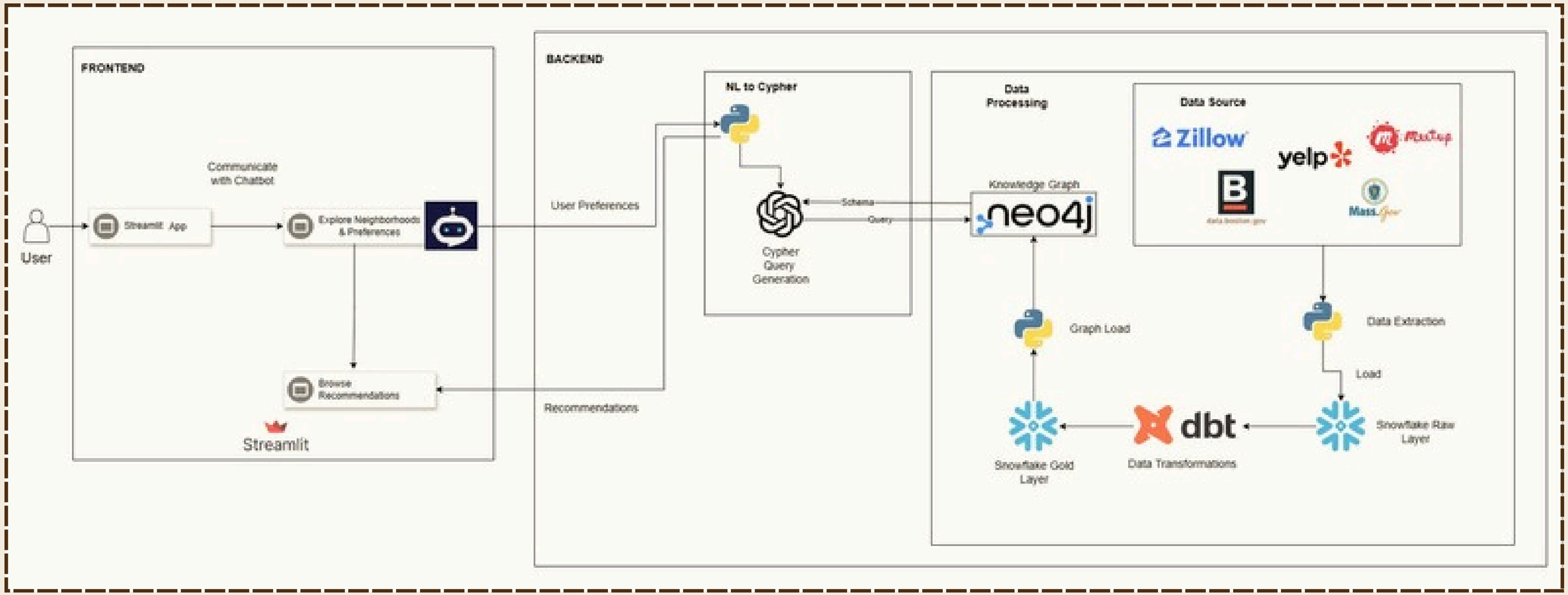


What is Saarthi?

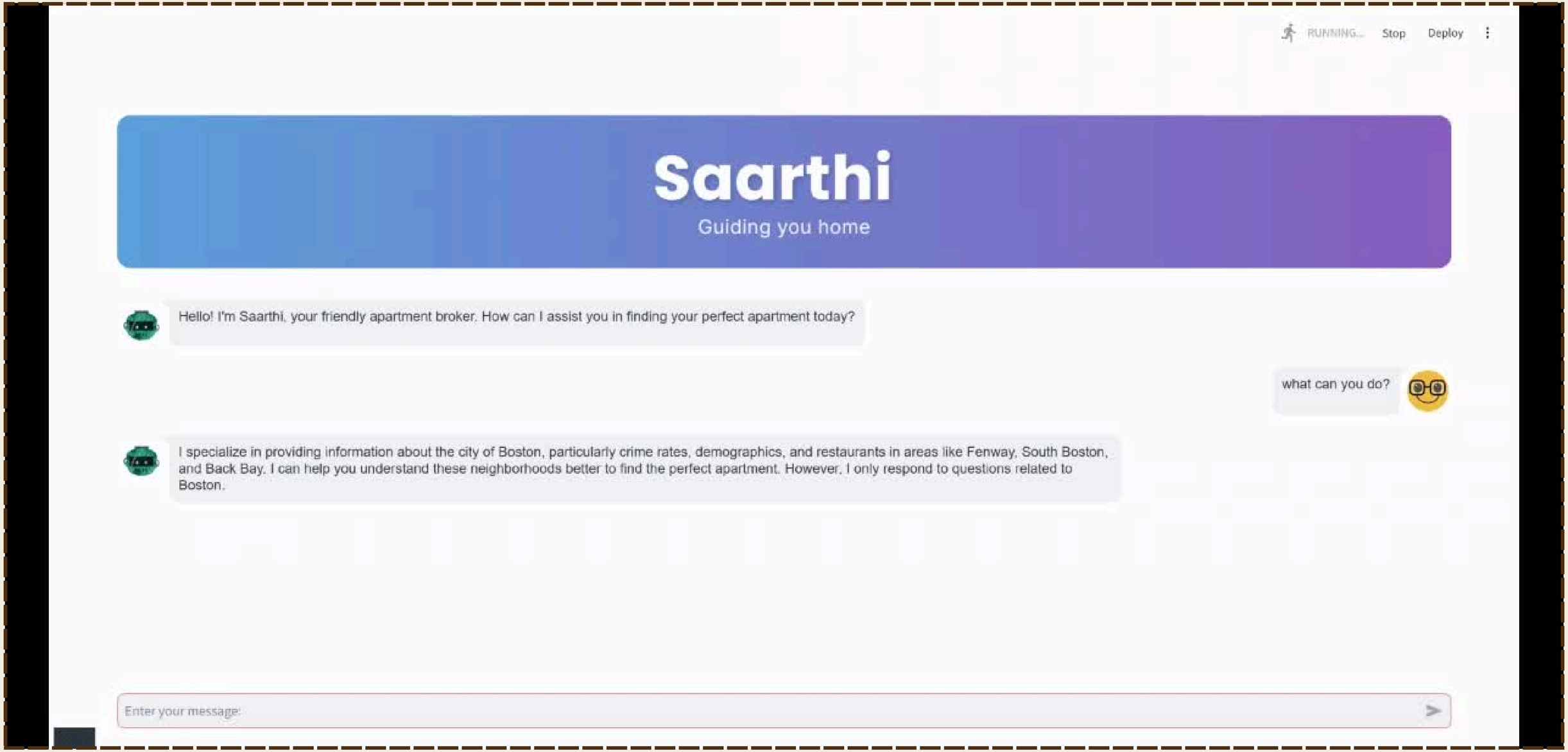
- Leverages LLM-driven RAG to guide users in making well-informed decisions for renting apartments in new cities (e.g., Boston).
- Enables users to explore neighborhoods by asking about crime rates, demographics, dining options, parks, and more—all in a conversational interface.
- Goes beyond RAG by utilizing a Neo4j-powered knowledge graph, offering dynamic and highly connected data insights for a deeper understanding of the area.

The word Saarthi is a Sanskrit/Hindi word that means "charioteer". It can also refer to someone who guides or leads someone on a journey.

