Robotic Memory with Efficient Robotic Task Planning using Large Language Models, Vector Databases, and Embeddings

Faisal Alshinaifi

July, 15 2023

Abstract

In the intricate domain of robotic task planning, traditional methodologies often find themselves overwhelmed, especially when navigating the vast expanse of potential robotic actions in dynamic environments. While large language models (LLMs) have emerged as a beacon of hope, capable of discerning potential actions and crafting sequences from natural language, they too have their limitations. Some demand an exhaustive breakdown of every possible move, while others produce narratives that occasionally veer into the realm of the unattainable.

Our research pivots on a transformative approach: the fusion of LLMs with vector databases and embeddings to solve zero shot tasks that gets better the more the robot is used. By equipping the system with a text-based "memory" reservoir, the vector database, empowered by embeddings, conducts rapid similarity searches. This not only pinpoints the most pertinent data for the LLM prompt but also drastically reduces computational overhead and accelerates the planning process.

Building on this foundation, our system employs programmatic blueprints enriched by executable sample programs to further refine its output by adding the output to the prompt.

1 Background

1.1 Vector Databases in the Era of Embeddings

1.1.1 Introduction

With the proliferation of embeddings across diverse domains, ranging from Natural Language Processing (NLP) to computer vision, there emerges an imperative need for efficient mechanisms to store, search, and retrieve these high-dimensional vectors. Traditional databases, tailored for scalar data, falter in this new paradigm, paving the way for the advent of vector databases.

1.1.2 Definition

A vector database, occasionally termed a vector search engine, stands as a specialized database system adept at managing high-dimensional vectors. Its core functionality revolves around operations such as the nearest neighbor search, pinpointing vectors in the database that bear the highest similarity to a provided input vector.

1.1.3 Core Principles

- **Distance Metrics:** At the heart of vector databases lie distance metrics, such as the Euclidean distance or cosine similarity, which gauge the similarity between vectors.

- **Indexing:** Diverging from traditional databases that index records based on scalar keys, vector databases architect multi-dimensional indexes. Techniques: (TO BE FILLED)

- **Approximate Nearest Neighbor (ANN) Search:** Given the computational heft of exact nearest neighbor searches in sprawling dimensions, vector databases often resort to approximate methodologies. These strategies sacrifice a modicum of accuracy in favor of substantially expedited search durations. (MORE EXPLANATION?)

1.1.4 Applications

- **Semantic Search:** In the realm of NLP, embeddings encapsulate semantic nuances. Vector databases facilitate document or sentence searches grounded in semantic similarity, transcending the confines of exact keyword matches.

- **Image Retrieval:** In computer vision tasks, images transmute into vectors, termed image embeddings. Leveraging a vector database, one can swiftly retrieve images bearing similarity from an expansive dataset.
- **Recommendation Systems:** (TO BE FILLED)

  (MAYBE ADD LLM AND MACHINE LEARNING)

2 Introduction

The world of robotics is in constant flux, continuously pushing boundaries and redefining possibilities. At the heart of this evolution is the challenge of task planning. As robots venture out from their controlled confines into the unpredictable tapestry of real-world scenarios, the imperative grows for them to adeptly decipher and act on human directives. While conventional methods of robotic task planning have their merits, they often stumble when confronted with the intricate subtleties and multifaceted nature of human language.

This is where Large Language Models (LLMs) come into play. Trained on a treasure trove of textual data, these sophisticated models have demonstrated an ability to grasp and produce text that mirrors human expression and reasoning. Their application in robotic task planning is exciting, offering a glimpse of a future where robots can fluidly translate human directives into actionable tasks. Yet, even these formidable models can be daunted by the expansive array of potential actions and ever-shifting environmental contexts.

Our paper delves into an innovative approach, intertwining the prowess of LLMs with the efficiency of vector databases and embeddings. This fusion aims to not just interpret human directives but to also swiftly access and process pertinent domain knowledge. In this exploration, we endeavor to craft a bridge, seamlessly connecting human directives to precise robotic actions, heralding a new era of harmonized human-robot collaboration.

