This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

Towards Responsible AI in the Era of Generative AI: A Reference Architecture for Designing Foundation Model based Systems

Qinghua Lu*, Liming Zhu, Xiwei Xu, Zhenchang Xing, Jon Whittle
Data61, CSIRO, Australia

Abstract—The release of ChatGPT has drawn huge interests on foundations models. There is a broad consensus that foundations models will be the fundamental building blocks for future AI systems. However, there is a lack of systematic guidance on the architecture design. Particularly, the the rapidly growing capabilities of foundations models can eventually absorb other components of AI systems, posing challenges of moving boundary and interface evolution in architecture design. Furthermore, incorporating foundations models into AI systems raises significant concerns about responsible AI due to their opaque nature and rapidly advancing intelligence. To address these challenges, the paper first presents an architecture evolution of AI systems in the era of foundation models, transitioning from "foundation-model-as-a-connector" to "foundation-model-as-a-monolithic architecture". The paper then identifies key design decisions and proposes a pattern-oriented reference architecture for designing responsible foundation-model-based systems. The patterns can enable the potential of foundation models while minimising associated risks.

Index Terms—Responsible AI, ethical AI, AI safety, architecture, pattern, foundation model, large language model, LLM, ChatGPT, AGI, GenAI.

1 Introduction

THE release of ChatGPT, Bard, and other large language model (LLM)-based chatbots has drawn huge attention on foundations models (FMs) worldwide. FMs are massive artificial intelligence (AI) models that are pre-trained on vast amounts of broad data and can be adapted to perform a wide variety of tasks [1]. With numerous projects already underway to explore their potential, it is widely predicted that FMs will serve as the fundamental building blocks for most future AI and artificial generative intelligence (AGI) systems.

Many reusable solutions have been proposed to tackle various challenges in designing FM-based systems. However, there is a lack of systematic guidance on the architecture design of FM-based systems. The impact of integrating FMs into software architecture are not fully studied yet. Additionally, the FM's growing capabilities can eventually absorb the other components of AI systems, introducing the moving boundary and interface evolution challenges in architecture design.

On the other hand, there are unique challenges on building responsible AI into the architecture of FM-based systems. First, accountability becomes more complex due to the involvement of multiple stakeholders. The accountability for decisions made by FM-based systems may be shared among the system owner, the FM provider, and various providers of external tools (such as ChatGPT plugins [1]). Second, enabling accountability necessitates the underlying supporting mechanisms for traceability. It is essential to record the inputs and outputs of FMs, systems, and external tools, services, and systems. Third, trustworthiness is a substantial obstacle in designing the FM-based systems, e.g., whether the user prompts, the FM outputs and the intermediate steps align with human goals and fulfill trustworthiness criteria. Fourth, the potential AI/AGI misuse poses a considerable challenge, which requires continuous risk assessment to ensure the instructions for the FM-based systems set by humans are trustworthy and responsible.

There is an urgent need for concrete system-level guidance to design responsible FM-based systems. In this paper, we first discuss the potential architecture evolution of AI systems in the era of FMs and highlight the key quality attributes necessary for the design of responsible FM-based systems. We then identify the major decision making points when designing FM-based systems. Finally, we propose a pattern-oriented reference architecture, which provides a responsible-AI-by-design template architectural solution for designing FM-based systems and considers the evolution of architecture to ensure adaptability over time.

2 Architecture evolution of AI systems

FMs are designed to provide a wide range of comprehensive capabilities that can be applied to various tasks, rather than being limited to specific functionalities [1]. One key challenge that the architecture design of AI systems faces with FMs is that FMs can eventually absorb the external components such as system functionalities and software engineering tools. While these components may exist for a while, they can become short-lived and eventually get integrated into the FM, resulting a single, monolithic blob at the center of the architecture. As illustrated in Fig.1, the architecture evolution of AI systems can be divided into three stages:

* *Contact email: qinghua.lu@data61.csiro.au*
1. https://openai.com/blog/chatgpt-plugins