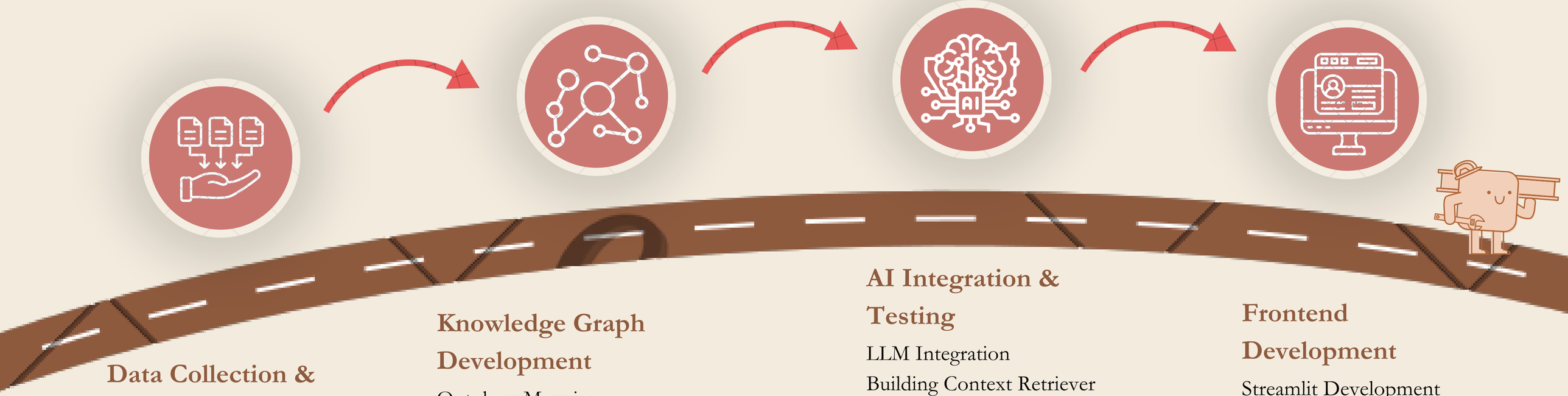
Project Architecture



Quick demo of our product



Implementation Plan



Data Collection & Preparation

Extraction Scripts
ELT Process
Data Quality Testing
Data Orchestration

Knowledge Graph Development

Ontology Mapping
Entity-Relationship Linking
Change Data Capture Mechanism
Knowledge Graph Loading

AI Integration & Testing

LLM Integration
Building Context Retriever
Prompt Engineering
NL-to-Cypher Component
Testing

