Phi-3-MLX Agent and Toolchain:
A Flexible Framework for AI Task Execution

Josef Albers

July 19, 2024

Abstract

This paper presents an in-depth analysis of a novel Agent and Toolchain system, a flexible framework designed for AI task execution. This system introduces a unique approach to defining and executing AI workflows, centered around a string-based toolchain definition and a versatile agent executor. We explore its architecture, focusing on the lightweight toolchain representation, dynamic parsing mechanism, and efficient execution flow. The paper contextualizes this system within the broader landscape of AI agent frameworks, comparing it with existing approaches. We discuss the system's advantages, including its lightweight design, model agnosticism, and adaptability to various AI tasks. Finally, we examine the implications for tool design and potential applications across different domains.

Contents

1 Abstract

This paper presents an in-depth analysis of a novel Agent and Toolchain system, a flexible framework designed for AI task execution. This system introduces a unique approach to defining and executing AI workflows, centered around a string-based toolchain definition and a versatile agent executor. We explore its architecture, focusing on the lightweight toolchain representation, dynamic parsing mechanism, and efficient execution flow. The paper contextualizes this system within the broader landscape of AI agent frameworks, comparing it with existing approaches. We discuss the system's advantages, including its lightweight design, model agnosticism, and adaptability to various AI tasks. Finally, we examine the implications for tool design and potential applications across different domains.

2 Introduction

As artificial intelligence continues to advance, there is a growing need for flexible frameworks that can orchestrate complex, multi-step tasks involving large language models (LLMs) and other AI components. The Agent and Toolchain system presented in this paper represents a significant step forward in this domain, offering a unique blend of simplicity, flexibility, and efficiency.

This system introduces a novel approach to defining and executing AI workflows. At its core are two key components:

1. **Agent**: A versatile executor that manages the flow of data and control between various AI tools and models.
2. **Toolchain**: A simple, string-based definition of workflows that can be easily created, modified, and understood by both humans and machines.

The Agent and Toolchain system introduces several key features to the landscape of AI task execution:

- **Lightweight Workflow Definition**: The system uses a string-based representation for defining workflows, offering a simple yet powerful way to create and modify AI task sequences.
- **Flexible Integration**: Designed with adaptability in mind, the system aims to easily incorporate various AI models and tools as they evolve.
- **Consistent Abstraction**: By providing a uniform interface for diverse AI operations, the system seeks to simplify the development of complex, multi-step AI workflows.

This paper aims to provide a comprehensive overview of the Agent and Toolchain system, exploring its design philosophy, key components, and operational mechanisms. We will delve into the system's approach to defining and executing AI workflows, its flexible integration capabilities with various AI models and tools, and its efficient execution strategy.

By examining this system in detail, we seek to contribute to the ongoing discussion on AI agent architectures and provide insights that could inform future developments in this rapidly evolving field. The Agent and Toolchain system represents an alternative approach to AI task execution, potentially opening new avenues for both researchers and practitioners in the AI community.

3 Background and Related Work

To fully appreciate the innovations of the Agent and Toolchain system, it is essential to understand the broader context of AI agent frameworks and their approaches to task execution.

3.1 Task-Oriented AI Frameworks

Several frameworks have been developed to facilitate the creation and execution of AI-driven tasks:

- **LangChain**: Combines LLMs with a set of tools to accomplish tasks through thought-action cycles. It offers various agent types and provides numerous pre-built tools along with easy integration of custom tools.
- **Hugging Face Transformers Agents**: Tightly integrated with the Hugging Face ecosystem, this framework includes different agent types for specific tasks like code generation and JSON manipulation. It implements a secure Python interpreter for code execution and offers granular control over system prompts and tool creation.
- **OpenAI's Function Calling**: While not a full agent system, this feature allows language models to interact with predefined functions in a structured manner, integrating function definitions as JSON schemas directly into the model's output.
- **AutoGen**: Developed by Microsoft, this framework emphasizes a multi-agent approach to task solving, implementing sophisticated conversation management features for inter-agent communication.

