An Object Oriented Analysis of AI Software Frameworks

Kevin McMahon*, **Caleb Kumar***

[1] University of Colorado, Boulder
Office of the Registrar
Regent Administrative Center
20 UCB
Boulder, CO 80309

Abstract

Since the release of OpenAI's ChatGPT, GenerativeAI has taken the world by storm. GenerativeAI's ability to be used in software products is now growing immensely. New software frameworks with a focus on AI are becoming more and more popular. For these frameworks to integrate well into our current industry software practices they need to apply the same concepts and philosophies when being built. Import object-oriented concepts like inheritance, polymorphism, encapsulation, and abstraction, are being applied to AI frameworks to hopefully integrate them smoothly into industry workflows. Our paper will examine new and growing frameworks, LangChain and LlamaIndex, and their application/adherence to the previously stated OOP concepts. We will use a comparative analysis against current frameworks used for AI-focused software in the past, Elasticsearch and HuggingFace. We will compare these frameworks relative to their application of OOP concepts as well as their focus on making AI development easy and accessible for software developers and engineers.

Introduction

AI integration into software frameworks and workflows has quickly evolved with the development of AI models. As these models become more important to software applications across the software/tech industry, developers are turning more and more to frameworks that help with AI integration while maintaining the clarity, modularity, and reusability that are key points and highlights of object-oriented programming (OOP). OOP principles like inheritance, polymorphism, encapsulation, and abstraction, allow software to be structured in such a way that allows developers to manage complexity and larger scales, and recent AI frameworks are designed to hopefully use these concepts in their architecture. We will look into two main frameworks, LangChain and LlamaIndex, to evaluate how well they apply OOP principles and concepts in their design and implementation against currently used frameworks in the industry like HuggingFace Transformers and ElasticSearch, to understand how well they apply OOP concepts.

*These authors contributed equally.

Background and Related Work

As AI systems grow in complexity, software developers are turning more and more to object-oriented programming (OOP) to manage their AI software structure and scalability. The frameworks, LangChain and LlamaIndex show the trend of AI software frameworks looking to use OOP principles to encapsulate and abstract AI tasks and data workflows. LangChain gives developers the ability to create AI agents in software that have memory and can execute tasks, Langchain uses OOP concepts/principles to manage the agents' interactions with large language models (LLMs) and developers' custom-defined tools. LlamaIndex gives developers the ability to utilize large unstructured datasets like PDFs. LlamaIndex uses indexing and retrieving AI techniques to give developers access to large-scale unstructured data, employing OOP concepts to encapsulate and simplify data operations. Both frameworks offer reusable, well-structured approaches that align with OOP's goal of managing complex systems.

Previous work shows the growing overlap between software engineering practices and AI integration into software. For example, (Amershi et al. 2019) discusses how "modularity and maintainability" are redefined to meet AI's unique software structure demands. Frameworks like Hugging Face Transformers and ElasticSearch already show us how OOP concepts, like the four mentioned, can help developers interact with AI, minimizing the developer's need to manage underlying nuances. LangChain and LlamaIndex extend these ideas, offering specific solutions for AI workflows and data-centric software. This paper builds on this prior work by researching how these frameworks apply OOP principles to improve AI-based software development.

Overview of Frameworks

Langchain

LangChain is designed to allow the addition of AI and LLMs in software products, systems, and more. Its main goal is to create agents and agentic workflows, where agents act as autonomous pieces of software with LLMs as their brains, predefined tools, and external software or APIs to complete tasks. These agents maintain memory or "state", which represents the context of their past actions, decisions, and results from both.

LlamaIndex

LlamaIndex gives developers efficient document indexing and retrieval, specifically for unstructured datasets. It simplifies the process of creating and querying knowledge bases by connecting data sources to LLMs. This is now a common AI concept known as Retrieval Augmented Generation (RAG). LlamaIndex provides APIs for constructing various types of indices, such as tree-based or vector-based structures, to support different retrieval strategies.