Frontend Development

Streamlit Development
Adding Guardrails
Feedback Mechanism
Analytics Dashboard

Sources*

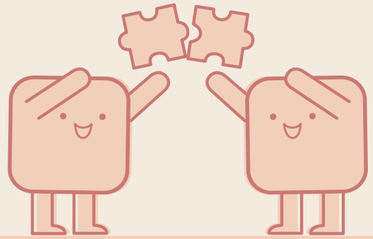
yelp

meetup

Zillow

Mass.Gov

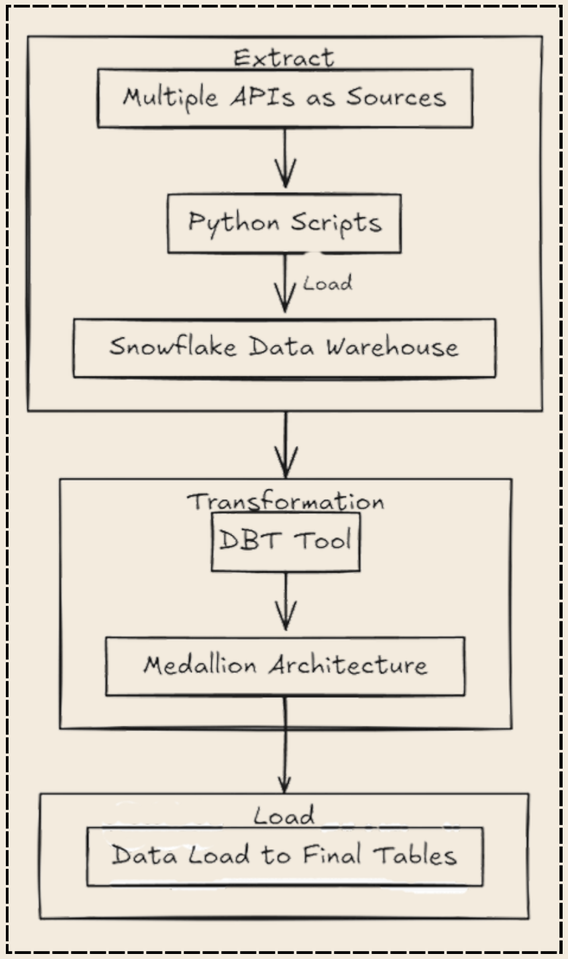
data.boston.gov

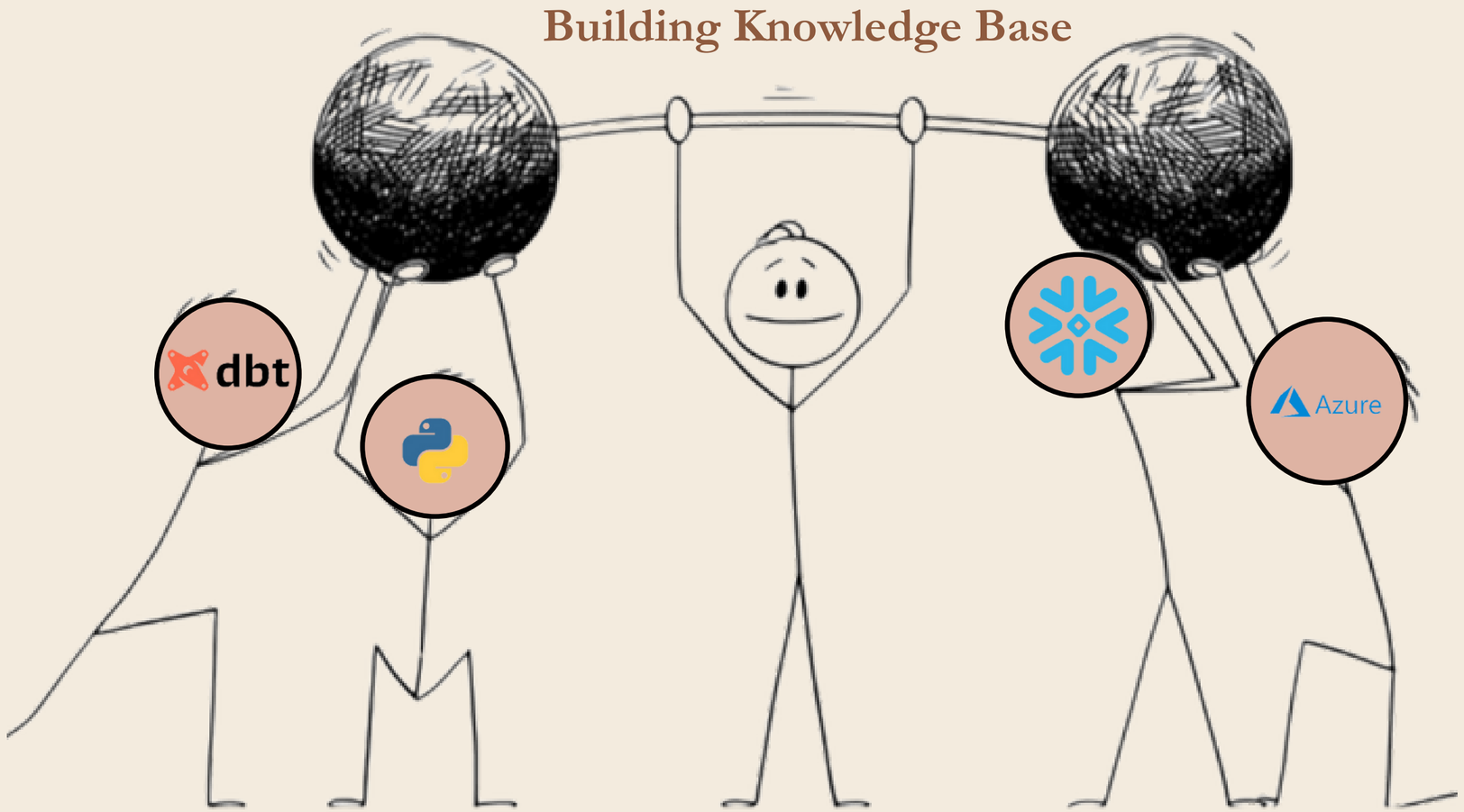


*Disclaimer and Acknowledgments

We would like to extend our gratitude to the teams at Zillow, Meetup, Yelp, and other data providers for their publicly available APIs, which made this project possible. The data presented in this project has been extracted from their respective APIs and is used solely for educational purposes.

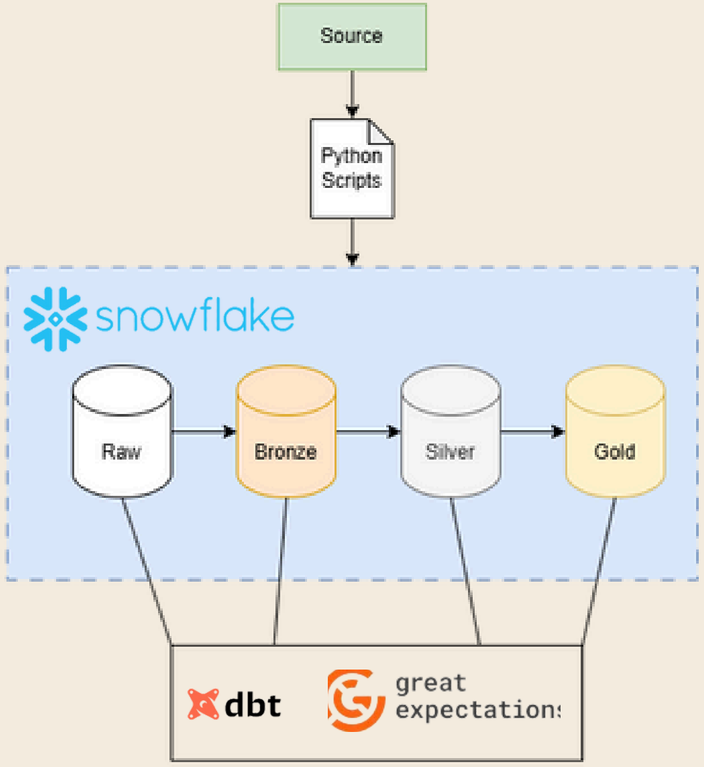
Data Collection and Preparation





Approach: ELT (Extraction - Loading - Transformation)
Data Warehouse: Snowflake

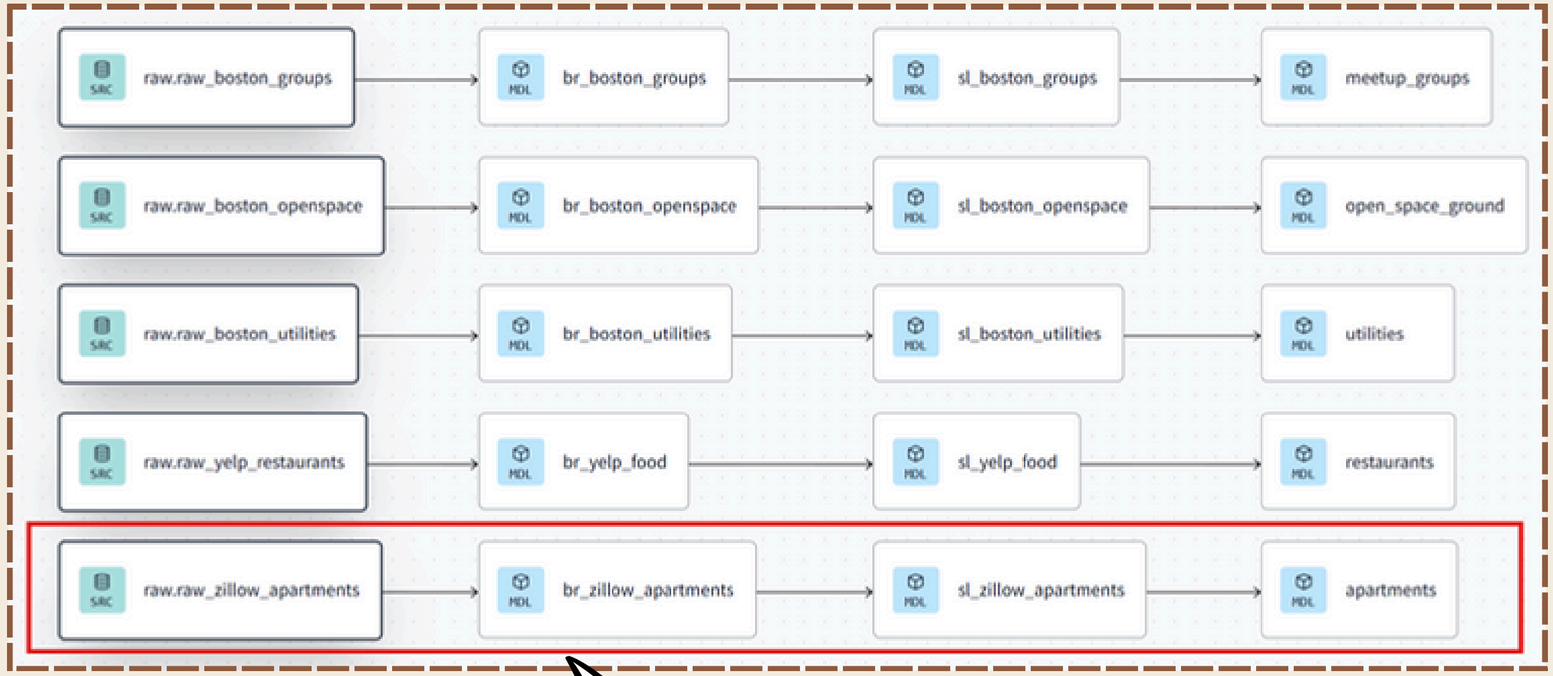
Data pipelines



Great Expectations and dbt-cloud work together within the Snowflake ecosystem, creating a robust framework for