3.2 Workflow Definition Approaches

Different frameworks employ various methods for defining task workflows:

- **Python-based Definition**: Many frameworks use Python classes and methods to define workflows, offering flexibility but requiring programming knowledge.
- **YAML or JSON Configuration**: Some systems use configuration files to define workflows, providing a more declarative approach but potentially limiting expressiveness.
- **Natural Language Instructions**: Emerging approaches allow users to define workflows using natural language, which are then interpreted by AI to create executable plans.

3.3 Execution Models

Task execution strategies vary across frameworks:

- **Sequential Execution**: Tasks are executed in a predefined order, with each step completing before the next begins.
- **Reactive Execution**: The system dynamically chooses the next action based on the outcome of previous steps, allowing for more adaptive workflows.
- **Parallel Execution**: Some frameworks support concurrent execution of independent tasks, improving efficiency for certain workflows.

3.4 Tool Integration

The method of integrating external tools and APIs is a key differentiator among frameworks:

- **Built-in Tool Libraries**: Many frameworks come with pre-defined sets of tools for common tasks.
- **Custom Tool Definition**: Most systems allow users to define custom tools, typically as Python functions with specific input/output structures.

- **API Wrapping**: Some frameworks provide mechanisms to easily wrap external APIs as tools that can be used within workflows.

3.5 State Management

How frameworks manage and pass state between different steps of a workflow is crucial:

- **Global State**: Some systems maintain a global state that all tools can access and modify.
- **Chained Outputs**: Other frameworks pass the output of one tool as the input to the next, creating a chain of transformations.
- **Context Objects**: More sophisticated systems use dedicated context objects to manage state, allowing for more controlled data flow between components.

Understanding these different approaches provides context for evaluating the unique contributions of the Agent and Toolchain system presented in this paper, particularly its string-based workflow definition and flexible execution model.

4 Phi-3-MLX Agent and Toolchain System

The Phi-3-MLX Agent and toolchain system enters this diverse landscape with its own unique approach, designed to offer a balance of simplicity, flexibility, and performance.

4.1 System Architecture

The Agent system consists of several key components:

1. **Agent**: The core component that manages the execution of tasks and interactions with tools. It encapsulates a workflow and provides a consistent interface regardless of its internal implementation.
2. **Toolchain**: A string-based definition of the workflow to be executed.
3. **Tools**: Functions that can be called by the agent to perform specific tasks.
4. **Parser**: Responsible for interpreting the toolchain string and converting it into executable steps.
5. **Execution Engine**: Manages the flow of data and control between tools during task execution.

4.2 Agent Interface and Abstraction

The Agent system provides a powerful abstraction for encapsulating AI workflows:

- **Consistent Interface**: Each agent, regardless of its internal complexity, accepts a prompt (and optionally images) as input and produces responses and files as output.
- **Encapsulation**: The internal workings of an agent can involve any mix of Python logic, LLM calls, API interactions, or other operations, all encapsulated within the agent abstraction.
- **Composability**: Agents can be chained or organized into hierarchical structures, allowing for complex workflows to be built from simpler components.
- **Flexibility**: The system can easily accommodate different backends (e.g., replacing `generate()` with `mistral_api()`) without changing the overall agent structure or interface.

Input and Output Structure:

- **Input**:
  - `prompt`: A string containing the task or query for the agent.
  - `images` (optional): Any image data relevant to the task.
- **Output**:
  - `responses`: String or list of strings containing the agent's textual output.
  - `files` (optional): Similar to Claude AI's artifact system, this can include any non-textual outputs or files generated during the task execution.

This structure allows for a wide range of tasks, from simple text generation to complex multi-modal operations, all within a consistent framework.

4.3 String-Based Toolchain Definition

One of the most distinctive features of the Phi-3-MLX's Agent system is its use of string-based toolchain definitions. This approach offers several advantages:

- **Readability**: Toolchains are defined in a format that is easy for humans to read and understand.
- **Flexibility**: Developers can quickly modify or create new toolchains without complex programming.
- **Lightweight**: The string-based approach minimizes overhead in toolchain definition and modification.

A typical toolchain definition might look like this:

```
toolchain = """
    prompt = add_code(prompt, codes)
    responses = generate(prompt, images)
    files, codes = execute(responses, step)
"""
```

This simple string defines a sequence of operations, their inputs, and their outputs.