HuggingFace Transformers

Hugging Face Transformers is a library that gives access to a wide variety of pre-trained and customer AI models for tasks such as text generation, classification, and translation. It has a high level of abstraction from nuances of fine-tuning and using AI models thus allowing developers to load and use models quickly and easily.

ElasticSearch

Elasticsearch allows software developers to manage and query large datasets through its design of a distributed architecture. Data is stored as JSON documents, which are indexed to allow searching. The framework uses a search engine that takes advantage of an optimized inverted index of the JSON data to allow for full-text search and filtering. This engine and indexing make it ideal for retrieving information based on keywords and specifically semantic similarity.

Analysis of Object-Oriented Principles in Frameworks

Langchain vs HuggingFace

Inheritance Inheritance, as defined in (Johnson and Foote 1991) is the concept that "Each class has a superclass from which it inherits operations and internal structure". HuggingFace uses inheritance to ensure consistency across models. Subclasses such as AutoModel and AutoTokenizer subclasses make the process of using different models and architectures easy. These subclasses reduce the need for redundant and duplicated code by inheriting base functionality like inference or differences in tokenization strategy to allow users to use any model or tokenizer they choose. Langchain uses inheritance to allow developers to code flexible and custom agents, as well as custom tools. Developers can inherit from the BaseAgent class to create their own custom AI agents with unique and custom-defined logic, while still inheriting important functionality like state/memory management, error handling, and error propagation. Langchain closely follows Inheritance as it focuses on extending the base API classes and modularity from base agents to be more customizable for developers. HuggingFace prioritizes simplicity so their API is not as extendable and only allows for simple extensions of their API.

Polymorphism In (Pierce 2002), Benjamin Pierce describes polymorphism as code to be used with multiple types, and Cardelli et al 1985, describe polymorphism as a function or code having multiple types with similar interfaces. Hugging Face uses polymorphism to allow different models to interact through shared interfaces. Models subclassed from AutoModel can perform inference using the same method signature, regardless of the underlying model architecture (like GPT or BERT). This allows developers to switch between models easily without needing to modify or change their code. LangChain uses polymorphism to bring workflows involving diverse agents and tools together. Various agent types, ConversationalAgent and ToolUsingAgent, inherit a common interface from BaseAgent, allowing them to work in the same system while also being able to implement custom logic. Tools that agents use also conform to shared interfaces, enabling easy use and changing of tools without modifying code. Both frameworks adhere well to polymorphism as they allow developers to create multiple instances of the same class or modules with different types and functionality.

Encapsulation (Pierce 2002) describes how encapsulation only lets the object or outside objects interact with data, methods, and others through a set interface. HuggingFace encapsulates the nuances and complexities of working with LLMs, like loading models, sharding, loading model checkpoints, tokenization, and inference, into more simple interfaces like frompretrained() and generate(). These encapsulations give developers all the information they need to accomplish the task at hand without needing to manage different complex tasks and data from different models or concepts in the framework. LangChain encapsulates software/-tool execution and workflows in its agent and tool interfaces. Agents manage their internal state with external tools, hiding these details from developers. For example, a ToolUsingAgent can invoke tools without requiring the developer to manually use the tool or allow the Agent to use it. Hugging Face adheres more strongly to encapsulation because of its focus on simplicity and ease of use. LangChain also adheres to encapsulation but at a higher level compared to HuggingFace. LangChain is a more complex framework thus its encapsulation is more inherently complex. LangChain still requires a good amount of knowledge about the underlying systems to use its framework to the absolute best.

