A Taxonomy of Foundation Model based Systems through the Lens of Software Architecture

Qinghua Lu, Liming Zhu, Xiwei Xu, Yue Liu, Zhenchang Xing, Jon Whittle
Data61, CSIRO
Sydney, Australia

ABSTRACT

Large language model (LLM) based chatbots, such as ChatGPT, have attracted huge interest in foundation models. It is widely believed that foundation models will serve as the fundamental building blocks for future AI systems. However, the architecture design of foundation model based systems has not yet been systematically explored. There is limited understanding about the impact of introducing foundation models in software architecture. Therefore, in this paper, we propose a taxonomy of foundation model based systems, which classifies and compares the characteristics of foundation models and system design options. Our taxonomy comprises three categories: the pretraining and adaptation of foundation models, the architecture design of foundation model based systems, and responsible-AI-by-design. This taxonomy can serve as concrete guidance for designing foundation model based systems.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; • **Computer systems organization → Architectures**; • **Computing methodologies → Artificial intelligence**.

KEYWORDS

Software architecture, foundation model, responsible AI, large language model, LLM, taxonomy, generative AI

ACM Reference Format:
Qinghua Lu, Liming Zhu, Xiwei Xu, Yue Liu, Zhenchang Xing, Jon Whittle. 2024. A Taxonomy of Foundation Model based Systems through the Lens of Software Architecture. In *Conference on AI Engineering Software Engineering for AI (CAIN 2024), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3644815.3644956

1 INTRODUCTION

Foundation models (FM) are pretrained on massive amounts of data and can be adapted to perform a wide variety of tasks and significantly improve productivity [1]. With numerous projects already underway to explore their potential, it is widely predicted that FMs will serve as the fundamental building blocks for future AI systems. However, the design of FM-based systems is still in an early stage and has not yet been systematically explored. There is limited understanding of the impact of introducing FMs into software architecture. Many reusable solutions have been proposed to tackle various challenges in designing FM-based systems, which motivates the creation of a design taxonomy for FM-based systems. On the other hand, the black box nature and the rapid advancements in FMs have introduced significant responsible AI (RAI) challenges [9, 17]. Accountability becomes more complex with multiple stakeholders involved, including the system owner, FM provider, and external tool providers. Another major challenge is trustworthiness of both the final results and intermediate process. Additionally, the potential misuse of FM-based systems poses a considerable challenge.

Taxonomies have been used in the software architecture community to gain a deeper understanding of existing technologies [4, 11]. By categorising existing work into a compact framework, taxonomies enable architects to explore the conceptual design space and make rigorous comparisons and evaluations of different design options. Therefore, in this paper, we present a taxonomy that defines categories for classifying the key characteristics of FMs and system-level design options. The taxonomy is structured into three categories: the pretraining and adaptation of FMs, the system architecture design, and responsible-AI-by-design. The development of this taxonomy is grounded in an extensive review of literature, supplemented by insights obtained from our project experience. The taxonomy serves as guidance to help software architects make rational decisions in designing FM-based systems.

The remainder of the paper is organised as follows. Section 2 discusses related work. Section 3 presents the design taxonomy. Section 4 concludes the paper.

2 BACKGROUND AND RELATED WORK

OpenAI launched ChatGPT [1] in November 2022, which has gained over 100 million users within two months of its release. This triggered an arms race among big tech companies to develop FM-based generative AI (GenAI) products. Google responded with Bard [2], its own conversational GenAI products. By February 2023, Microsoft had integrated GPT-4 into its search engine, Bing. Another type of notable GenAI products is text-to-image generators, such as DALL-E [3]. Additionally, there are domain-specific FMs for finance [18], medicine [12], climate [13], etc. There are growing interests in integrating FMs into existing software systems or building new systems. To meet specific trustworthiness requirements many studies explore techniques for adapting FMs for downstream tasks, such as using domain-specific data [7] or in-context learning [19]. There are also many efforts on prompt engineering, including various prompt patterns [14] and prompt engineering tools used for developing FM-based systems. Langchain [4] is the most recognised