4.4 Parsing Mechanism

The parsing mechanism is crucial to translating the string-based toolchain into executable steps. The process involves:

1. **String Splitting**: The toolchain string is split into individual lines, each representing a single operation.
2. **Line Parsing**: Each line is parsed to extract:
   - Output variables
   - Function name
   - Input parameters
3. **Function Resolution**: The agent resolves function names to actual callable objects.
4. **Creating Executable Steps**: The parsed information is used to create a list of executable steps.

4.5 Execution Flow

The execution of a parsed toolchain is a key part of the Phi-3-MLX system's functionality. The core execution loop looks like this:

```
for tool in self.toolchain:
    _returned = tool['fxn'](*[self.ongoing.get(i, None) for i in tool['args']],
                            **{k:v for k,v in self.kwargs.items()
                      if k in inspect.signature(tool['fxn']).parameters.keys()})
    if isinstance(_returned, dict):
        self.ongoing.update({k:_returned[k] for k in tool['out']})
    else:
        self.ongoing.update({k:_returned for k in tool['out']})
```

This execution mechanism provides several key features:

- **Dynamic Argument Passing**: Arguments for each tool are dynamically retrieved from the agent's ongoing state.
- **Flexible Keyword Argument Handling**: Only relevant keyword arguments are passed to each tool.
- **Precise Return Value Processing**: The system handles both dictionary and single value returns, mapping them correctly to the intended state variables.
- **State Management**: A persistent state is maintained across tool executions, allowing for information flow between steps.

This execution mechanism is framework-agnostic, allowing for great flexibility in tool implementation.

4.6 Core Features and Design Philosophy

The Phi-3-MLX Agent system is built on principles of flexibility, modularity, and ease of use. Its key features and design philosophy include:

1. **Cross-Platform Compatibility**:
    - The system is designed to work with various AI backends and models.
    - It offers compatibility with different frameworks and LLMs, allowing users to choose the most suitable options for their specific needs.
    - The system allows easy integration of different LLMs or API services without altering its core structure.
2. **Interchangeable Backend**:
    - The system can seamlessly work with various AI backends or API services.
    - For example, the `generate()` function can be easily replaced with alternatives like `mistral_api()`:

```
agent = Agent(toolchain = "responses, history = mistral_api(prompt,
 ↪  history)")
agent('Write a neurology ICU admission note')
```

- The modular design allows for easy swapping of components, such as using different LLMs or API services, without altering the fundamental agent structure.

3. **Modular and Composable Components**:
   - The system decomposes agent interactions into reusable components through its toolchain mechanism.
   - Agents can be chained or organized into hierarchical structures, allowing for complex workflows to be built from simpler components.
4. **Consistent Interface with Powerful Abstraction**:
   - Each agent, regardless of its internal complexity, accepts a prompt (and optionally images) as input and produces responses and files as output.
   - This consistent interface allows for easy composition of agents into more complex systems.
   - The file output is analogous to Claude AI's artifact system, enabling the agent to produce and work with non-textual data.
5. **String-Based Toolchain Definition**:
   - Toolchains are defined using a simple, readable string format.
   - This approach offers a balance between simplicity and power, making it easy to create and modify agent behaviors.
6. **Adaptability**:
   - The system's design allows it to be used for a wide range of AI tasks, from simple text generation to complex, multi-modal operations.
   - Its flexibility makes it suitable for various applications, including NLP tasks, multi-modal AI, automated reasoning, and interactive AI systems.

These features combine to create a system that is not only powerful and efficient but also highly adaptable to a wide range of AI tasks and development scenarios. The Phi-3-MLX Agent system provides a consistent, extensible framework for AI task execution, suitable for both simple applications and complex, multi-agent workflows.

5 Implications for Tool Design

The Phi-3-MLX system's architecture has important implications for how tools should be designed:

1. **Input Flexibility**: Tools should handle potentially missing inputs gracefully.
2. **Output Consistency**: Tools can return either dictionaries with expected keys or single values, with the toolchain specifying how outputs are mapped to the ongoing state.
3. **State Awareness**: Tools can be designed to read from and write to the ongoing state, enabling complex, multi-step operations.
4. **Parameter Matching**: Tool function parameters should match the expected input names in the toolchain definition.