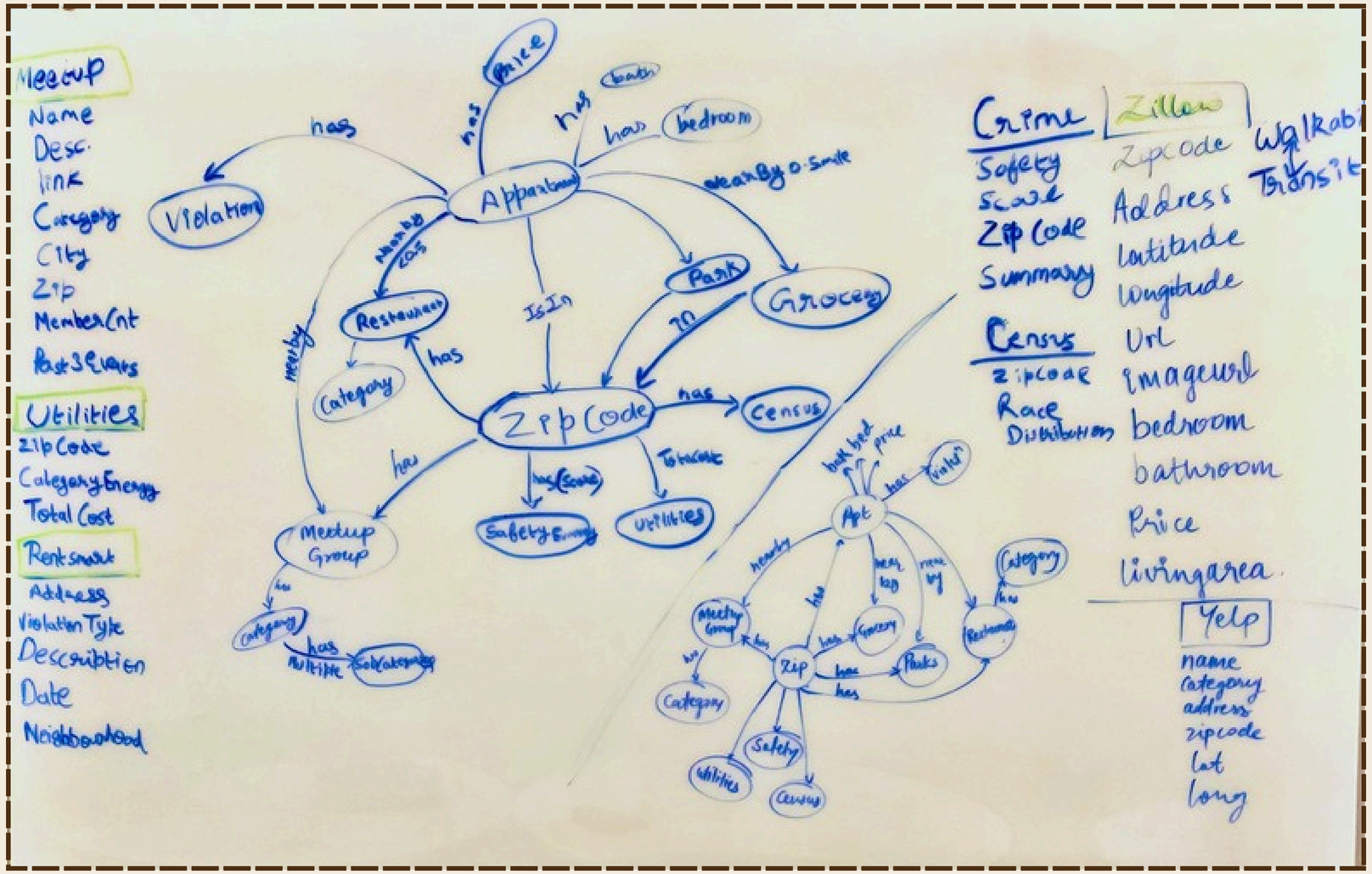
- transforming
- testing
- delivering high-quality data

DBT Lineage Graph



- Incremental DBT model (apartments)
- Model updates apartment listings and other information like rent changes every 24 hours, orchestrated seamlessly through dbt-cloud

Knowledge Graph Development



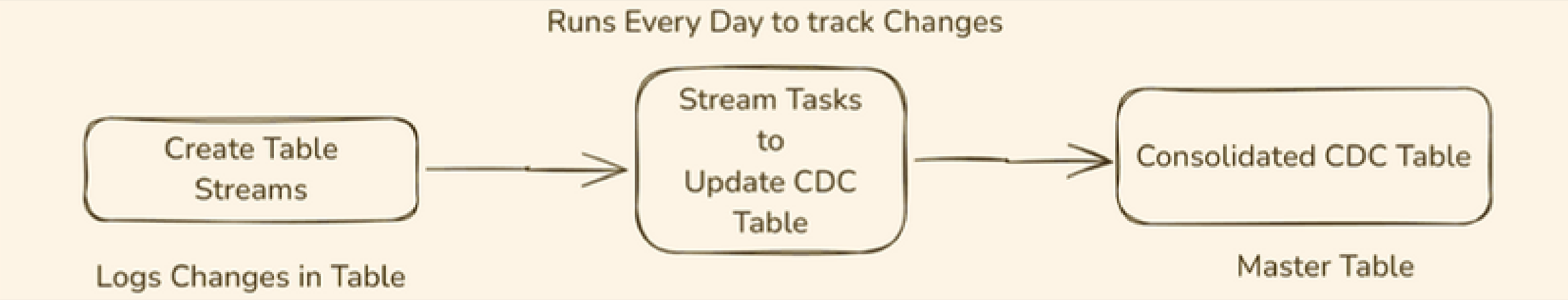
Entity Linking:

1. **Apartments -> nearbySubway**
2. **Apartments -> nearbyRestaurants**
3. **Neighborhood -> hasCrime**
4. **Neighborhood -> hasOpenSpace**
5. **Neighborhood -> hasSubway**
6. **Apartments -> isInNeighborhood**

and more...

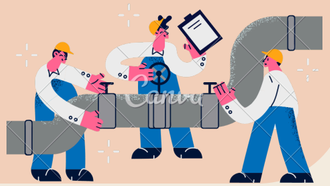
Snowflake CDC

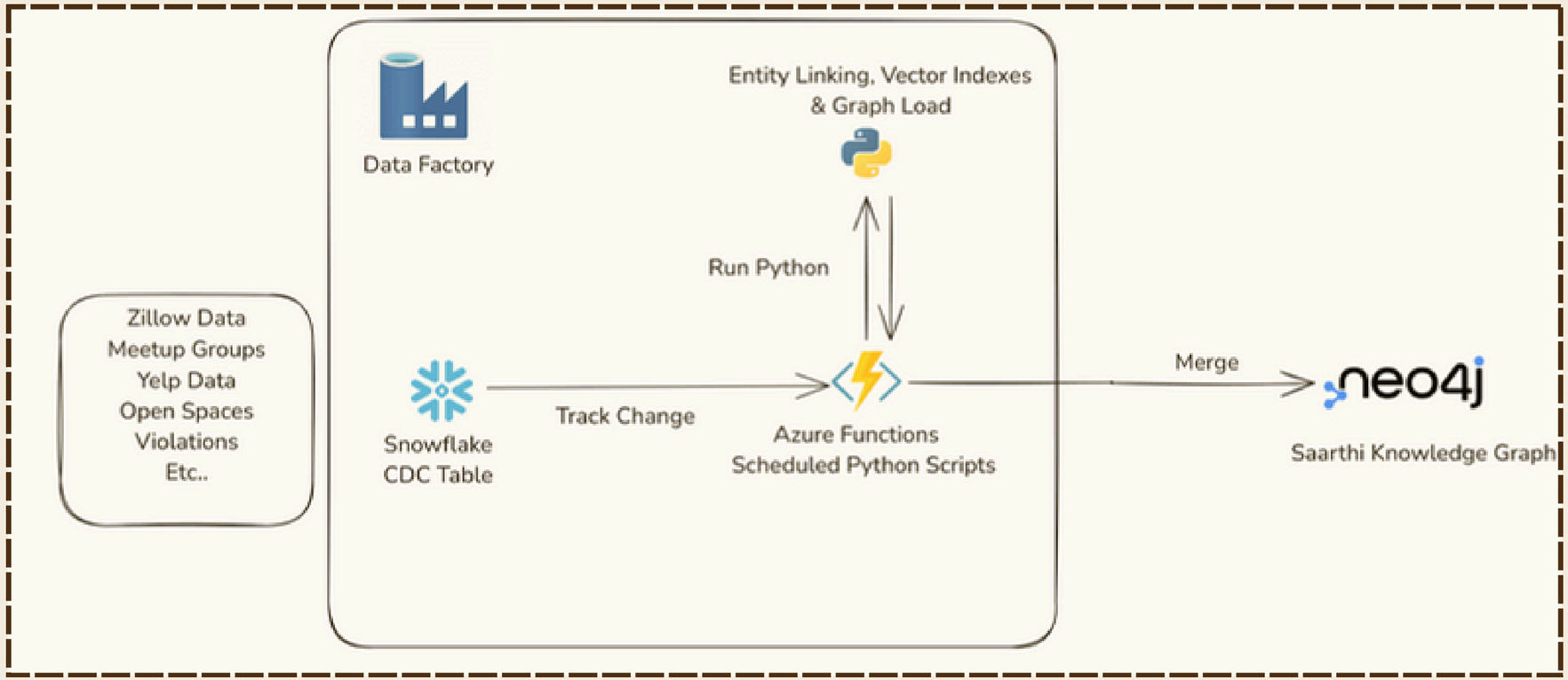
Since Snowflake doesn't provide a built-in method to capture data changes directly, we used Snowflake streams to monitor and track these changes.

