April 24, 2024

Aparicio Jorge
Fusi Gabriele
Troiani Gian Maria

Machine Learning Foundations

Final Project:
Advanced Topics in Large Language Models

Agents are a crucial advancement in the development of more useful and better-performing Large Language Model (LLM) applications. While most people use LLMs via zero-shot prompting, generating output token by token without revision, it has been observed that agentic workflows can significantly enhance performance. In fact, agentic workflows using GPT-3.5 can even surpass GPT-4 zero-shot performance on the same task, as measured by the HumanEval benchmark [1] [Exhibit A]. Furthermore, agentic workflows extend the usefulness of LLM applications beyond mere text generation, creating more valuable applications that leverage LLMs' reasoning capabilities to power generative AI automation.

The primary aim of our project is to empower an AI agent with the ability to interact with textual data in a manner that mirrors human comprehension and productivity, requiring minimal manual intervention. To accomplish this objective, we created a sample agentic LLM application. We devised three tools the agent can use: download_pdf, summarize_paper, and create_file. These tools collectively enable the AI agent to efficiently collect, distill, and synthesize information from various textual sources, such as arXiv, thereby facilitating rapid data assimilation for users.

At the heart of our endeavor lies the utilization of Large Language Models (LLMs), sophisticated neural network architectures trained on extensive text corpora. Models like OpenAI's GPT (Generative Pre-trained Transformer) exemplify the remarkable capacity of LLMs to comprehend and generate human-like text across diverse contexts and languages. By harnessing the vast knowledge encoded within LLMs, our project aims to equip the AI agent with a profound understanding of textual data, enabling it to extract insights, generate summaries, and engage in coherent interactions with users.

Large Language Models (LLMs)

The modus operandi of LLMs encompasses several pivotal steps:

1. **Data Preprocessing:** The input textual corpus undergoes tokenization, segmenting it into discrete units such as words or subwords, each assigned a numerical representation.
2. **Model Architecture:** LLMs typically employ transformer-based architectures renowned for their efficacy in handling long-range dependencies within text sequences, as pioneered by Vaswani et al. (2017) [0].
3. **Training Regimen:** LLM training involves supervised learning, where the model predicts the next token in a sequence based on preceding tokens, iteratively adjusting parameters to minimize prediction errors.
4. **Fine-Tuning:** After initial training on a large-scale dataset, LLMs undergo fine-tuning on specific tasks or domains to enhance performance, further training on task-specific datasets to adapt to the target domain.
5. **Inference Mechanism:** During inference, the trained LLM generates text by sampling from learned probability distributions over tokens, predicting the most probable continuation based on input prompts or context.

Retrieval-Augmented Generation (RAG)

LLMs are typically trained on vast datasets covering diverse topics but lack access to specific data beyond their training cutoff point, rendering them static and prone to errors when queried about untrained data. Retrieval-Augmented Generation (RAG) addresses this limitation by enabling organizations to provide their data as part of the query, augmenting the LLM's understanding with relevant real-time information for up-to-date and accurate responses tailored to specific domains.

RAG facilitates:

1. **Question and answer chatbots:** Integration of LLMs with chatbots yields more accurate responses derived from company documents and knowledge bases.
2. **Search augmentation:** LLM-generated answers enhance informational queries on search engines, improving user experience.
3. **Knowledge engine:** Using company data as context for LLMs enables employees to easily obtain answers to specific questions, such as HR inquiries or compliance-related queries.

The RAG process involves:

1. **Data Preparation:** Collection and preprocessing of data, including handling Personally Identifiable Information (PII) and chunking documents appropriately for indexing and retrieval.
2. **Indexing Relevant Data:** Generating document embeddings to populate a Vector Search index for efficient similarity searches.
3. **Retrieving Relevant Data:** Retrieving parts of indexed data relevant to user queries to enrich the LLM's understanding and improve response quality.
4. **Building LLM Applications:** Wrapping components for prompt augmentation and querying the LLM into an endpoint for integration with various applications, such as Q&A chatbots.

Additionally, Databricks recommends key architectural elements for RAG implementations, including vector databases for fast similarity searches, MLflow LLM Deployments or Model Serving for standardized interfaces with third-party language model APIs, and deployment of fine-tuned models and environments to reduce dependencies on external services.

In essence, RAG amalgamates language model capabilities with external knowledge retrieval mechanisms to generate contextually relevant and informative responses, involving data preparation, indexing, retrieval, and effective application building.