3 Methodology

3.1 Data Acquisition and Preprocessing

The primary data source is a text file (`Memory.txt`), containing domain-specific knowledge or a set of commands suitable for the robot. Ranging from experiences and capabilities of the robot to be utilized to solve given tasks.

```
with open('Memory.txt', 'r') as f:
    text = f.read()
```

The data is tokenized using the GPT-2 tokenizer, converting the raw text into a sequence of tokens.

```
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
def count_tokens(text: str) -> int:
    return len(tokenizer.encode(text))
```

3.2 Text Chunking

The text is segmented into manageable chunks to optimize subsequent processes.

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 512,
    chunk_overlap  = 24,
    length_function = count_tokens,
)
chunks = text_splitter.create_documents([text])
```

3.3 Embedding Generation

Each text chunk is transformed into a dense vector representation using OpenAI's embedding model.

```
embeddings = OpenAIEmbeddings()
```

3.4 Vector Database Creation

The embeddings are stored in a FAISS vector database, facilitating efficient similarity searches.

```
db = FAISS.from_documents(chunks, embeddings)
```

3.5 Retrieval Mechanism

Upon receiving a user's movement request, the system conducts a similarity search in the FAISS database.

```
Uquery = input("request␣a␣movment:␣")
docs = db.similarity_search(Uquery)
```

3.6 LLM Integration

The retrieved chunks are processed by an OpenAI Large Language Model to generate a sequence of actions.

```
chain = load_qa_chain(OpenAI(temperature=0), chain_type="stuff")
query = Uquery +"/###/␣you␣are␣talking␣to␣a␣robot␣who..."
commands = chain.run(input_documents=docs, question=Uquery)
```

3.7 Command Execution Simulation

The generated commands are parsed and simulated, showcasing their potential for real-world robotic actions.

```
str_list = commands.split(',␣')
for i in range(len(str_list)):
    command = str_list[i % len(str_list)]
    # ... (rest of the command execution simulation code)
```

4 Results and Discussion

In our exploration of robotic task planning using extensive text files and trails, several pivotal findings emerged. Given the voluminous nature of the text file, potentially can span hundreds of pages, a primary recommendation is the strategic clustering of analogous commands. This not only enhances data manageability but also augments the efficiency of the subsequent retrieval processes.

The role of the vector database in this framework is multifaceted. It functions as a dynamic memory retriever, housing both specific commands and a repository of previously successful examples. This reservoir equips the LLM with a rich reference, enabling it to generate apt low-level instructions when presented with high-level directives. Furthermore, the text file serves as an exhaustive memory archive, meticulously cataloging the robot's capabilities and encapsulating its historical experiences. This dual-layered memory system—combining immediate retrieval and comprehensive archival—ensures that the LLM's responses are both contextually apt and operationally feasible. The reason we believe is that when the similarly search engine retrieves a vector that corresponds to the question or command it retrieves the chunk and if the chunk is large enough it gives all the context needed. Further research is needed to conclude the best text file structure.

Upon retrieving the pertinent memory vector, this information is seamlessly integrated into the LLM's prompt, enriching its context. It's noteworthy that there exists a trade-off between the granularity of the text chunks and the accuracy of the prompt. While smaller chunks might offer more detailed "memory" recall, they could potentially compromise the prompt's accuracy. Conversely, larger chunks, though potentially sacrificing some memory detail, might yield more accurate prompts. This balance can be iteratively fine-tuned through rigorous experiments and trials, optimizing the system for diverse operational scenarios.

5 Conclusion

Recap of the main findings. The potential impact on the field of robotics. Future directions and improvements.

6 References

List of academic papers, articles, and other sources referred to in the paper. @INPROCEEDINGS10161317, author=Singh, Ishika and Blukis, Valts and Mousavian, Arsalan and Goyal, Ankit and Xu, Danfei and Tremblay, Jonathan and Fox, Dieter and Thomason, Jesse and Garg, Animesh, booktitle=2023 IEEE International Conference on Robotics and Automation (ICRA), title=ProgPrompt: Generating Situated Robot Task Plans using Large Language Models, year=2023, volume=, number=, pages=11523-11530, doi=10.1109/ICRA48891.2023