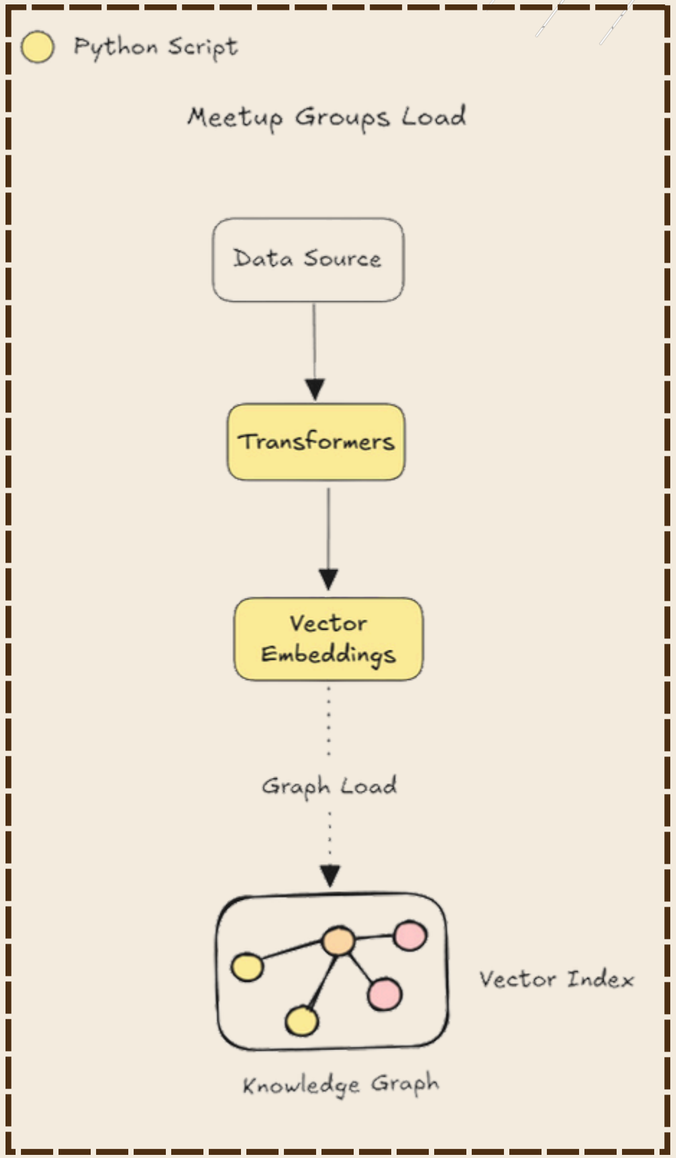


SOURCE_TABLE	ACTION	ISUPDATE	CHANGE_TIME
MEETUP_GROUPS	INSERT	false	2024-11-10T13:00:01.2582
MEETUP_GROUPS	INSERT	false	2024-11-10T13:00:01.2582
MEETUP_GROUPS	INSERT	false	2024-11-10T13:00:01.2582
MEETUP_GROUPS	INSERT	false	2024-11-10T13:00:01.2582
MEETUP_GROUPS	INSERT	false	2024-11-10T13:00:01.2582