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

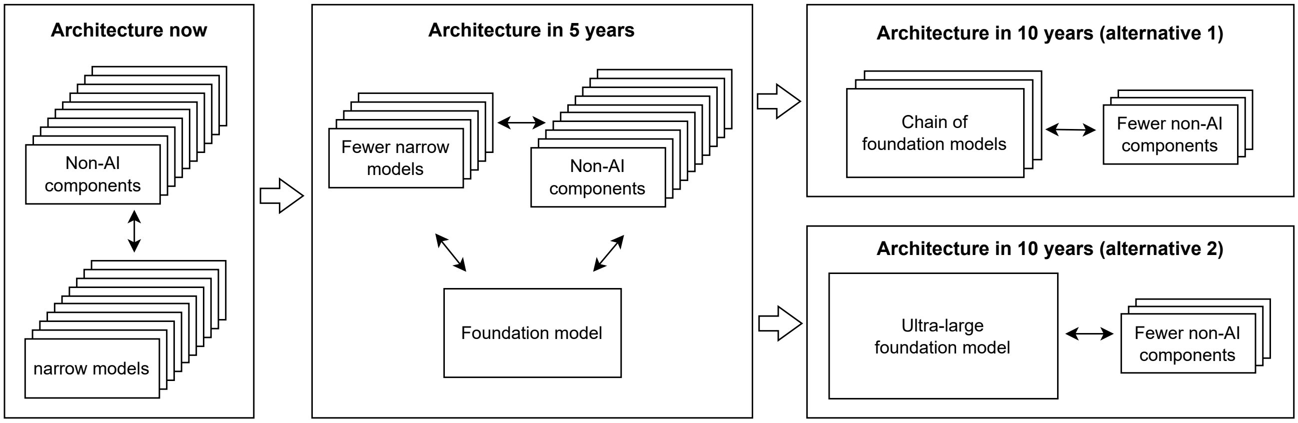

Fig. 1. Architecture evolution: from "foundation-model-as-a-connector" to "foundation-model-as-a-monolithic-architecture".

- ***Architecture now*: many AI models + many non-AI components**. The current architecture of AI systems usually comprises AI models (i.e., AI components) and non-AI components. These AI models and non-AI components co-exist within the architecture of the AI systems and interact with each other to enable the systems to function properly. The AI models are responsible for processing data and making inference, while the non-AI components are responsible for tasks such as user interface, data storage, interaction with other systems.
- *Architecture in 5 years* - FM-as-a-connector: 1 FM + fewer narrow AI models + many non-AI components. In this architecture, the FM acts as a connector between external components, i.e., narrow AI models or non-AI components. The FM can provide four types of connector services [2]:
  - **FM-as-a-communication-connector**: enabling the transfers of data between software components, e.g., extracting the task description from the user prompt and transferring to other components for further processing.
  - **FM-as-a-coordination-connector**: planing a workflow and coordinating task execution through various software components.
  - **FM-as-a-conversion-connector**: functioning as an interface adapter for software components that use different data formats to communicate with each other, e.g. parse the task into machine-readable template for executing by an AI model.
  - **FM-as-a-facilitation-connector**: facilitating the interactions between components, e.g., creating logs or deciding the invocation of local models.

  In such architecture, FMs still need to interface with narrow AI models and non-AI components to tackle complex tasks, such as HuggingGPT [3]. However, as the capabilities of FMs continue to expand rapidly, it is expected that many of those components will be eventually absorbed into the FMs and ultimately disappear.
- *Architecture in 10 years*:
  - **Alternative 1: chain of FMs + fewer AI and non-AI components**. There is a chance that most of the AI and non-AI components could be absorbed into the FMs. Thus, one alternative of the architecture in 10 years is a modularised architecture, such as Socratic Models [4]. This architecture relies on a few FMs that are chained together and a limited number of AI and non-AI components to perform tasks (e.g., through language-based interactions) without requiring additional training or fine-tuning. The inference for a task-specific output is jointly performed by multimodal interactions between the independent FMs, such as text-to-text models, text-to-visual models and text-to-audio models. Those FMs can be connected via APIs with external AI or non-AI components that offer additional capabilities or access to databases, such as robotic systems or web search engines. By multimodal interaction between independent models, the architecture can effectively leverage the capabilities of different FMs and external AI and non-AI components. In this architecture, prompt engineering is important for guiding the FM to produce high-quality responses. Various prompt patterns can be applied, including few-shot prompting, self-consistency, chain-of-thought, retrieval augmented generation, etc [2].
  - **Alternative 2 - FM as a standalone component: 1 ultra-large FM**. Another potential type of future architecture is a monolithic architecture, which only contains a single big FM capable of performing a variety of tasks by incorporating different types of sensor data for cross-training. An example of this type of architecture is PaLM-E [5], which is used for performing language, visual-language, and reasoning tasks. In this type of architecture, no external components are required, including prompt components. In this architecture, the non-AI components may include context engineering components (such as multimodal context injection), prompt engineering components (such as prompt generator), and responsible AI components (such as continuous risk assessment).