AI Agents

An AI agent serves as an autonomous entity perceiving its environment through sensors and acting upon it through actuators to achieve specific objectives. These agents, typically implemented using computational systems, make decisions based on available information and past experiences.

In technical parlance, an AI agent operates within an environment, which may be discrete or continuous, deterministic or stochastic. It receives observations from the environment, processes them internally (e.g., through knowledge representation, reasoning, and decision-making algorithms), and selects actions based on its current state and observed environment, guided by a utility function or objective function it seeks to optimize over time.

Various types of AI agents exist, including simple reflex agents, model-based reflex agents, goal-based agents, and utility-based agents, each differing in sophistication regarding internal mechanisms and decision-making capabilities. In this paper, every time we will use the word “agent” to refer specifically to “LLM agents”.

Agents are a crucial advancement in the development of more useful and better-performing Large Language Model (LLM) applications. While most people use LLMs via zero-shot prompting, generating output token by token without revision, it has been observed that agentic workflows can significantly enhance performance. In fact, agentic workflows using GPT-3.5 can even surpass GPT-4 zero-shot performance on the same task, as measured by the HumanEval benchmark [1] [Exhibit A]. Furthermore, agentic workflows extend the usefulness of LLM applications beyond mere text generation, creating more valuable applications that leverage LLMs' reasoning capabilities to power generative AI automation.

Agentic Design Patterns - Overview

Andrew Ng and his team at AI Fund provide a useful framework to categorize agentic design patterns [2]:

1) **Reflection:** the LLM considers its own previous output to enhance it. This technique is easy, quick to implement and can lead to improved performance and decreased hallucinations in the final output. It can be repeated multiple times and can involve one or more agents.

   Example: *Here's code intended for task X: [previously generated code]*
   Check the code carefully for correctness, style, and efficiency, and give constructive criticism for how to improve it.

2) **Tool Use:** the LLM is given *tools* to help it "take actions" (examples: web search, code execution, etc…). Please note that although "tools" are a good metaphor to convey the concept, this is not completely accurate. The mechanism happening behind the scenes is LLM "function calling".

   Example: *Asking an LLM to provide a weather forecast for a specific location by calling a weather API. The LLM application would use a tool to talk to a weather API to extract relevant data, and retrieve the weather forecast tailored to the user's question.*

3) **Planning:** the LLM breaks down a larger task into smaller ones and autonomously decides on the sequence of steps to accomplish the goal. Most complex tasks cannot be accomplished in a single step or with a single tool invocation, but an agent can decide which steps to take.

   Example: *Asking an LLM to do online research on a given topic, asking to break down the objective into smaller subtasks, such as researching specific subtopics, synthesizing findings, and compiling a report.*

4) **Multi-Agent Collaboration:** multiple LLM agents work together to provide better solutions than a single agent would be able to provide.

   Example: *Asking an LLM to develop a piece of software by employing multiple agents to collaborate on different aspects of the project, such as defining requirements, writing code, designing user interfaces, and testing for bugs, with each agent contributing their specialized skills and knowledge to create a cohesive and functional final product.*

Combining the four agentic design patterns—Reflection, Tool Use, Planning, and Multi-Agent Collaboration—is crucial for creating sophisticated and capable agentic workflows. Each pattern brings unique strengths and challenges, and their integration allows for more robust and adaptable LLM applications.

Agentic Design Patterns - In Depth

Reflection and Tool Use are more established patterns, with proven track records of enhancing LLM performance. Reflection enables the LLM to iteratively refine its outputs by critiquing and improving upon its own generated content. This self-assessment helps to catch errors, reduce hallucinations, and produce more coherent and relevant results.

Tool Use, on the other hand, empowers LLMs to interact with external resources and APIs, greatly expanding their knowledge and capabilities beyond their training data. By calling functions to access real-time information, execute code, or perform specific tasks, LLMs can provide more accurate and dynamic responses to user queries. The more tools an agent has, the more actions it can take.

To provide the reader with a more comprehensive understanding of Tool Use, a particularly crucial agentic design pattern, we will delve into its inner workings and explain the process in greater detail in the following section.