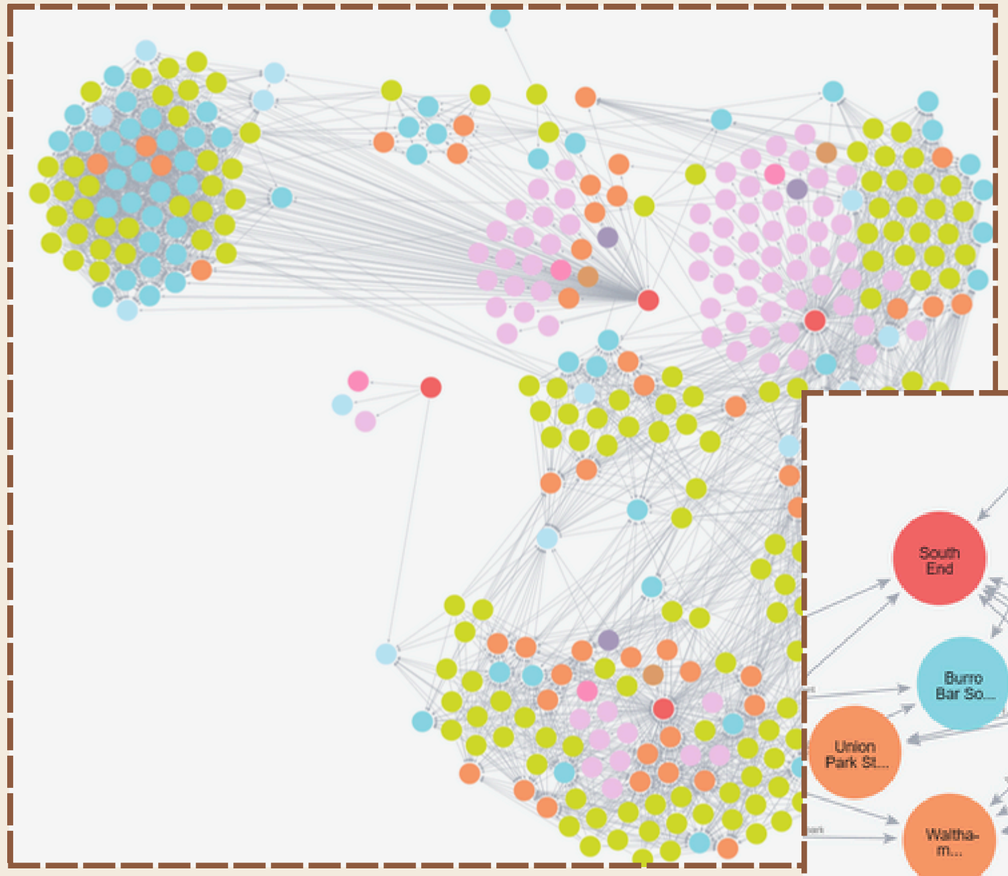
Knowledge Graph Load

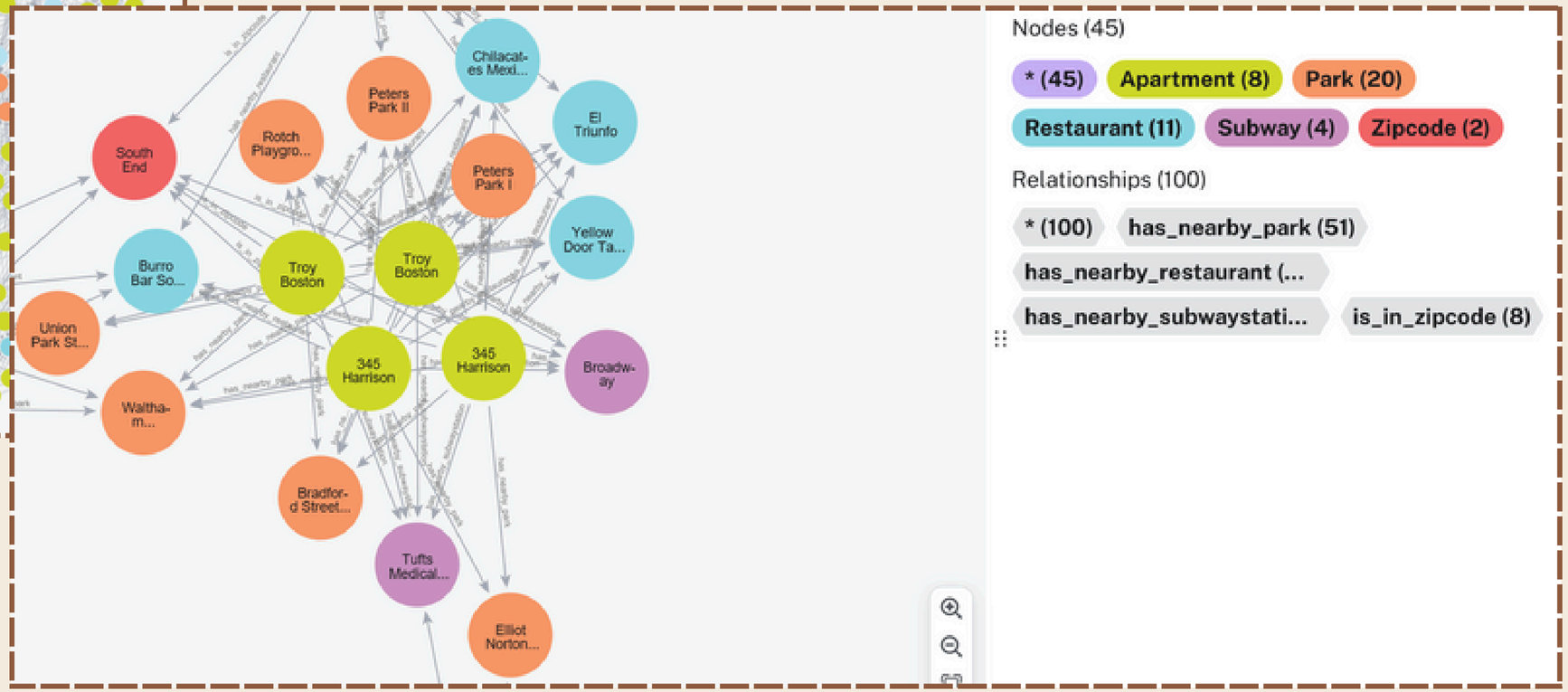






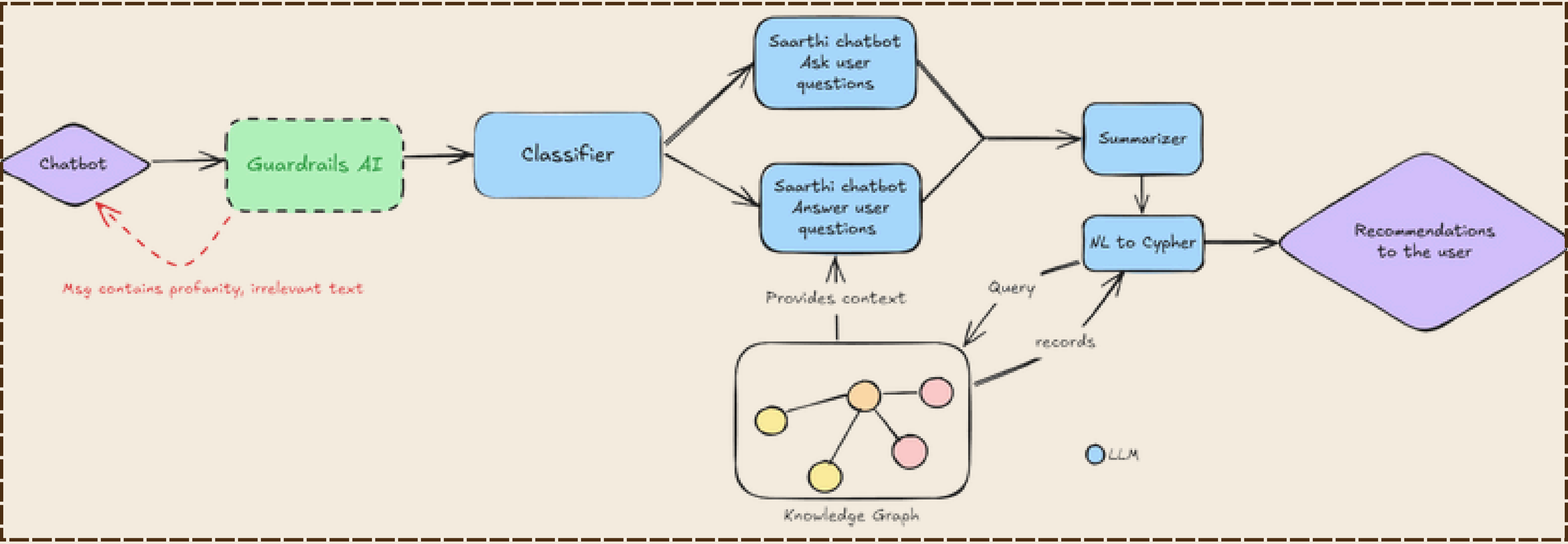
Knowledge Graph





LLM Integration

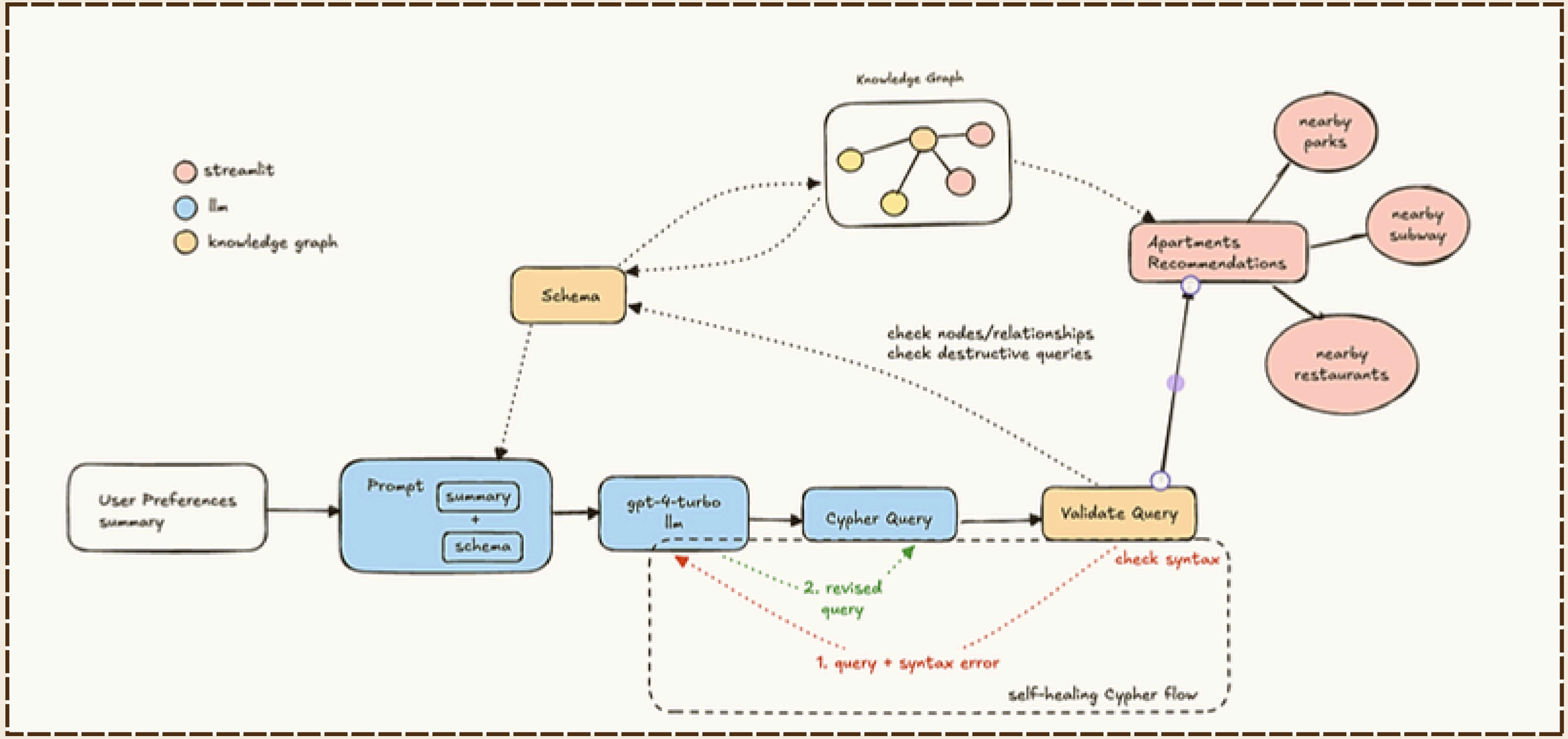
High-level flow of Chatbot



OpenAI LangChain Guardrails AI neo4j

Apartments Recommendation

Natural Language to Cypher Query

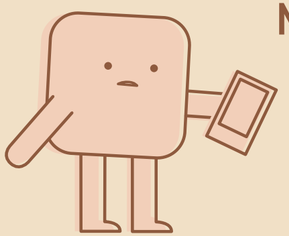


LLM Selection

"LLama 3.1"

Filters, and aggregations

Misunderstanding Schema



Hallucination

Multiple relationships

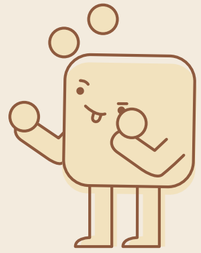
Syntax errors

```
--------------------------------------
 Cleaned query  MATCH (a:Apartment)-[:is_in_zipcode]-(z:Zipcode),
      (a)-[:has_nearby_restaurant]-(r:Restaurant {restaurant_cuisine: 'Indian'})
WHERE a.apt_bedroom_count = 2 AND a.apt_bathroom_count = 1
  AND a.apt_rent >= 2500 AND a.apt_rent <= 5500
  AND z.zipcode = '02215'
  AND toInteger(split(r.has_nearby_restaurant.distance, ' ')[0]) <= 15
RETURN DISTINCT a
ORDER BY a.apt_rent
LIMIT 4
Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.Unknown
RECOGNIZED} {title: The provided property key is not in the database} {description: One of the property nam