CAIN 2024, April 14–15, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0591-5/24/04.
https://doi.org/10.1145/3644815.3644956

[1]https://openai.com/blog/chatgpt
[2]https://bard.google.com
[3]https://openai.com/product/dall-e-2
[4]https://www.langchain.com

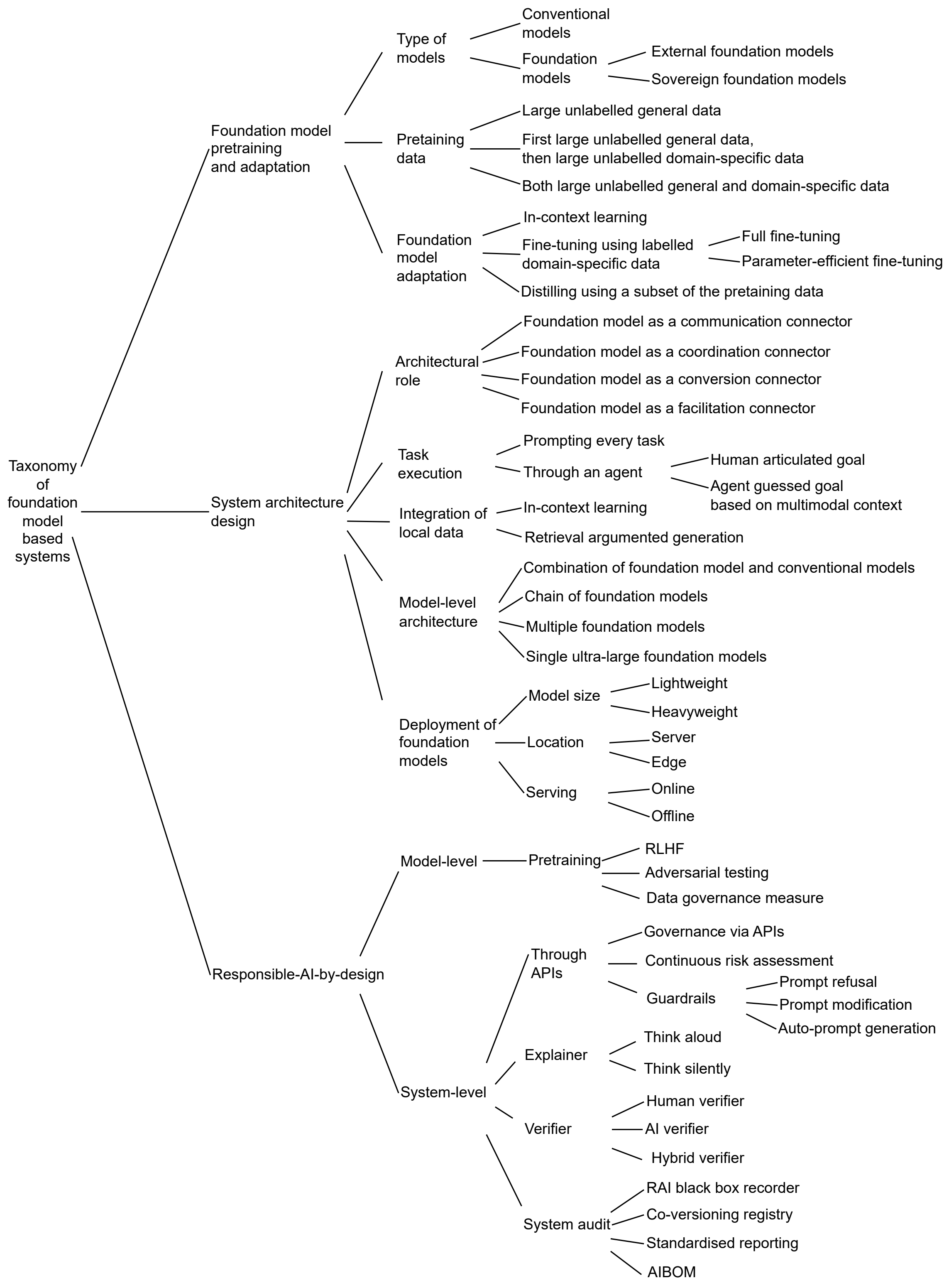
Taxonomy of foundation model based systems
Foundation model pretraining and adaptation
Type of models
Conventional models
Foundation models
External foundation models
Sovereign foundation models
Pretaining data
Large unlabelled general data
First large unlabelled general data, then large unlabelled domain-specific data
Both large unlabelled general and domain-specific data
Foundation model adaptation
In-context learning
Fine-tuning using labelled domain-specific data
Full fine-tuning
Parameter-efficient fine-tuning
Distilling using a subset of the pretaining data
System architecture design
Architectural role
Foundation model as a communication connector
Foundation model as a coordination connector
Foundation model as a conversion connector
Foundation model as a facilitation connector
Task execution
Prompting every task
Through an agent
Human articulated goal
Agent guessed goal based on multimodal context
Integration of local data
In-context learning
Retrieval argumented generation
Model-level architecture
Combination of foundation model and conventional models
Chain of foundation models
Multiple foundation models
Single ultra-large foundation models
Deployment of foundation models
Model size
Lightweight
Heavyweight
Location
Server
Edge
Serving
Online
Offline
Responsible-AI-by-design
Model-level
Pretraining
RLHF
Adversarial testing
Data governance measure
System-level
Through APIs
Governance via APIs
Continuous risk assessment
Guardrails
Prompt refusal
Prompt modification
Auto-prompt generation
Explainer
Think aloud
Think silently
Verifier
Human verifier
AI verifier
Hybrid verifier
System audit
RAI black box recorder
Co-versioning registry
Standardised reporting
AIBOM

