Tool Pre-Selection Using Embeddings

André Baltazar
me@andrebaltazar.com

Abstract

As the ecosystem of tools and agents available for integration with Large Language Models (LLMs) expands, selecting the most relevant tools for a given task becomes critical for efficiency and improving interaction quality. This paper explores the use of vector-based search and embeddings to pre-select tools from a large pool (e.g., 100+ tools), ensuring a more targeted and contextually relevant interaction. Additionally, this approach can be extended to agent orchestration systems, such as OpenAI's Swarm [1], to optimize handoffs between agents in complex multi-agent scenarios.

1. Introduction

Modern LLMs are increasingly being augmented with external tools to extend their capabilities, such as retrieving real-time information, performing complex calculations, or manipulating structured data. While maintaining a large pool of tools does not inherently create computational overhead, passing an excessive number of tools into the LLM can lead to context length constraints and inefficiencies. Efficiently narrowing down the pool of tools to those most relevant to a user's query is crucial to ensure seamless interaction and reduce token usage.

Vector-based search, leveraging embeddings to represent tools and user queries in a shared high-dimensional space, provides a scalable and efficient mechanism for this pre-selection. Techniques such as Hypothetical Document Embeddings (HyDE) [2] can further enhance the matching process by generating hypothetical intermediate outputs to better align queries with tool capabilities.

In addition, systems like OpenAI's Swarm, which orchestrate the actions of multiple specialized agents, face similar challenges of scalability and relevance. Embedding-based approaches can be extended to optimize agent selection in these multi-agent environments.

2. Methodology

2.1 Tool Embedding Generation

Each tool is described using its metadata, documentation, and sample use cases. These descriptions are converted into embeddings using transformer-based models (e.g., OpenAI embeddings or Sentence-BERT). The tool representation may vary, but the core idea is to capture the tool's functionality, parameters, and potential use cases in a way that is easy to understand, and create a relevant embedding. The embedding process could include:

- Metadata Representation: Capturing the tool's core functionality, parameters, and potential use cases
- Contextual Examples: Generating short synthetic examples of tool usage to enrich the embedding

2.2 Query Embedding and HyDE

When a user provides a query, it is embedded into the same vector space as the tools. Optionally, the Hypothetical Document Embeddings (HyDE) technique generates a plausible response to the query and embeds this response. The final query embedding can then be a blend of the direct query

embedding and the HyDE-enhanced embedding, improving alignment with tool capabilities.

2.3 Vector-Based Matching

The cosine similarity between the query embedding and each tool embedding is computed. Tools are ranked based on similarity scores, and the top results are selected. These tools can either be directly passed to the LLM or used as input for further decision-making processes.

2.4 Extension to Agent-Orchestration Systems

In multi-agent systems like OpenAI's Swarm, each agent's capabilities and domain expertise can be embedded using a similar methodology. Tasks or queries are matched to the most relevant agents, enabling efficient decomposition and assignment in complex scenarios.

3. Applications

3.1 Pre-Selection of Tools for LLMs

The described approach allows systems to pre-select a subset of tools to pass into the LLM, ensuring that only the most relevant options occupy the limited context space. This prevents wasting tokens on irrelevant tools and allows the LLM to focus on producing high-quality results.

3.2 Agent Selection in Multi-Agent Frameworks

Embedding-based filtering is also used to dynamically assign tasks to the best-suited agents in systems like OpenAI's Swarm. This approach can further optimize multi-agent systems by ensuring that each agent is utilized according to its strengths and capabilities.

4. Implementation

A practical implementation of the vector-based search approach has been developed and tested. The system successfully demonstrates the effectiveness of embedding-based tool selection in reducing context size and improving response relevance.

The implementation uses modern embedding models to generate high-quality vector representations of both tools and queries. Detailed pseudocode for the implementation is provided in Appendix A, which covers the core components including tool embedding generation, query processing with optional HyDE enhancement, and similarity-based ranking.

5. Experiments

An experiment was conducted to evaluate the effectiveness of the vector-based tool selection approach. The system was tested with a configurable number of tools (default 100) and test queries (default 400). These tools and queries were generated by AI, and minimal processing was done to curate them.

Each experiment followed these steps:

1. Tool Embedding: All tools were embedded into vector space using the Nomic Embed Text model.
2. Query Processing: Each query was passed through an HyDE pass to generate an hypothetical description of the tool that would be used to answer the query.
3. Vector-Based Matching: The cosine similarity between the HyDE-enhanced query embedding and each tool embedding was computed and the top 5 tools were selected based on cosine similarity.
4. Metrics collected: Token usage (measuring the number of tokens required to represent the selected tools versus the full tool set) and selection accuracy (determining whether the expected tool was included in the vector-selected subset).

5.1 Results

The pre-selection obviously results in a very high reduction in terms of token usage, because reducing from 100 to 5 tools would reduce the token usage by 95% for most queries if we consider the tools to have around the same size in tokens. The interesting part is attempting to maintain a high selection accuracy while reducing the token usage, which makes it possible to run in smaller models.