Below, we explore the sequential steps and workflow that enable an LLM to invoke tools:

1. **Tool Definition –** Defines tools as functions with specific required inputs and return values. Each tool function is described by a standardized schema (usually json) that the LLM can understand. The schema includes function name, description, required input and output type.
2. **Prompt Construction** – Makes a request to the LLM that contains both the user query and the tools defined. This makes the LLM aware of the tools it can 'access' in the form of tool functions.
3. **LLM Reasoning –** Processes the prompt and determines if and when to call a tool function based on its understanding of the query and the function descriptions. If the LLM calls a tool, it structures the output in a way that includes function name and arguments.
4. **Function Parsing –** This step parses the LLM response and is parsed by the user's application to extract function name and arguments.
5. **Function Execution –** The parsed function is extracted, matched against the defined tool functions, and the corresponding function is invoked with the provided arguments. This is the step where the API call, database query, or other external interaction happens. The function is actually executed by the user's application, not by the LLM.
6. **Observation Processing –** The user's application captures the result of the tool function and formats it into an 'observation' that is fed back to the LLM as additional context. This allows the LLM to be updated with the outcome of its previously chosen action and factor it into its current reasoning.
7. **Status Update –** The function output is appended to the conversation history, and the LLM informs the user in its response of what tools were used and whether their use was successful.
8. **Loop –** LLM is prompted again to continue reasoning based on the updated context. This loop of reasoning, function calling, and observation processing continues until the LLM determines it has completed the task and provides a final response.

The remaining two designing patterns are Planning and Multi-Agent Collaboration represent the cutting edge of agentic workflows, offering immense potential for tackling complex, multi-step problems.

Planning allows an LLM to break down a high-level goal into a series of sub-tasks and autonomously determine the optimal sequence of actions to achieve the desired outcome. This enables LLMs to handle more open-ended and ambiguous requests, adapting their approach based on the specific context and available tools. The main approaches to planning in agentic language models are:

1. **Task Decomposition** [4]: a. Decomposition-First Methods: The task is decomposed into sub-goals first, and then planning is done for each sub-goal successively. b. Interleaved Decomposition Methods: Task decomposition and sub-task planning are interleaved, with each decomposition revealing only one or two sub-tasks at the current state.
2. **Multi-plan Selection** [5] : a. Multi-Plan Generation: Multiple paths of plans are generated to form a candidate plan set. b. Optimal Plan Selection: Various strategies, such as majority voting or tree search algorithms, are used to select the optimal plan from the candidate set.

These approaches align with the idea of breaking down high-level goals and determining the optimal sequence of actions to achieve the desired outcome.

Multi-Agent Collaboration takes this a step further by enabling multiple specialized LLMs to work together, leveraging their individual strengths and knowledge to solve problems more effectively than a single agent could. In Multi-Agent Collaboration, well-designed prompts are crucial for establishing effective communication, coordination, and cooperation among LLM agents.

Prompts serve key purposes such as role assignment, communication protocols, objective alignment, and collaboration dynamics. They provide the necessary context, objectives, and guidelines for agents to engage in productive interactions and work towards a common goal [3]. Multi-Agent Collaboration can also involve a hierarchy of agents with different levels of authority, responsibility, and specialization. This hierarchical structure typically includes supervisory agents, specialized agents, coordinator agents, and worker agents, each with specific roles and contributions to the collaboration. The hierarchy helps manage the complexity of the collaboration and ensures effective coordination among agents, with prompts reflecting this structure and providing clear instructions for each level of agents.