As FMs continuously and rapidly evolve with growing capabilities, many of the existing software components will be likely to become obsolete since their functions will be provided by new versions of FMs. For example, Tesla AI is working on an end-to-end model which learns all steps

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

from the initial input phase and the output result phase [2]. This means that context data goes in and driving decisions come out, without a single line of code implemented in the process of autonomous driving. Thus, adaptability and modifiability are the two key concerned software quality attributes. Adaptability refers to a software system's ability to adapt to run-time changes in its environment without requiring external intervention [6], such as changes in the data being processed. Modifiability is the ease with which a software system can be changed at static-time [6], such as adding new features, fixing bugs, or changing the underlying infrastructure. Both adaptability and modifiability are important qualities attributes for an evolving architecture, as they can significantly impact the long-term maintainability of a system. The patterns and tactics of conventional software systems could be applied to manage the issues of moving boundary and interface evolution in the FM-based systems.

3 Architectural Design decisions

There are some major architectural design decisions that developers need to consider when building FM-based systems.

3.1 Design decision 1: Different design options for using FMs

When designing the architecture, one of the most important decisions is choosing which type of FM to use. There are four types of FMs:

- **FM type 1**: pre-trained by an external organisation using large unlabeled general data (e.g., general text corpus).
- **FM type 2**: first pre-trained using large unlabeled general data, then pre-trained using large unlabeled domain specific data (e.g., public real-estate data). The training can be conducted by the same external organisation or two different external organisations.
- **FM type 3**: pre-trained by an external organisation using both large unlabeled general data and large unlabeled domain-specific data.
- **FM type 4**: sovereign FM which is trained from scratch within an organisation using large unlabeled/labeled general data and/or large unlabeled/labeled domain-specific data.

There are two design options for the use of FMs for FM type 1, 2, and 3: using FMs via in-context learning or fine-tuning FMs using labelled target data. Using FMs through in-context learning can save costs, as FMs are pre-trained by external organisations on vast amounts of data using numerous computational resources. However, these options may pose responsible AI issues (such as reliability and safety) as the pre-training is conducted towards generating generic outputs rather than being designed to a specific purpose. This could result in reliability, safety, ethical issues with the model's outputs. Furthermore, there could be data privacy issues. For example, the data used to train these models may be biased or include personal information. The big tech companies may use the data collected from these models for their own purposes, such as improving their products or developing new features, which may raise privacy concerns. Thus, additional responsible AI design patterns can be introduced at system-level, such as continuous risk assessment and guardrails.

Fine-tuning FMs can help improve reliability and privacy of the FMs by locally fine-tune the model with labeled target data (e.g., labeled domain-specific data). However, responsible AI issues may still exist as organisations cannot control the FM training process of the external organisations.

To have complete control over data and model training and ensure responsible AI, FM type 4 - sovereign FM is the best option. Also, some organisations may possess unique internal data and training a sovereign FM in-house from scratch can become their unique competitive advantage. However, it requires high investment in cost and resources, including data, computational, and human resources.

3.2 Design decision 2: Chain of FMs vs. an ultra-large FM

When considering FMs developed by external organisations, one important decision is whether to use a chain of models (such as Socratic Models [4]) or an ultra-large FM (such as PaLM-E [5]). The chain of FMs generates joint predictions and offers an modularised architecture that allows for easy switching to other FMs with specific capabilities, e.g., switching to a more powerful visual language model to improve performance. This option may improve maintainability, but it may come with an additional cost to understand the capabilities and limitations of different foundations models. On the other hand, using ultra-large FM may achieve better performance via cross-training on numerous multi-modal data [5]. However, this option may come at the cost of reduced maintainability. There may be a risk for vendor lock-in, as there may be few providers in the market with similar capabilities. It is challenging to determine which option is better, and experimentation is necessary to evaluate each option's effectiveness for a specific context.