Figure 1: Taxonomy of foundation model based systems.

prompt engineering tool. However, it has complex software architecture and difficulties in debugging and maintenance. This has led to the emergence of competitors, such as AutoChain[5], which primarily focus on assembling task instructions for FMs. However, there remains a lack of systematic research on software architecture design for FM-based systems. On the other hand, FMs pose significant AI risks, such as hallucinations, lack of source evidence, and potential for user manipulation. Various mechanisms, such as regulation, standards, tools, are typically used to address RAI. Many of these mechanisms have been summarised as patterns, including governance patterns, process patterns, and product patterns [10]. However, there are limited studies specifically designed for the development of responsible FM-based systems.

3 DESIGN TAXONOMY

In this section, we present a taxonomy for building FMs and integrating FMs into the design of software systems. We have performed a systematic literature review [8] to develop the taxonomy and formulated three research questions: 1) What are the pretraining and adaptation mechanisms for FMs? 2) What are the key architectural decisions in designing FM-based systems? 3) How can RAI be implemented in the design of FM-based systems? We selected the seminal paper [1] as our seed paper. We initially retrieved 810 studies that cite the seed paper. After applying inclusion/exclusion criteria and quality assessment, 72 primary studies were finally included for analysis. Additionally, we adapted some patterns from the RAI pattern catalogue [10] when building the Responsible-AI-by-design category. As illustrated in Fig. 1, our taxonomy is structured in three categories: the pretraining and adaptation of FMs, architectural design of FM-based systems, and responsible-AI-by-design. We discuss the characteristics of each design options and their impact on cost, accuracy, and RAI-related properties [6].