6 Applications and Use Cases

The flexibility and efficiency of the Phi-3-MLX system make it suitable for a wide range of applications:

- **Natural Language Processing Tasks**: Text generation, summarization, translation, etc.
- **Multi-modal AI**: Combining text, image, and potentially other modalities in complex workflows.
- **Automated Reasoning**: Implementing step-by-step reasoning processes for problem-solving tasks.
- **Data Analysis Pipelines**: Creating flexible workflows for data processing and analysis.
- **Interactive AI Systems**: Building responsive AI assistants that can execute multi-step tasks based on user input.
- **Hierarchical Task Decomposition**: Complex tasks can be broken down into subtasks, each handled by a specialized agent, with a higher-level agent coordinating the overall process.
- **Multi-Agent Systems**: Creating systems where multiple agents with different specializations collaborate to solve complex problems.
- **Adaptive Workflows**: Developing workflows that can dynamically adjust their structure based on intermediate results or changing requirements.

7 Future Directions

While the Agent and Toolchain system represents a significant advance in AI task execution frameworks, several areas for future research and development emerge:

1. **Enhanced Tool Discovery**: Developing mechanisms for dynamically discovering and integrating new tools.
2. **Improved Error Handling**: Implementing more sophisticated error recovery and graceful degradation strategies.
3. **Multi-Agent Coordination**: Exploring ways to coordinate multiple agents for complex, distributed tasks.
4. **Learning from Execution**: Investigating methods for the system to learn and optimize toolchains based on past executions.
5. **Advanced Agent Orchestration**: Developing sophisticated methods for coordinating multiple agents in complex, hierarchical structures.
6. **Dynamic Toolchain Generation**: Exploring ways for agents to dynamically construct or modify their toolchains based on the task at hand.
7. **Secure Execution Environment**: Implementing a robust sandbox or containerized environment for safely executing code generated by LLMs or defined in toolchains. This is crucial for preventing potential security risks associated with running arbitrary code, especially in production environments.
8. **Code Analysis and Validation**: Developing pre-execution analysis tools to validate and sanitize code before execution, identifying potential security threats or unintended behaviors.
9. **Permissions and Access Control**: Creating a fine-grained permissions system that limits what resources and operations each tool or agent can access, enhancing overall system security.

10. **Audit Logging and Monitoring**: Implementing comprehensive logging and monitoring systems to track all executions, providing transparency and aiding in debugging and security analysis.

8 Conclusion

The Phi-3-MLX Agent and toolchain system introduces a powerful and flexible approach to AI task execution. Its framework-agnostic design, coupled with a consistent agent interface and string-based toolchain definition, offers a unique balance of simplicity, power, and adaptability.

By providing a uniform abstraction for diverse AI operations, from simple function calls to complex LLM interactions, the system enables the construction of sophisticated, multi-agent workflows while maintaining a clear and consistent structure. This approach not only simplifies the development of AI applications but also opens up new possibilities for creating adaptive, hierarchical AI systems capable of tackling complex, real-world problems.

As the field of AI continues to evolve, the principles embodied in the Phi-3-MLX system – modularity, flexibility, and clear abstraction – will likely play a crucial role in shaping the future of AI application development. The system's ability to integrate seamlessly with various backends and adapt to different computational environments positions it as a versatile tool for researchers and developers alike, potentially influencing the broader landscape of AI frameworks and methodologies.

9 References

1. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.
2. Harrison, H. (2023). LangChain. GitHub repository. https://github.com/hwchase17/langchain
3. Lhoest, Q., Villanova del Moral, L., & Hutter, F. (2023). Hugging Face Transformers Agent. GitHub repository. https://github.com/huggingface/transformers
4. OpenAI. (2023). Function calling. OpenAI Platform Documentation. https://platform.openai.com
5. Wu, S., & Zhou, H. (2023). AutoGen. GitHub repository. https://github.com/microsoft/autogen
6. Albers, J. (2024). Phi-3-Vision-MLX [Software]. GitHub. https://github.com/JosefAlbers/Phi-3-Vision-MLX