3.3 Design decision 3: Responsibilities of external components

Responsibility is a concept in a software context that comes from object-oriented design. A responsibility can be an action, a piece of knowledge to be maintained, or a decision to be carried out by a software component.

FMs can gradually absorb external components by taking on their responsibilities over time. This can create a moving boundary issue where the responsibility of a software component shifts from an external component to the FM. To address this issue, one key design decision is to determine the responsibilities of software components. The responsibility can be split into a bunch of smaller responsibilities that are placed in distinct components. Changes can be isolated to specific components, making it easier to manage the external component that could be absorbed by the FM over time. As FMs are built around capabilities [1], it may be worth breaking down a large component along capability lines. This allows the developers to choose which

2. https://twitter.com/Tesla_AI/status/1730761835694153790

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

FM's capabilities to use, e.g., use a good enough one or an emerging new one.

Breaking down responsibilities into smaller components can improve adaptability and modifiability, ensuring long-term maintainability. However, it can also introduce additional communication overhead between smaller components, as each component may need to interact with other components to accomplish tasks. Additionally, it can make the system more complex and difficult to understand how the components work together, potentially reducing maintainability.

3.4 Design decision 4: Automatic response vs. verifier

When designing FM-based systems, an important consideration is how to ensure the systems' responses are accurate and responsible. One option is to rely solely on the FM to generate responses to user queries. While this option can be efficient and cost-effective, it may result in inaccurate or irresponsible responses that can affect user trust or cause harm.

To address this issue, before responding to the user, a verifier can be adopted to verify whether the FM outputs meet the specified requirements such as topic requirements or trustworthiness requirements [7]. The verifier could be a human verifier or an AI verifier (FM-based or non-FM-based) [2]. However, verifying the output of FMs directly can be challenging. It is necessary to take a conversational step-by-step approach to conduct verification. The choices between automatic response and verifier depends on the system's priorities and the consequences of inaccurate or irresponsible responses. For systems where accuracy and trustworthiness are critical, a verifier approach may be the better option, even if it comes at a higher cost. However, for systems where efficiency and cost-effectiveness are more important, an automatic response approach may be more suitable, with periodic checks.

3.5 Design decision 5: passive interaction vs. proactive interaction

Interaction in FM-based systems involves context engineering and prompt engineering. Context engineering aims to gather and structure the context in which the FM-based systems operate to understand the users' goals [8], while prompt engineering creates prompts that act as guiding instructions, enabling the FM-based systems to effectively complete human's tasks and goals.

There are two distinct interaction patterns: passive interaction and proactive interaction. Passive interaction interpretes the user's intentions as described through text prompts submitted via the dialogue interface. In contrast, proactive interaction anticipates the user's intentions and makes proactive suggestions by understanding multimodal context data, such as screen recording [9], mouse clicks [3], typing, eye tracking, gestures [10], document annotations and notes. Compared to passive interaction, proactive interaction introduces a deeper level of autonomy.

3. https://github.com/ddupont808/GPT-4V-Act

3.6 Design decision 6: Single agent vs. agent team

The tasks that need to be performed by an FM can vary in complexity and scope. For simple tasks or goals, such as answering frequently asked questions, a single agent may be sufficient. However, for more complex tasks, it may be necessary to use a team of agents to ensure the performance. For example, one agent can be responsible for question-answering interaction, while another agent can be designed for analysing context data. However, using multiple agents can also introduce communication costs between agents and increase the level of design complexity.

3.7 Design decision 7: Think aloud vs. think silently

There are two options to consider when it comes to the explaining the decision-making process: think aloud and think silently. The think aloud design pattern can be used disclose the decision-making process, such as the intermediate steps such as prompt pattern implementation and verification/validation. This design can help build human trust in the system, but it may sacrifice data privacy. For example, some system providers may view the prompt design and verification/validation as business sensitive data and intelligence property. In such case, they may need to carefully consider which parts of the intermediate process they are willing to share with the users.