3.1 FM pretraining and adaptation

3.1.1 Using conventional AI models or FMs. **Conventional AI models** are trained from scratch on a specific task using a dataset collected and labelled specifically for that task. Most current AI systems comprises conventional AI models and non-AI components. Building conventional AI models for various tasks can be costly, requiring extensive data gathering and labelling. Limited amounts of training data and computation resources available within organisations can lead to low accuracy of the models. On the other hand, FMs are large AI models that are pretrained on massive amounts of broad data for general-purpose tasks such as language processing or image recognition, which can be then adapted to perform a wide variety of tasks through fine-tuning [1]. Using a **FM** can reduce human labour for developing the AI components as organisations either only need to pay for external FM usage fees based on the number of API requests made or share a single FM within the organisation. Fine-tuning can improve the accuracy of FMs for downstream tasks. In addition, a FM can interact with other small AI models for specific tasks. However, managing the FM pipeline can be more complex for organisations, whether the model is pretrained externally or internally. There can be more RAI-related issues with FM outputs, especially given the large-scale deployment of FMs. For example, the huge amounts of data used to train these models may be biased or include sensitive information.

3.1.2 Using external or sovereign FMs. Using an **external FM** can save human resources for model training, deployment and maintenance, as organisations only pay for API calls. Additionally, using an FM can potentially result in higher accuracy and generalisability to a wider range of tasks. However, there can be RAI-related issues due to limited visibility. For example, privacy concerns can arise as organisations are uncertain whether their customers' data will be reused without acknowledgement. On the other hand, some organisations may possess unique internal data and training a **sovereign FM** from scratch can become their unique competitive advantage. Also, the sovereign FM provides organisations with complete control over the model pipeline. The organisations can train the sovereign FM to meet specific needs and ensure RAI-related properties. The trained FM can be shared across different departments. However, this requires significant investments in terms of cost and resources, including data, computational power, and human resources. Moreover, it may take considerable expertise and time to train a FM that can achieve the desired accuracy and RAI-related qualities. For example, the UK recently announced initial start-up funding of £100 million to build its own sovereign FMs [7].

3.1.3 Pretraining data. A FM (pretraining type 1) is typically pretrained by an organisation (e.g., big tech companies) using **large, unlabelled, and general data**, e.g., general text corpus. This FM can then be further pretrained by the same organisation or a different organisation using **large, unlabelled, domain-specific data** (pretraining type 2), e.g., public real-estate data. A FM (pretraining type 3) can also be pretrained by an organisation using **both large, unlabelled, general data and large, unlabelled, domain-specific data simultaneously**. While pretraining type 2 & 3 require more cost and resources, they are capable of achieving better accuracy when performing domain-specific tasks and can improve RAI-related qualities (such as reliability). Pretraining type 3 may require less time and resources compared to pretraining type 2, since the pretraining process is carried out in a single step rather than sequentially. However, pretraining type 3 may require more complex infrastructure to manage different types of data, and may also require additional safeguards to ensure RAI.

3.1.4 FM adaptation. There can be different ways to adapt FMs for downstream tasks. **In-context learning** is to improve the accuracy of the FM by providing examples to perform a specific task. This allows the FM to perform a task more accurately without the need to tune any parameters. There might be still accuracy and RAI issues with the model outputs. This is due to limited input-output examples size that can be used through prompts, which can result in the model generating outputs that lack the necessary context and understanding required for accurate inference. **Fine-tuning the parameters of the FM using labelled domain-specific data** can be done in two ways. Full fine-tuning retrains all the parameters, which is less feasible and extremely expensive (such as GPT-3 with

[5]https://github.com/Forethought-Technologies/AutoChain
[6]https://www.industry.gov.au/publications/australias-artificial-intelligence-ethics-framework/australias-ai-ethics-principles
[7]https://www.gov.uk/government/publications/integrated-review-refresh-2023-responding-to-a-more-contested-and-volatile-world