 in the database, make sure you didn't misspell it or that the label is available when you run this stateme
ng property name is: has_nearby_restaurant)} {position: line: 6, column: 25, offset: 288} for query: "MATCH
z:Zipcode),\n      (a)-[:has_nearby_restaurant]-(r:Restaurant {restaurant_cuisine: 'Indian'})\nWHERE a.apt_
oom_count = 1\n  AND a.apt_rent >= 2500 AND a.apt_rent <= 5500\n  AND z.zipcode = '02215'\n  AND toInteger(
tance, ' ')[0]) <= 15\nRETURN DISTINCT a\nORDER BY a.apt_rent\nLIMIT 4"
--------------------------------------
```

"GPT-4-Turbo"

Correct attribute reference

Understanding Schema



Handles complex queries

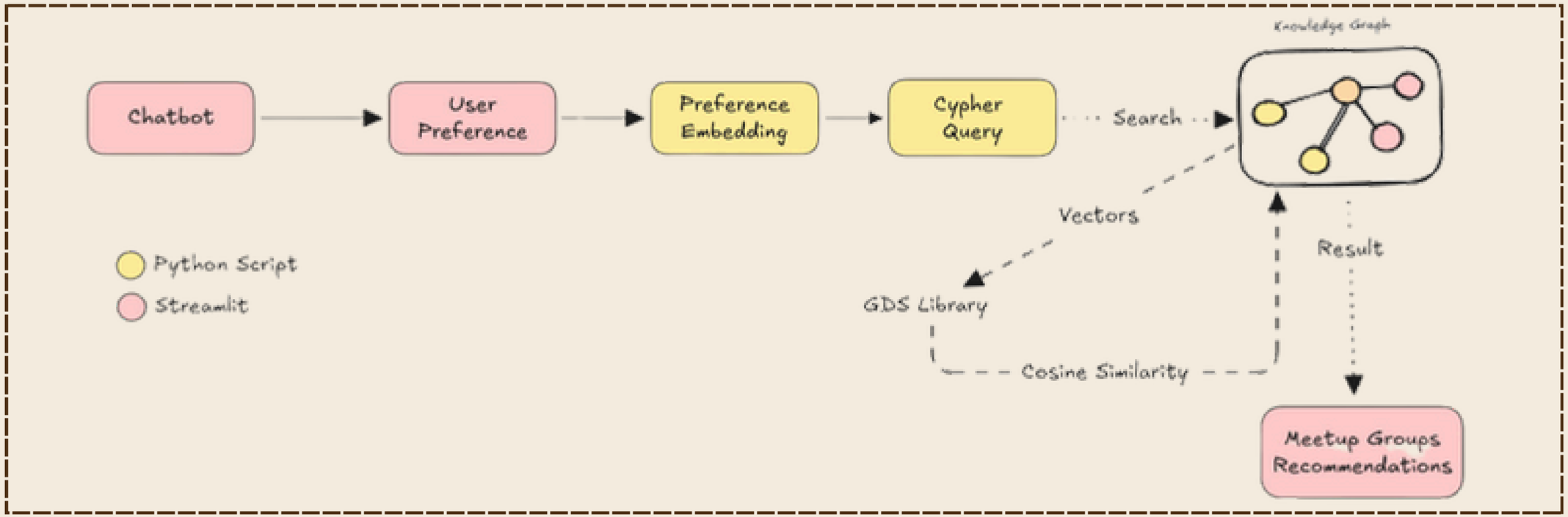
Consistent performance

```
--------------------------------------
 Cleaned query  MATCH (a:Apartment)-[:is_in_zipcode]->(z:Zipcode)
OPTIONAL MATCH (a)-[ns:has_nearby_subwaystation]->(s:Subway)

WHERE a.apt_address CONTAINS 'Back Bay'
  AND a.apt_zip_code = '02116'
  AND a.apt_bedroom_count IN [2, 3]
  AND a.apt_bathroom_count >= 2
  AND a.apt_rent <= 6500
  AND (ns.walking_time IS NULL OR toInteger(split(ns.walking_time, ' ')[0]) <= 15)

RETURN DISTINCT a
ORDER BY a.apt_rent
LIMIT 4
--------------------------------------
```