4 Reference Architecture

Fig. 2 illustrates a pattern-oriented reference architecture for designing responsible and adaptable FM-based systems. The architecture comprises three layers: the system layer, which includes the components of the deployed AI system, the operation layer, which provides responsible AI tooling functions to the AI system, and the supply chain layer, which generates the software components that compose the AI system. The grey-coloured boxes and cylinders are the components where the design patterns are applied. An empirically-grounded design methodology has been adopted for designing the reference architecture [11]. The type of our reference architecture is an industry-crosscutting, classical, facilitation reference architecture. Our design strategy is a combination of research-driven and practice-driven, as the design of this reference architecture is founded mainly on the findings of literature review [12] and our project experience[4].

4.1 System layer

The system layer comprises the components of the deployed FM-based systems. Interaction components comprises two sub-components: **multimodal context engineering** and **auto-prompt generation**. Multimodal context engineering is designed to collect and structure the context in which the system operates to understand the user's goals or tasks. Instead of analysing user's goals or tasks described through text prompts sent by the user via the dialogue interface, the system can proactively anticipate the user's goals by analysing multimodal context information, including screen recording [9], mouse clicks, typing, eye tracking, gestures [10], document annotations and

4. https://research.csiro.au/ai4m/operationalising-responsible-ai/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

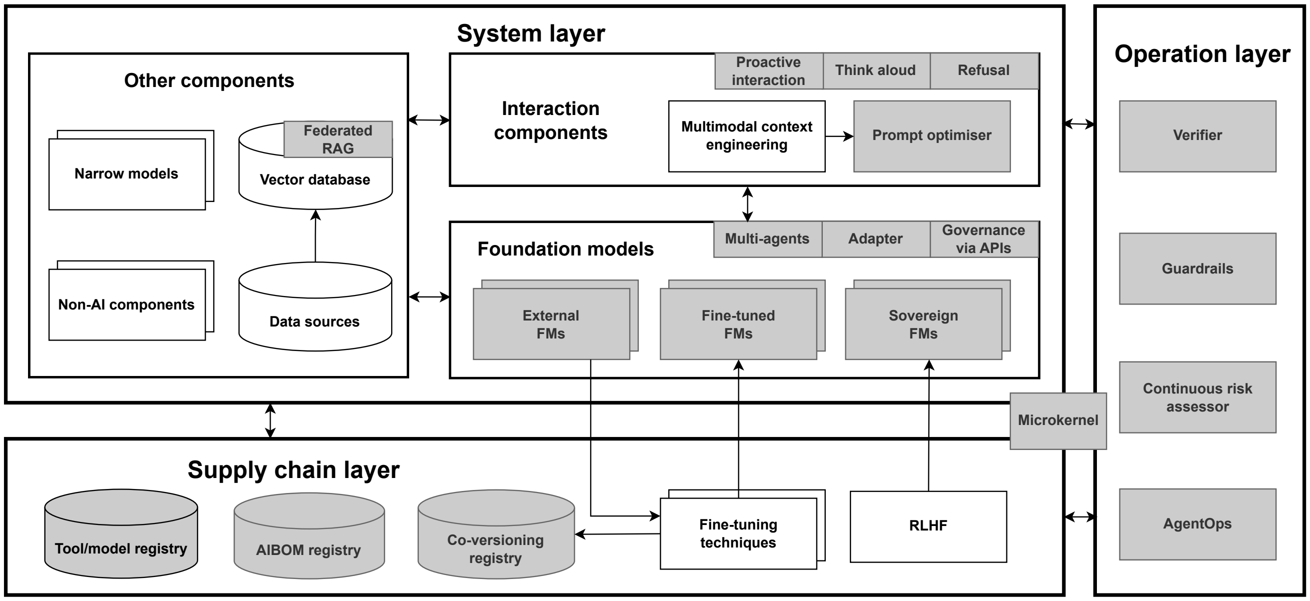

Fig. 2. A pattern-oriented reference architecture for designing responsible FM-based systems.

notes. **Auto-prompt generation** automatically produces the prompts with constraints and specifications, defining the desired input or output content and format in alignment with the the ultimate goal. A prompt template is often used as a factory that creates prompt instances from the template. The template provides a structured way to standardise the queries, which can improve the response accuracy and interoperability with external systems.**Think aloud** is a pattern for explainability, which can describe the system's capabilities, limitations, the rationale behind its intermediate or final outputs, and ethical or legal implications. This design pattern can help build human trust in the system, but it may sacrifice data privacy. **Prompt refusal** filters inappropriate or harmful tasks, e.g., refusing to generate responses that contain violence promotion or hate speech.