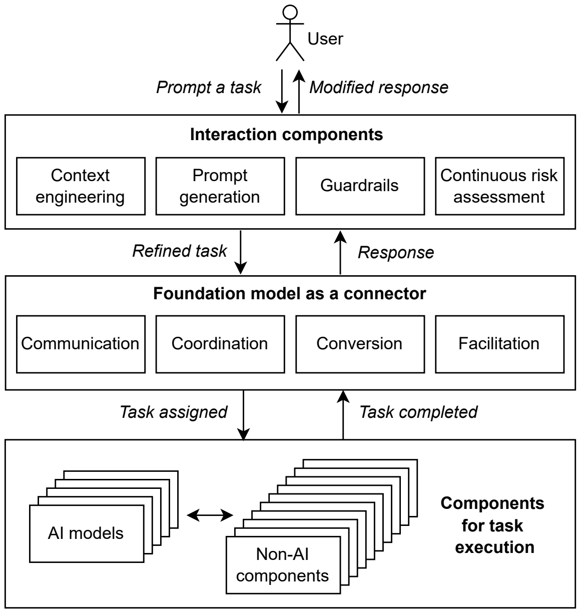

Figure 2: Foundation-model-as-a-connector.

175B parameters). Another way is to employ parameter-efficient fine-tuning techniques, e.g., LoRA [6] which reduces the number of training parameters by 10,000 times and decreasing GPU usage by threefold. If there is a need for improving the model's architecture and using pretraining model weights directly is not feasible, **distillation** becomes necessary. Distilling the FM is about training a smaller and more lightweight model to mimic the behavior of the larger and more complex FM using knowledge transfer techniques such as knowledge distillation, attention distillation, or parameter sharing. The distilled model is trained on a subset of the same data used to train the FM, but with a different loss function that encourages it to reproduce the output of FM. This can result in a more efficient and lightweight model that still maintains an acceptable level of performance. However, fine-tuning or distilling the FM highly couple with the open source model (e.g., LLaMA [16]).

3.2 Architecture design of FM-based systems

3.2.1 Architectural role: foundation-model-as-a-connector. In software architecture, software connectors are the fundamental building blocks of interactions between software components [11]. As shown in Fig. 2, FMs can be considered as a software connector. Once the task is prompted by a user, it can be further refined by interaction components for accuracy and RAI-related properties (see Section 3.3). The FM can play an architectural role as the following connector services. First, FMs can serve as a **communication connector** that enables the transfers of data between components. For example, an LLM can be employed to receive task text descriptions from users and extract meaning and intention from the text. The extracted task information can then be transferred to other components for further processing, such as sending to an AI model or a robotics system to perform a specific task [2]. Secondly, FMs work as a **coordination connector** to coordinate the computation of different software components. For example, an LLM can be used to plan a complex task or a workflow, which coordinates the planning, selection, and cooperation of multiple AI models through a text interface [5, 15]. The LLM first needs to decompose the task into a set of subtasks and decide dependencies and execution order for these subtasks. Then the LLM needs to obtain the model description and match the tasks and models. Thirdly, FMs can function as a **conversion connector**, i.e., an interface adapter, for software components that use different data formats to communicate with each other. For example, an LLM can analyze text task descriptions from users and parse it into machine-readable template for executing by an AI model [15]. Fourthly, FMs can be integrated as a **facilitation connector** to optimise interactions between components. For example, an LLM can be employed to maintain logs, determine whether to run some time-consuming models locally, manage resource dependencies between tasks in task execution stage, and summarise the task execution process and inference results.

3.2.2 Task execution with FMs. The dialogue interface receives user prompts for the FM to respond to, which can be inefficient for complex tasks or workflows, as users may need to prompt every single step. Prompt patterns (such as such as few-shot prompt, retrieval augmented generation) are often applied to guide the output of FMs. An autonomous **agent**, such as BabyAGI [8], can be used to improve efficiency. Users only need to prompt the overall goal, and the agent can break down the given goal into a set of tasks and use the other software components or external tools to achieve those tasks in an automatic way. Alternatively, the agent can go beyond the explicit user text prompt and anticipate the user's goals by understanding the user interface (UI) of tools and human interaction. This is facilitated through the analysis of multimodal context information, including screen recording, mouse clicks, typing, eye tracking, document annotations. However, replying solely on an autonomous agent can lead to issues with accuracy, as the agent may not fully understand the users' intentions. On the other hand, hybrid agents, such as AI chain [9], involve users in the loop to confirm the plan and provide feedback.