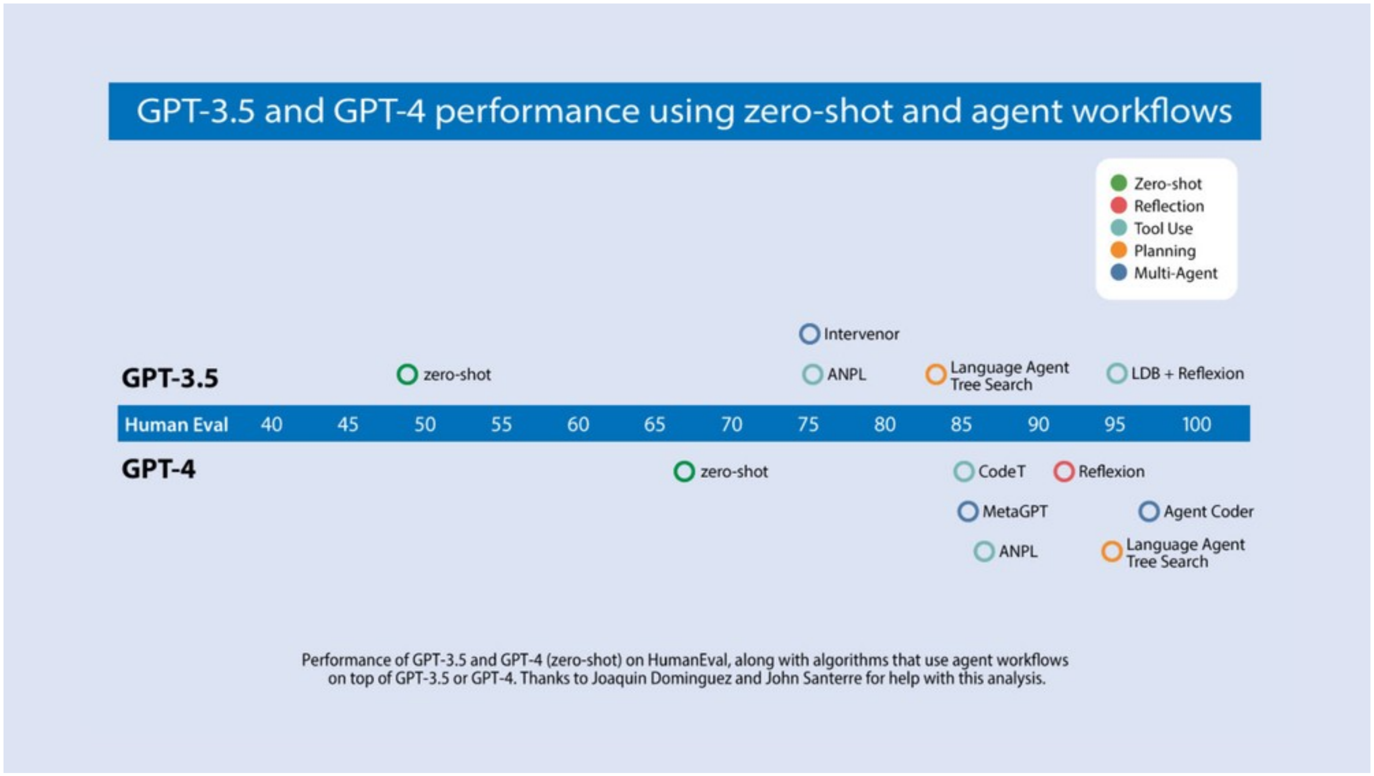
While Planning and Multi-Agent Collaboration can be extremely powerful, the trade-off is that Planning and Multi-Agent Collaboration are less predictable and harder to control than Reflection and Tool Use. As LLMs gain more autonomy in deciding their actions and collaborating with other agents, ensuring alignment with human goals and values becomes more challenging. Nonetheless, the potential benefits of these advanced patterns are immense, enabling LLMs to tackle real-world problems with greater flexibility and intelligence.

In conclusion, the emergence of agentic design patterns has revolutionized the development of Large Language Model (LLM) applications, enabling them to tackle complex, real-world problems with unprecedented effectiveness and automation capabilities. By harnessing the power of Reflection, Tool Use, Planning, and Multi-Agent Collaboration, LLM agents can break down high-level goals into manageable sub-tasks, take actions, interact with external resources, collaborate to generate contextually relevant and informative responses and take actions.

The sample agentic LLM application presented in the Appendix of this paper demonstrates the potential of these techniques, showcasing how a simple AI agent equipped with tools like download_pdf, summarize_paper, and create_file can efficiently collect, distill, and synthesize information from various textual sources in a semi-autonomous fashion.

As research in this field continues to advance, we can expect to see increasingly sophisticated and reliable agentic workflows that transform various domains, from scientific research and creative industries to business operations and public policy. However, it is crucial to address the challenges associated with these advanced patterns, such as ensuring alignment with human goals and values, managing computational costs, and developing robust evaluation frameworks. By carefully designing prompts, establishing suitable agent hierarchies, and incorporating techniques like Retrieval-Augmented Generation (RAG), we can harness the full potential of LLM agents while mitigating potential risks. The future of AI lies in the seamless integration of LLMs, agentic design patterns, and external knowledge retrieval mechanisms, paving the way for a new era of intelligent, adaptive, and human-centric applications.

Exhibits:



(Exhibit A)

Sources:

[0] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[1] Ng, A. (2024, March 20). Four AI agent strategies that improve GPT-4 and GPT-3.5 performance. DeepLearning.AI. https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/?ref=dl-staging-website.ghost.io

[2] ibid

[3] Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). Communicative Agents for Software Development. arXiv preprint arXiv:2305.03819.

[4] Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., & Chen, E. (n.d.). Understanding the planning of LLM agents: A survey.

[5] ibid