There are several patterns for using FMs, including the **external FMs**, **fine-tuned FMs**, **sovereign FMs**, **chain of FMs** and **ultra-large FMs**. Due to the rapidly growing capabilities of FMs, there are issues with the moving boundary and interface evolution in architecture design. Most of the components in the system layer, operation layer, and supply chain layer will eventually be absorbed by FMs. In some cases, absorption is not done component by component, but rather by splitting a component in the middle. To address these issues, two classic patterns can be applied to ensure adaptability and modifiability: **microkernel** pattern [13] and **adapter** pattern (also known as **wrapper**). **Microkernel** pattern can help place smaller responsibilities in distinct component so that changes can be isolated to specific components, which could be absorbed by the FM overtime. When a component is absorbed by an FM, the component's original connector used to communicate with other components may need to be converted into a certain format of interface (such as a text interface for LLMs) through the use of an **adapter** pattern.

With autonomous agents, users only need to provide a high-level goal, rather than providing explicit step-by-step instructions. These agents derive their autonomy from the capabilities of FMs, enabling them to break down the given goal into a set of manageable tasks and orchestrate task execution to fulfill the goal. The agents can be categorised into two types of roles: agent-as-a-coordinator and agent-as-a-worker. Agents in the coordinator role primarily formulate high-level strategies and orchestrate the execution of tasks by delegating task execution responsibilities to an **agent team** in the worker role. These agents in the worker role need to generate strategies and execute specific tasks in line with those strategies. To complete these tasks, agents in the worker role may need to cooperate or compete with other agents, or call external tools or non-agent AI/non-AI systems.

To prevent harmful dual-use of AI systems, developers should impose restrictions on their usage and prevent users from getting round of restrictions through unauthorised reverse engineering or modification of the system design. One way to do this is by implementing **governance via APIs** pattern, which involves providing AI services on cloud platforms and managing interactions through API controls (such as GPT4), rather than allowing AI systems to run locally with unrestricted access.

To improving the response accuracy of FMs, retrieval augmented generation (RAG) can be adopted by using a **vector database** (such as Pinecone [5]) for storing the domain data from various data sources as vector embeddings. These embeddings can be used to perform similarity searches and enable the retrieval of data that are related to specific tasks. When the tasks involves cross-organisational data analytics, **federated RAG** can be applied by adapting federated learning. Each organisation deploys an FM and has its own RAG in which the data is confidential to other organisations. The data in the local RAGs can be aggregated and further processed by a central FM.

5. https://www.pinecone.io

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

4.2 Operation layer

The operation layer includes components responsible for monitoring and managing the responsible AI related qualities in deployed FM-based systems. A **verifier**, whether human or AI (FMs or non-FM models) [6], can be introduced to check whether the final or intermediate outputs meet the specified requirements such as topic requirements or trustworthiness requirements.

Guardrails can be added at three stages: 1) Preprocessing: After receiving the user prompts, guardrails can be enforced after verifying whether the prompts can comply with responsible AI requirements through the verifier. For example, some users prompts might not be relevant to the pre-configured scope and should be rejected [7]. Whitelists and blacklists can be established to identify actions that are permitted and prohibited respectively. Additionally, some users prompts may contain personal identifiable information (PII) and need to be removed by employing data de-identification and anonymisation before being sent to the FMs. 2) Intermediate process: During task execution, if the output at each intermediate step does not fullfil the users' specific requirements including trustworthiness requirements, the respective types of guardrails should be invoked. 3) Postprocessing: When FM returns results, guardrails are necessary to ensure that the outputs meet the requirements including responsible AI requirements, structure requirements, etc.

A **continuous risk assessor** continuously monitors and assesses AI risk metrics [8] to prevent the misuse of the agent and to ensure the trustworthiness of the agent. For example, when a user submits a prompt through the dialogue interface, the continuous risk assessor can assess the potential risks of the intended goals and may modify or reject the prompt based on the risk assessment results. A **black box recorder** records the runtime data, which can be then shared with relevant stakeholders to enable transparency and accountability [14]. The recorded data includes the input, output, and intermediate data for each layer or component within the architecture, such as the input and output of the FMs or other components. The data recorded by the black box recorder can be helpful in determining accountability, including which stakeholders share the accountability and to what extent. All these data need to be kept as evidence with the timestamp and location data. Design decisions need to be made on what data should be recorded and where the data should be stored (e.g., using a blockchain-based immutable log or a cloud-based data storage). The **standardised reporter** pattern can be used to inform stakeholders (such as regulators and users) about the development process and product design of AI systems, as well as the runtime data collected by the black box recorder.