3.2.3 Integration of internal data. It is often challenging to leverage organisation's internal data (e.g., documents, images, videos, audios, etc.) to improve the accuracy of FMs, as FMs are often inaccessible or expensive to retrain or fine-tune. To address this, there are two design options to consider. The first design option is **in-context learning**, which integrates the internal data through prompts. However, there is a token limit for the context, e.g., the latest version of GPT-4 has a maximum limit of 128k tokens, equivalent to approximately 300 pages of text [10]. It is difficult to do a prompt with context data larger than the token limit. Alternatively, another design option is **retrieval augmented generation (RAG)**, which uses a vector database (such as Pinecone [11] and Milvus [12]) for storing the personal or organisational internal data as vector embeddings. These embeddings can be used to perform similarity searches and enable the retrieval of internal data that are related to

[8]https://github.com/yoheinakajima/babyagi
[9]https://aichain.online
[10]https://openai.com/blog/new-models-and-developer-products-announced-at-devday
[11]https://www.pinecone.io
[12]https://milvus.io

specific prompts. Compared to the direct integration of internal data through prompting, using a vector database is more cost-efficient and can achieve better accuracy and RAI-related properties.

3.2.4 Model-level architecture. There are four design options for the model-level architecture. The first design option is **a combination of the FM and conventional models**. The second design option is **a chain of FMs**, which is modularised architecture, such as Socratic Models [20]. This architecture relies on a few FMs that are chained together and a limited number of AI and non-AI components to perform tasks 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 LLMs, visual LLMs and audio LLMs. 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. The third design option is to **employ multiple FMs in parallel** to perform the same task or enable a single decision. The fourth design option 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 [2]. With the rapidly growing capabilities, in this type of architecture, no external software components might be required.

3.2.5 Deployment of FMs. FM **size** can range from lightweight to heavyweight. Lightweight FMs have a smaller number of parameters, making them suitable for edge devices, while heavyweight models have a larger number of parameters, providing more powerful capabilities but requiring more computational resources. In terms of the **location**, FMs can be deployed on a server or at the edge. There is no maintenance cost for end users to deploy FM on a server. Deploying a FM at the edge reduces latency and allowing for real-time inference. Regarding the **serving mode**, online deployment involves making API calls to a remote server for inference, which requires an active internet connection but allows for model governance. Offline deployment enables the model to be served directly on a device without relying on internet connectivity. For example, PaLM 2 [13] is available in different sizes, Gecko, Otter, Bison and Unicorn. Geckos is a lightweight FM that can be deployed on mobile devices and serve offline.

3.3 Responsible-AI-by-Design

3.3.1 Model-level risk control. FM providers can use **reinforcement learning from human feedback (RLHF)** which allows humans to provide feedback on the quality of the responses and uses this feedback to adjust the model's parameters. **Adversarial testing** can be done with domain experts to test the accuracy and RAI-related properties through adversarial examples. By testing the model in this way, developers can identify vulnerabilities or weaknesses in the model's decision-making process. **Data governance measures** are necessary to examine the reliability of data sources and data biases.