The results show that the vector-based approach is able to maintain a high selection accuracy even when the number of tools is increased. In the majority of the test runs, the expected tool was in the top 5 results. This accounts for more than 94% of the test queries. Using HyDE, the selection accuracy was able to reach higher accuracy, sometimes more than 98%, for runs of 400 queries, especially when the queries were trickier.

For example, if the query was "Hash the contents: 'How's the weather today?'", the expected tool would be a hashing tool, instead of a weather tool. These are the cases where the HyDE technique is better than just using the plain query embedding.

The results were obtained by running over multiple runs of 400 queries with 100 tools. The code for the experiment, with one example test run, is available at https://github.com/AndreBaltazar8/tool-pre-selection.

5.2 Discussion

The experimental results were obtained using a limited dataset of AI-generated tools and queries, which introduces potential sampling bias as these may not fully represent real-world use cases. Nevertheless, the findings demonstrate that the vector-based approach successfully maintains high selection accuracy as the toolset scales, while achieving substantial reductions in token utilization.

The integration of HyDE in the query embedding pipeline introduces additional computational overhead but yields improved selection accuracy. With appropriate prompt engineering, the technique demonstrates robust performance even when using smaller models.

Smaller models like 3B parameters still result in a good selection accuracy, which makes it possible to run HyDE at an higher token/s and increase the selection accuracy.

Further research opportunities include evaluating the approach using a more comprehensive, manually curated dataset of tools and queries. Additionally, empirical validation through end-to-end testing with LLM-based tool execution would provide valuable insights into the practical impact of reducing the tool context from 100 to 5 options. It could also be interesting to normalize the description of the tools in regards to what the HyDE generates, which would increase the selection accuracy even more.

6. Conclusion

Vector-based search and embedding techniques provide an efficient mechanism for narrowing down large pools of tools or agents in LLM-driven systems. By leveraging techniques like HyDE and embedding-based similarity, these systems dynamically identify the most relevant tools or agents for a given task.

This approach has been successfully implemented in real-world systems to pre-select tools that are passed into the LLM or, in some cases, to outright select the best tool for the task.

Acknowledgments

I want to express my deep gratitude to my good friend, José Camacho, for our insightful discussions on these topics. He pointed me to the HyDE technique and suggested it as an approach for achieving more relevant results when selecting tools, compared to relying solely on embeddings of the input query.

References

1. OpenAI Cookbook, 'Orchestrating Agents with Swarm', https://cookbook.openai.com/examples/orchestrating_agents

2. Gao et al., 'Precise Zero-Shot Dense Retrieval without Relevance Labels', arXiv:2212.10496, https://doi.org/10.48550/arXiv.2212.10496

Disclaimer

This paper was co-written by AI with additional guidance from the human editor. The pseudocode in Appendix A was fully written with AI assistance.

Appendix A: Implementation

Below is a high-level pseudocode representation of how vector-based search can be used to select relevant tools:

```
# Step 1: Embed all tools into a shared vector space
def generate_tool_embeddings(tool_descriptions, embedding_model):
    tool_embeddings = {}
    for tool_name, description in tool_descriptions.items():
        embedding = embedding_model.encode(description)
        tool_embeddings[tool_name] = embedding
    return tool_embeddings

# Step 2: Embed the user query and (optionally) use HyDE
def generate_query_embedding(query, embedding_model, use_hyde=False):
    if use_hyde:
        hypothetical_response = generate_hypothetical_response(query)
        query_embedding = embedding_model.encode(hypothetical_response)
    else:
        query_embedding = embedding_model.encode(query)
    return query_embedding

# Step 3: Calculate cosine similarity between query and tools
def rank_tools(query_embedding, tool_embeddings):
    similarities = {}
    for tool_name, tool_embedding in tool_embeddings.items():
        similarity = cosine_similarity(query_embedding, tool_embedding)
        similarities[tool_name] = similarity
    return sorted(similarities.items(), key=lambda x: x[1], reverse=True)

# Step 4: Select top-N tools to pass to the LLM
def select_relevant_tools(query, tool_descriptions, embedding_model, top_n=5, use_hyde=False):
    tool_embeddings = generate_tool_embeddings(tool_descriptions, embedding_model)
    query_embedding = generate_query_embedding(query, embedding_model, use_hyde)
    ranked_tools = rank_tools(query_embedding, tool_embeddings)
    return [tool for tool, score in ranked_tools[:top_n]]

# Example usage
tool_descriptions = {
    'WeatherAPI': 'Provides real-time weather data for a given location.',
    'Calculator': 'Performs basic and advanced mathematical calculations.',
    'Translator': 'Translates text between multiple languages.',
    # Add more tools...
}

query = "What's the weather in New York?"
embedding_model = SomeEmbeddingModel()  # Use your embedding model here
selected_tools = select_relevant_tools(
    query,
    tool_descriptions,
    embedding_model,
    top_n=3,
    use_hyde=True
)

print('Selected tools:', selected_tools)
```

This pseudocode demonstrates how tools can be pre-selected based on their embeddings and the similarity to a query. Similar logic applies to agent selection, with agent descriptions replacing tool descriptions.