Abstraction In (Kramer 2007) describes Abstraction and its importance as the "significance of taking away complication". Hugging Face provides a strong abstraction for interacting with pre-trained models. Its pipeline() method simplifies many different LLM tasks like text generation and others into a single high-level method call. This abstracts away all details related to model loading, tokenization, and inference pipelines. LangChain abstracts its complicated multi-step workflows, tools, and memory management. By allowing developers to define high-level behaviors, like an agent retrieving information from documents or a database, and answering questions, while LangChain manages the underlying complex logic. This abstraction is extremely important when building large scale complex AI systems or integrating AI into already complex systems. LangChain adheres better to abstraction as it handles the complexities of tools, memory management, state, and the orchestration of complex agentic workflows. Hugging Face is great at abstracting individual AI tasks, but its abstractions are more interactive

in small form with LLMs and less for managing AI software.

LlamaIndex vs ElasticSearch

Inheritance LlamaIndex uses inheritance (Johnson and Foote 1991) to allow developers to build custom indexing and querying software for their data. Core classes like BaseIndex and BaseRetriever provide the base functionality for managing data, storage of data, and retrieval of data, which the framework allows developers to extend, to implement custom logic or a unique flow to optimize for their specific software case. For example, developers can inherit from BaseIndex to create an index that combines both tree-based and a vector-based retrieval. Elasticsearch uses inheritance less, compared to LlamaIndex, in its design but applies similar concepts through their plugins and extensions. Developers can extend the base capabilities of Elasticsearch by adding custom analyzers, tokenizers, or plugins that integrate directly into the Elasticsearch engine. While this does provide flexibility, it requires a lot of knowledge of Elasticsearch's internal software and code. LlamaIndex more closely adheres to inheritance by providing better and more extendable code that is easily accessible to developers. On the other hand, Elasticsearch relies on plugins and custom software, to extend its capabilities for developers, making it harder to use.

Polymorphism LlamaIndex uses polymorphism (Pierce 2002) to provide consistent interfaces for querying different and custom index types. TreeIndex and VectorIndex have a shared interface for data retrieval/access, which allows developers to interact with multiple different index types using the same exact methods, such as query() or retrieve(). Elasticsearch also uses polymorphism in its query language, called Domain Specific Language (DSL). Different query types like match, term, and bool queries, can be used within the same search pipeline. This allows developers to dynamically create complex queries without modifying their overall search logic. Both frameworks adhere well to polymorphism, but Elasticsearch uses polymorphism across most of the framework by supporting polymorphism across its query system and extensible components. LlamaIndex, while adhering well, focuses its polymorphism more on managing multiple index types.

Encapsulation LlamaIndex encapsulates (Pierce 2002) the process of indexing and querying data through methods such as, adddocuments() and query(). Developers can use these methods without understanding the underlying implementation of a function, method, or tree construction and vector similarity calculations, which LlamaIndex uses for more complex retrievals. Elasticsearch has encapsulation through its REST API and libraries, which allows users to perform search and indexing operations without needing to understand the underlying index or clustering logic of code. Except for the fact that Elasticsearch's configuration design requires developers to be familiar with a wide range of settings and configurations, which can take away from its adherence to the full encapsulation concept. LlamaIndex adheres more strongly to encapsulation by providing a simpler and more focused interface for indexing and querying, where Elasticsearch, while powerful, exposes more of its complexity to the user.

Abstraction LlamaIndex excels in abstraction (Kramer 2007) by giving developers a simplified interface for integrating unstructured data with LLMs. Functions to create a tree-based index or do semantic queries are abstracted into a few different function calls, allowing developers to focus on high-level workflow logic rather than implementation details. Elasticsearch also provides a strong abstraction for search and analytics operations. Its DSL allows users to build complex search logic without needing to interact directly with the underlying complex structures of the search logic. Except, the abstraction is less focused on AI workflows, requiring developers to code/integrate more logic for AI applications. LlamaIndex adheres more closely to abstraction for AI-specific tasks, as it is explicitly designed to simplify workflows involving LLMs and unstructured data. Elasticsearch offers a robust abstraction for general-purpose search but requires additional effort to adapt to AI-driven use cases. Both for their respective tasks and framework purpose align well with Abstraction.

