Git link - GIT

Phase 1

Extraction Approach:

First i will focus on extraction of 3 types of information:

- Entities - People, Objects, Places..
- Events - “created playalist”, “purchased iphone”...
- Relationships - connection between the user, entities and events. User can OWN an entity, User can CREATE/ATTEND an entity, can PREFER entity, RELATED_TO other entities..

I will structure it as a json file, probably divide it to 3 parts: entities, relationships and events. Each part will have some fields/attributes (name, type, time/date…)

I will use the LLM to extract the information because:

- It has context and understands relationships better than rule based or patterns.
- It may be better than the naive approaches in varied text and edge cases that I didn't come up with when I designed the extraction method.
- Can add to it some tools to create an agent to infer the time better if it won’t work out good.

This method seems the simplest to implement and will probably identify a lot of details. It may take some time to run the extraction but I believe it will have the best accuracy with the fastest implementation time.

The challenges will be :

- Figure out how to work with neo4j (LLM will help me with it quite good)
- Extracting the multi-session data (will probably need to show the LLM the history or use a second LLM to combine multiple graphs together.

Database Design

I chose to work with neo4j. I believe it fits directly to this type of mission (relationships and entities). Although I’m not familiar with neo4j from before, I believe AI tools will help me learn and implement fast.

First draft of scheme for storing extracted memories:
Nodes:

- User
- Entities
- Event

Relationship types:

- User - Entity : OWN, CREATE, HAS, PREFER, HAS_GOAL, HAS_PROBLEM
- User - Event : ATTEND, MENTION, IMAGINE
- Entity - Entity : REFERS_TO, RELATED_TO
- Event - Event : BEFORE, AFTER

Properties/Attributes: extracted_from, session_date, name, type, description..
The json format and this graph scheme are very similar to each other. I split the json to entities,events and relationships and from there I can build the neo4j graph with more ease.

Current Challenges and Solutions

The extraction will be difficult in this areas:

- pronouns (he, it) reference to previous entities (I believe LLM can deal with this if its in the same conversation)
- Temporal reasoning - “7 days ago” needs to be calculated from the session date (if the LLM won’t deal with it, I can make an agent with tools or use some simple packages to calculate it if I have the time)
- Implicit information - “going to the vet” means the user has a pet. (currently not dealing with it with my time frame.)
- If a new fact is a contradiction to a former memory - the old one should be removed (not dealing with it in my time frame). Can be dealt with another call to a LLM, can only look at the newest memory.

Tradeoffs

I chose to do simple fact extraction and currently not deal with implicit information. Extracting only what’s directly stated. If I have the time I will think of a way to deal with the implicit information.
Temporal reasoning will be dealt with by the LLM first, if time will be enough and results will be poor I will try other tools.
Extraction will be done in a single call to LLM.

I prioritize a simpler working system over a highly sophisticated system that will not be achievable in my time frame.

Future Improvement

Originally I thought of doing preferences/goals in nodes and not as a relationship, I believe it will let me handle complicated preferences better and save more attributes on them if they were nodes.
If I had more time I could try to extract the information with multiple LLM calls and not just a single call. One LLM will extract the facts, another one will deal with implicit information, maybe another one will get the history and try to connect linked memories together. Also there could be a judge in the end that will have to check all their work and if it does not approve we do this cycle again.
Maybe another pipeline will be: extract entities, then relationships, then implicit information.

Limitation of current approach : deals with simple co reference in a conversation, I believe that multiple conversation reference won’t be achievable in my time frame.
Extraction of simple facts, not implicit relationships (“going to the vet”)
Simple temporal extraction and calculation.

Scale to Millions of Conversations:
Use smaller or faster models, maybe reduce the calls to them (if chosen the multi-step extraction). Try to add more tool calls to make the pipeline run faster instead of relying solely on LLM calls.

Development

I wanted to run the project in the university cluster (to use their GPUs for the llama3 model), so first I tried to install neo4j on the cluster. I tried for 45 minutes and didn't succeed.
I decided to split this project into 2 environments:
In the university cluster I ran the extraction with the LLM, producing json outputs.
The outputs were an input to the colab file to save it in neo4j (to save the time of the running llama model in colab). I tested 2 methods of extractions: passing all the input_text together to the model or splitting it. The second method produced superior results with more details and facts correctly identified. It's still not perfect but I believe that with more time I could have perfected the prompt and get better accuracy.

Initially I had a prompt that worked fine, but a lot of the relationships were not extracted well enough (for example: I ate yesterday was extracted as User →PERFORMS → eating yesterday). The file "llm_responses_split.json" is the result with this prompt. I added another file with the new prompt that should fix this problem, it's called "llm_responses_split_improved.json". This file is smaller because I ran out of time, so it only extracted data for 20 sessions. It still has a large number of "PERFORMS" but it's an improvement from before. I added both runs to the repository.

After the LLM extraction I uploaded the file to the colab, parsed the data and combined every session together. A good approach (if time was enough) was to look at every new data coming and decide whether I should add it to the memory or if it contradicts something from earlier. It can be done with an extra LLM checking all the contradictions and saving only the newest data.

Research Proposal

1. Extraction Challenge: Inconsistent Event & Relation Names

One problem I ran into is how the system turns sentences into events and relations.
Example:
User says: "I was listening to music yesterday."
The extraction sometimes gives:
User → performs → listening to music
but what we actually want is something like:
User → listens_to → Music

2. Why This Matters (Especially the “perform” Issue)

Using “performs” for almost everything causes major problems:

- Different events look identical.
- We lose important semantic information: the point of the memory is to remember what the user actually did. “User performs X” doesn’t tell you how they interacted with X.
- Retrieval becomes much weaker - if you ask: “What music does the user listen to?” You can’t reliably find it, because the relation is just “performs.”
- It doesn’t scale - the more conversations you have, the more useless “perform” relations pile up, making the graph harder to use.

In short:
The overuse of “performs” collapses many different human actions into one generic bucket, destroying the value of structured memory.

3. Current Approaches

Most existing memory systems (Mem0, LlamaIndex Memory) use:

- LLM prompts that describe a schema
- Templates for events and relations
- Some simple verb-mapping rules
- Post-processing to clean up relation names

What I tried:
I only had time to force a fixed schema in the prompt. I ran out of time to do more preprocessing and perfecting the prompt.

4. Proposed direction
My idea: Instead of trying to get extraction in one step, we can try two steps.
Step 1 : Let the LLM extract freely -It finds events, relations, entities without worrying about the schema.
Step 2 : Canonicalize (normalize) everything- A second LLM call (or classifier) takes the raw extraction and rewrites it into a fixed list of allowed event and relation types.

So:
User → performs → listening to music
becomes
User → listens_to → Music

Experiments:

- Compare one-step vs two-step extraction
- Measure how many unique relation names appear
- Check whether retrieval accuracy improves

Success metrics:

- Fewer “performs” in the data, better retrieval recall, fewer duplicate events in memory

5. How Retrieval Works With This Approach

If relations are normalized, retrieval becomes much easier:

- Queries map directly to known relation types (“What events did the user attend?” → look for ATTENDED_EVENT)
- Graph search becomes reliable.

Time log:

phase 1 (reading and planning) - 2 hours
phase 2 (setup) - 2 hours (had troubles with colab and cluster)
phase 3 (extraction implementation) - 2.5 hours
phase 4 (verification) - 45 minutes
phase 5 (documentation) - 45 minutes
phase 5 (deleting some code, arranging the files- 45 minutes