4.3 Supply chain layer

The supply chain layer includes all components involved in developing and procuring both AI components (including FMs) and non-AI components. FM providers can use reinforcement learning from human feedback (RLHF) to fine-tune the FM's behaviour and produce more accurate and responsible responses. RLHF allows humans to provide feedback on the quality of the responses and uses this feedback to adjust the model's parameters. The FM is then trained to maximise the reward it receives from human feedback, which can improve its accuracy and responsible AI related qualities over time. Parameter-efficient fine-tuning techniques, e.g., LoRA [9], can reduce the number of training parameters by 10,000 times and decreasing GPU usage by threefold. All components procured from third parties can be associated with a bill of materials (BOM) that records their supply chain details, which can include responsible AI (RAI) metrics or verifiable RAI credentials. This procurement information can be maintained in an **AIBOM registry**. FMs can refuse to call the third party components or systems that fail to provide registered AIBOM information. To ensure auditability, the **co-versioning registry** pattern can be applied to co-version the AI artifacts, such as external/sovereign FMs, fine-tuned FMs, and training/testing datasets.

5 Evaluation

In this section, we evaluate the completeness and utility of the proposed reference architecture by mapping it to the architecture of a real-world FM-based system, responsible AI (RAI) chatbot. This chatbot enables scientists to understand and assess potential AI risks in their AI projects[10]. The current version is built on GPT-4, an **external FM** provided by OpenAI. Users interact with the chatbot by describing their AI systems and asking questions through a web interface. Questions not relevant to RAI are automatically rejected through **prompt refusal**.

To improve accuracy in responding to user queries, the RAI chatbot employs **RAG** by integrating LlamaIndex [11], which connects GPT-4 with local **data sources**. The local data sources include a manually labelled AI incident database, the Responsible AI Question Bank [15], and the Responsible AI Pattern Catalogue [12].

The chatbot incorporates the **think aloud** pattern, making the intermediate processes transparent to users. **Agent team** is used by creating multiple instances to perform different tasks. For example, one agent is responsible for question-answering interactions, while another summarises the documents shared by users. All conversation histories are captured by a **black box recorder**. The **verifier** pattern is implemented, allowing a human expert to review and edit the answers generated by GPT-4. The project team is currently building an AIBOM registry for the RAI chatbot and discussing the potential of fine-tuning an FM to further increase response accuracy.

It can be concluded that our reference architecture is complete and usable as the architecture of RAI chatbot is successfully mapped to the proposed reference architecture. We have observed that the fundamental layers in an FM-based system architecture include the system layer, operation layer and supply chain layer. The key components in

6. https://openai.com/blog/our-approach-to-alignment-research
7. https://github.com/NVIDIA/NeMo-Guardrails
8. https://oecd.ai/en/catalogue/metrics
9. https://github.com/microsoft/LoRA
10. https://research.csiro.au/ai4m/operationalising-responsible-ai/
11. https://www.llamaindex.ai

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Software. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MS.2024.3406333

the system layer include FMs, interaction components, data sources.

6 Conclusion

This paper presents pattern-oriented responsible-AI-by-design reference architecture to address the challenges of responsible AI and architecture evolution in FM-based systems. We first discuss the architecture evolution and identify two important software qualities for building FM-based AI systems: adaptability and modifiability. Then, we summarise seven key design decisions in architecture design and discuss the trade-offs between responsible AI related software qualities. Finally, we present a pattern-oriented reference architecture to provide a concrete guidance for developers to design responsible and adaptable FM-based systems. In the future, we will build a pattern catalogue for building FM-based systems.

References

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] Q. Lu, L. Zhu, X. Xu, Y. Liu, Z. Xing, and J. Whittle, "A taxonomy of foundation model based systems through the lens of software architecture," *CAIN'24*, 2024.