3.3.2 System-level risk control. To prevent harmful dual-use, the FM providers should impose restrictions on FM usage and prevent users from getting around of restrictions through unauthorised reverse engineering or modification of the system design. The FM can be provided as AI services on cloud servers so the interactions can be managed **through API controls**. **Continuous risk assessor** continuously monitors and assesses the AI systems based on AI risk metrics to prevent the misuse of the AI systems and to ensure the trustworthiness of the systems. **Guardrails** can be set up to ensure the inputs and outputs are compliant with RAI rules and policies. Prompts can be appended/generated or modified to make the prompts or the responses more accurate or responsible. The **think aloud** pattern can be used to disclose the decision-making process, such as the intermediate steps like prompt pattern implementation and verification/validation. This design can help build human trust in the system, but it may sacrifice data privacy. To protect business data and intellectual property, **think silently** might be needed so the details of intermediate process are revealed. A **verifier** can verify or modify the responses, and provide feedback. A **human expert** can review and verify the responses returned by a FM-based system before they are sent to users. This type of verifier can lead more accurate and responsible responses, but it can also be more time-consuming, expensive, and subjective/biased than automated verifiers. An **AI verifier** can use machine learning or other types of AI solutions to identify inaccurate or irresponsible responses [3]. For example, a blockchain smart contract can automatically verify and enforce the rules in an immutable and transparent way. However, it can be costly and limited by blockchain performance. This can have more accurate and responsible responses over time, but it may be limited by the quality of the knowledge data used. Also, an AI verifier may not be able to explain its decisions and may be prone to errors or bias if the knowledge data is incomplete or biased. **Hybrid verifier** uses a combination of the above types of verifiers to ensure the trustworthiness of responses. This can be advantageous over any individual verifier, but it can also be more complex and resource-intensive to implement and maintain. By recording critical data in an immutable data ledger, the **RAI black box recorder** allows for accountability analysis after incidents. This includes data such as the input and output of FMs and small AI models, the versions of FMs and small (distilled) AI models, etc. To ensure traceability, the **co-versioning registry** can be applied to co-version the AI components, such as FMs, fine-tuned models, or distilled small models. All software components including the FM procured from third parties can be associated with an **AI bill of materials** AIBOM that records their supply chain details, which can include RAI metrics or verifiable RAI credentials. This procurement information can be maintained in an AIBOM registry. The **standardised reporting** can be used to inform stakeholders (such as regulators and users) about the development process and product design of AI systems, such as AIBOM information about the FMs and data/model cards.

4 CONCLUSION

In this paper, we present a taxonomy of FM-based systems from a software perspective. In the future, we plan to building a pattern catalogue for designing FM-based systems.

[13]https://ai.google/discover/palm2

REFERENCES

[1] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[2] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378* (2023).

[3] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. Gptscore: Evaluate as you desire. *arXiv preprint arXiv:2302.04166* (2023).

[4] Ian Gorton, John Klein, and Albert Nurgaliev. 2015. Architecture knowledge for evaluating scalable databases. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 95–104.

[5] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352* (2023).

[6] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[7] Kevin Maik Jablonka, Philippe Schwaller, and Berend Smit. [n. d.]. Is GPT-3 all you need for machine learning for chemistry?. In *AI for Accelerated Materials Design NeurIPS 2022 Workshop*.

[8] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[9] Qinghua Lu, Liming Zhu, Xiwei Xu, and Jon Whittle. 2023. Responsible-AI-by-Design: A Pattern Collection for Designing Responsible AI Systems. *IEEE Software* (2023).

[10] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, Didar Zowghi, and Aurelie Jacquet. 2022. Responsible AI Pattern Catalogue: a Multivocal Literature Review. *arXiv preprint arXiv:2209.04963* (2022).

[11] Nikunj R Mehta, Nenad Medvidovic, and Sandeep Phadke. 2000. Towards a taxonomy of software connectors. In *ICSE*. 178–187.

[12] Michael Moor, Oishi Banerjee, Zahra Shakeri Hossein Abad, Harlan M Krumholz, Jure Leskovec, Eric J Topol, and Pranav Rajpurkar. 2023. Foundation models for generalist medical artificial intelligence. *Nature* 616, 7956 (2023), 259–265.

[13] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. 2023. ClimaX: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343* (2023).

[14] Douglas C Schmidt, Jesse Spencer-Smith, Quchen Fu, and Jules White. [n. d.]. Cataloging Prompt Patterns to Enhance the Discipline of Prompt Engineering. ([n. d.]).

[15] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580* (2023).

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[17] Eva AM van Dis, Johan Bollen, Willem Zuidema, Robert van Rooij, and Claudi L Bockting. 2023. ChatGPT: five priorities for research. *Nature* 614, 7947 (2023), 224–226.

[18] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564* (2023).

[19] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080* (2021).

[20] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. 2022. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598* (2022).