Discussion of Results

LangChain shows a great utilization of inheritance and polymorphism by giving developers the ability to create custom AI software easily and repetitively. It uses encapsulation and abstraction to allow AI to integrate smoothly into software today, showing the creator's philosophies behind the framework. Encapsulation and abstraction on functionality like tools, and memory, also giveaway for more acceptance of the framework. However, the framework has a big learning curve and a long list of bugs (for interoperability between cloud services) which will prevent developers from easily using it.

Hugging Face is made simple, for developers to have easy access to AI models. Hugging Face is focused on simplifying interactions with pre-trained and custom AI models. Its use of encapsulation and abstraction makes it extremely user-friendly and perfect for individual AI tasks, but it fails in flexibility and good customization. The framework's limited application of inheritance and polymorphism is a massive trade-off for its simplicity and ease of use.

LlamaIndex uses encapsulation and abstraction very well, in focusing on unstructured data and integrating it with AI software. Its implementation of inheritance on indexes allows developers to extend the indexing functionality to apply to specific use cases and software logic, while its use of polymorphism helps support use among various index types. Elasticsearch is more general, focused on using polymorphism and encapsulation in its query logic. However, Elasticsearch requires more background knowledge and software experience, making it less usable for AI-focused developers.

Overall, the analysis shows a difference in philosophy between simplicity and customization. Frameworks like Hugging Face and Elasticsearch prioritize user-friendliness, while LangChain and LlamaIndex offer more customization for complex AI workflows. These differences emphasize the importance of applying OOP concepts to frameworks and of

knowing these concepts when choosing frameworks to build software products. OOP concepts, like the four described, in frameworks like LangChain and LlamaIndex not only help simplify AI integration into software but also provide the chance to integrate assurance practices, or testing/validation into AI software development. In (Batarseh, Freeman, and Huang 2021), they highlight the importance of using assurance methods in the implementation of AI in software to address issues like data bias and transparency and to promote trust in AI software.

Conclusion

Our paper analyzed the application of object-oriented programming (OOP) principles in four different frameworks for AI development: LangChain, Hugging Face, LlamaIndex, and Elasticsearch. Each framework shows various levels of application of OOP principles, this in turn shows the different design choices and philosophies of each framework. LangChain and LlamaIndex focus on workflow and data logic abstractions, making them the ideal choice AI applications. Meanwhile, Hugging Face and Elasticsearch prioritize simplicity and ease of use, focusing on specific tasks like model inference and search operations.

Our results and findings show the importance of OOP concepts in AI-focused software. As AI continues to grow and AI products scale more and more, these frameworks demonstrate the reason why good software engineering practices should be applied to AI frameworks to meet the demands of modern software. Future research should explore additional frameworks, performance metrics, and real-world applications to further add to our understanding of AI frameworks and to build better ones.

References

Amershi, S.; Begel, A.; Bird, C.; DeLine, R.; Gall, H.; Kamar, E.; Nagappan, N.; Nushi, B.; and Zimmermann, T. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291–300.

Batarseh, F. A.; Freeman, L.; and Huang, C.-H. 2021. A survey on artificial intelligence assurance. *Journal of Big Data*, 8(1).

Cardelli, L.; and Wegner, P. 1985. On Understanding. Accessed: 2024-10-25.

Johnson, R. E.; and Foote, B. 1991. Designing Reusable Classes. Cse.mit.edu.

Kramer, J. 2007. Is abstraction the key to computing? *Commun. ACM*, 50: 36–42.

Langchain AI. 2023. LangGraph Examples. Accessed: 2024-10-25.

Oracle. 2023. Abstract Classes and Methods. Accessed: 2024-10-25.

Pierce, B. C. 2002. *Types and Programming Languages*. MIT Press.

Run Llama. 2023. Llama Index. Accessed: 2024-10-25.