[3] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 38 154–38 180. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/77c33e6a367922d003ff102ffb92b658-Paper-Conference.pdf

[4] A. Zeng, M. Attarian, B. Ichter, K. M. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. S. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence, "Socratic models: Composing zero-shot multimodal reasoning with language," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=G2Q2Mh3avow

[5] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "Palm-e: an embodied multimodal language model," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[6] ISO/IEC, "Iso/iec 25010:2011 - systems and software engineering. systems and software quality requirements and evaluation (square). system and software quality models," 2011.

[7] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, G. Du, S. Shi, H. Mao, X. Zeng, and R. Zhao, "Tptu: Task planning and tool usage of large language model-based ai agents," *arXiv preprint arXiv:2308.03427*, 2023.

[8] Y. Xia, M. Shenoy, N. Jazdi, and M. Weyrich, "Towards autonomous system: flexible modular production system enhanced with large language model agents," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–8.

[9] D. Zhao, Z. Xing, X. Xia, D. Ye, X. Xu, and L. Zhu, "Seehow: Workflow extraction from programming screencasts through action-aware video analytics," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 1946–1957.

[10] X. Zeng, X. Wang, T. Zhang, C. Yu, S. Zhao, and Y. Chen, "Gesturegpt: Zero-shot interactive gesture understanding and grounding with large language model agents," *arXiv preprint arXiv:2310.12821*, 2023.

[11] M. Galster and P. Avgeriou, "Empirically-grounded reference architectures: a proposal," in *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, 2011, pp. 153–158.

[12] Q. Lu, L. Zhu, X. Xu, J. Whittle, D. Zowghi, and A. Jacquet, "Responsible ai pattern catalogue: A collection of best practices for ai governance and engineering," *ACM Computing Surveys*, 2022.

[13] F. Bachmann, L. Bass, and R. Nord, "Modifiability tactics," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2007.

[14] G. Falco and J. E. Siegel, "A distributedblack box'audit trail design specification for connected and automated vehicle data and software assurance," *arXiv preprint arXiv:2002.02780*, 2020.

[15] S. U. Lee, H. Perera, B. Xia, Y. Liu, Q. Lu, L. Zhu, O. Salvado, and J. Whittle, "Qb4aira: A question bank for ai risk assessment," *arXiv preprint arXiv:2305.09300*, 2023.

Qinghua Lu is the team leader of Software Engineering for AI team and a principal research scientist at CSIRO's Data61, Australia. She formerly worked as a researcher at NICTA. She received her PhD from University of New South Wales in 2013. Her recent research interest includes software engineering for AI, responsible AI, software architecture, and blockchain. She has published 200+ academic papers in international journals and conferences. She is an IEEE senior member.

Liming Zhu is a Research Director at CSIRO's Data61 and a conjoint full professor at University of New South Wales. He is the chairperson of Standards Australia's blockchain and distributed ledger committee and on AI trustworthiness related committees. He received his Ph.D. from the University of New South Wales. His research program innovates in the areas of AI/ML platforms, responsible/ethical AI, software engineering, computational science, blockchain, regulation technology, quantum software, privacy and cybersecurity. He has published more than 200 academic papers on software architecture, secure systems and data/ML infrastructure, blockchain, governance and responsible AI.

Xiwei Xu is a principal research scientist and serves as the Group Leader for the Software Systems Research Group at CSIRO's Data61, Australia. She is a conjoint senior lecturer with UNSW. She received her PhD from University of New South Wales in 2011. Her recent research interest includes software engineering and architecture for AI, responsible AI, and blockchain. She has published 200+ academic papers in international journals and conferences.

Zhenchang Xing is a Senior Principal Research Scientist and the science leader of the SE4AI team at CSIRO's Data61. He is an honorary associate professor in the School of Computing at Australian National University. He received his PhD from the University of Alberta, Canada in 2008. His recent research interest includes software engineering, responsible AI, knowledge graph, and human-computer interaction.

Jon Whittle is Director of CSIRO's Data61, the digital and data science arm of Australia's national science agency. With around 1000 staff and affiliates, Data61 is one of the largest collections of R&D expertise in Artificial Intelligence and Data Science in the world. Prior to joining Data61, Jon was Dean of the Faculty of Information Technology at Monash University, the largest university in Australia. He received his PhD from the University of Edinburgh in 1999. Jon is a world-renowned expert in software engineering and human-computer interaction, with a particular interest in IT for social good, AI research in health sciences, and sustainable development.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/