Meetup Groups Vector Search



Evaluating Approaches for Search

Explored Options

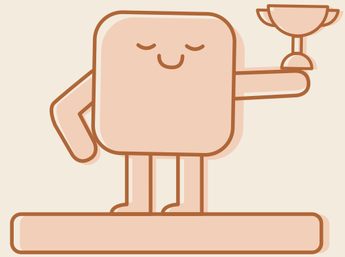
- Neo4j GDS Library
  - Best for batch computations
  - Excels in graph-based workflows
- Pinecone Integration
  - Real-time vector similarity
  - Scales to large datasets

Our Decision

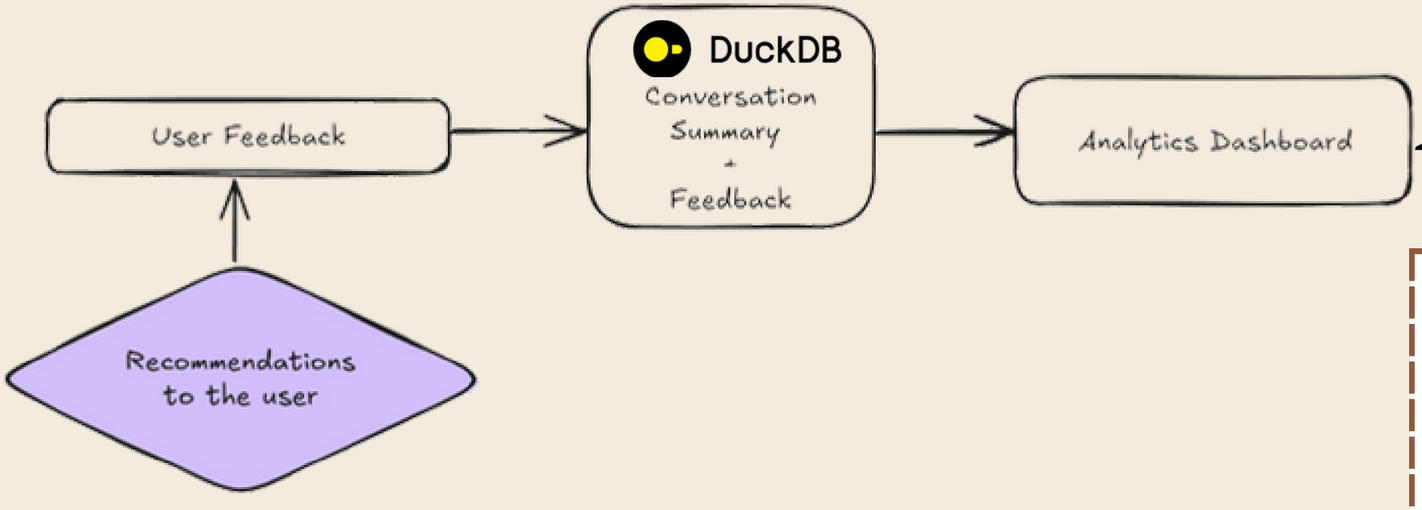
✅ Neo4j GDS

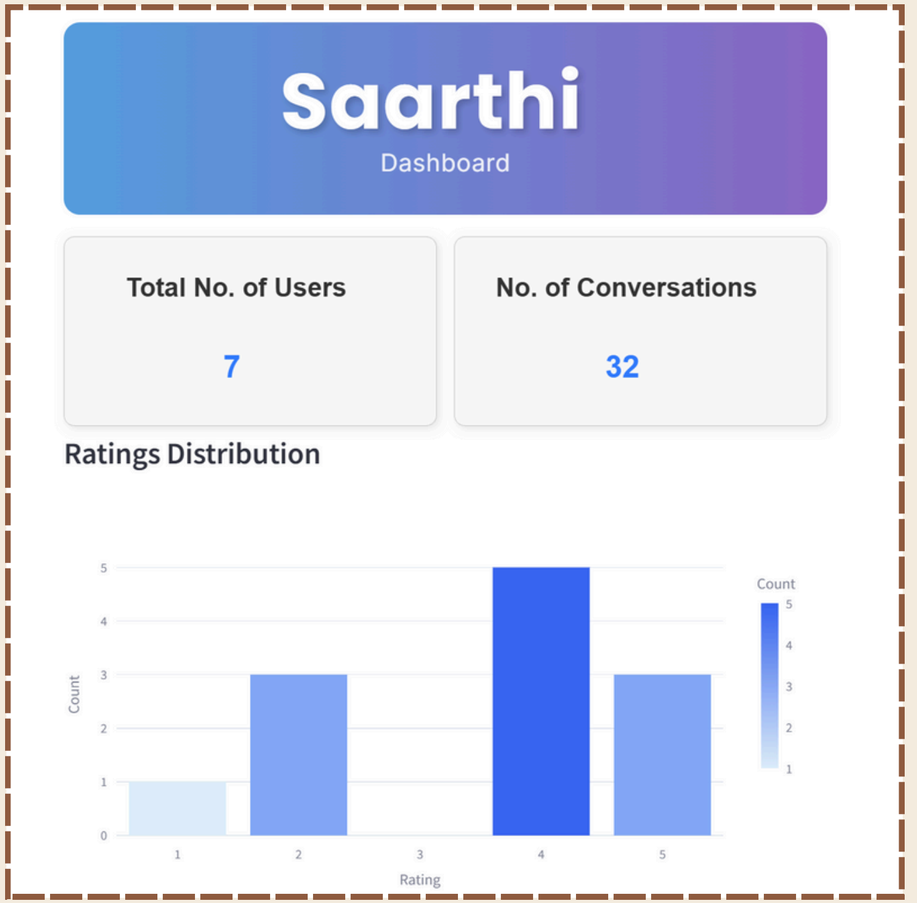
Why?

- Data size manageable
- No need for real-time analysis



Feedback and Analytics





Capturing user feedback enables us to evolve the chatbot by identifying gaps, refining responses, and enhancing user satisfaction over time.

Implement a feedback-driven improvement system where negative user feedback triggers a new AI agent. This agent analyzes the feedback, generates an optimized few-shot query for the user's question, and stores the improved query in the knowledge base for future use.

Key Choices

- Data Transformation- DBT vs Python vs PySpark

Tool	Pros	Cons	Why Not Chosen
DBT-cloud	Modular, declarative SQL-based, easy orchestration and testing	Requires structured SQL models	Chosen - Perfect fit for our data needs
Python	Highly flexible, extensive libraries	None as such	Need to write extra code for testing, configuring connectors
PySpark	Distributed processing, scalable for big data	Requires heavy setup and maintenance	Overkill for our data size and complexity

- Coming up with a tool to handle conversational flow- LangGraph vs DataStax vs LangChain vs CrewAI
- Researching different methods for Vector Storage and Search (Graph Data Science & Pinecone Integration)
- Researched various methods for hosting Python applications on Azure and decided to use Azure Functions due to its lower cost than other options, despite its limitations in logging capabilities.
- We used all-MiniLM for Meetup vectorization due to its speed, and quality embeddings, ideal for processing diverse descriptions quickly and at scale.

Thank You