A Functional Software Reference Architecture for LLM-Integrated Systems

Alessio Bucaioni*, Martin Weyssow† Junda He†, Yunbo Lyu†, David Lo†

* Mälardalen University (Sweden), *alessio.bucaioni@mdu.se*

† Singapore Management University (Singapore), {*mweyssow,jundahe,yunbolyu,davidlo*}*@smu.edu.sg*

Abstract—The integration of large language models into software systems is transforming capabilities such as natural language understanding, decision-making, and autonomous task execution. However, the absence of a commonly accepted software reference architecture hinders systematic reasoning about their design and quality attributes. This gap makes it challenging to address critical concerns like privacy, security, modularity, and interoperability, which are increasingly important as these systems grow in complexity and societal impact. In this paper, we describe our *emerging* results for a preliminary functional reference architecture as a conceptual framework to address these challenges and guide the design, evaluation, and evolution of large language model-integrated systems. We identify key architectural concerns for these systems, informed by current research and practice. We then evaluate how the architecture addresses these concerns and validate its applicability using three open-source large language model-integrated systems in computer vision, text processing, and coding

Index Terms—Software reference architecture, functional reference architecture, LLMs

I. INTRODUCTION

The integration of large language models (LLMs) into software systems is transforming the software landscape, enabling capabilities like natural language understanding and autonomous task execution [3]. LLMs are driving innovation across domains such as customer service, healthcare, and education [7], [17]. However, serving pre-trained or fine-tuned models to end-users introduces complexities, including modularity and reuse [26], quality attribute trade-offs [23], and challenges like bias, fairness, regulatory compliance [10], and ethical use. Software architecture plays a key role in shaping system quality [5]. Reference architectures (RAs) provide standardized guidance for design, reuse, and evolution [11], [12]. Well-known RAs, such as AUTOSAR and the NIST Big Data Framework, have reduced costs, improved interoperability, and promoted best practices [12]. More recently, the digital twin domain has highlighted their value for emerging fields [12]. Despite the rapid adoption of LLM-integrated systems designed to serve inference capabilities to end-users, no documented RAs exist to address their design and quality attributes. This gap leaves stakeholders without systematic guidance to tackle critical concerns, hindering design reuse and scalability. As these systems grow in complexity and societal impact, the need for a guiding framework becomes increasingly urgent.

In this paper, we present our *emerging* results for a preliminary functional reference architecture to address the challenges of LLM-integrated systems. We identify and analyze key architectural concerns for systems serving pre-trained or fine-tuned models for inference tasks, then propose a preliminary functional software RA [6], [12]. Our work is grounded in a review of software architecture literature for LLMs and our experience in LLM research and applications [20], [27]. Finally, we evaluate the proposed architecture against the identified concerns and validate its applicability through three open-source LLM-integrated systems in computer vision, text processing, and coding.

The remainder of this paper is organized as follows. Section II presents the architectural concerns for LLM-integrated systems. Section III details the proposed reference architecture. Section IV validates the architecture against the identified concerns and analyses three open-source LLM-integrated systems. Finally, Section V concludes with a discussion and outlines directions for future research.

II. ARCHITECTURAL CONCERNS

This section presents the architectural concerns for LLM-integrated systems, identified through a targeted literature review and enriched by established architectural principles and extensive experience in LLM research and applications [20], [27].

We searched IEEE Xplore, ACM Digital Library, and Scopus [16] using a concise query string applied to titles and abstracts: *"software architecture" AND ("LLM*" OR "large language model*")*. The search yielded 38 peer-reviewed publications, refined to 26 after removing duplicates and irrelevant entries. Applying inclusion criteria focused on peer-reviewed studies addressing LLM software architectures [4], the set was narrowed to two studies. To mitigate biases [13], backward and forward snowballing [24] identified one additional study, resulting in three primary studies [14], [18], [19] [1]. These studies were analysed and further enriched with insights from architectural principles and practical experience, leading to the concerns summarized in Table II. It is important to note that we do not claim these to be the only or exhaustive architectural concerns for such systems.

LLM integration requires exposing model capabilities as reusable services to avoid monolithic embedding, enabling modularity, scalability, and dynamic updates. For instance, OpenAI's GPT services leverage RESTful APIs for modular access. *Data handling* and *performance* are critical for managing structured and unstructured data efficiently while minimizing latency and maintaining reliability. Optimized

[1]The interested reader can refer to the automatic search and selection process detailed in [8]

TABLE I
ARCHITECTURAL CONCERNS

Concern	Description
LLM integration	Incorporating LLMs into software systems involves exposing their capabilities as reusable services.
Data handling	LLMs require efficient data pipelines and handling of structured and unstructured data.
User interaction	Interfaces should provide seamless access while abstracting complexity.
Performance	Minimizing latency during inference while maintaining accuracy.
Scalability	Handle increasing loads (e.g., concurrent queries, larger datasets).
Security	Protecting against threats like unauthorized access, malicious prompts, and data breaches.
Privacy and compliance	Adherence to data privacy regulations (e.g., GDPR, EU AI Act).
Modularity	Breaking down LLMs functionalities into reusable components.
Customizability	Customizing LLMs for domain-specific tasks through prompt engineering or retrieval-based methods.
Interoperability	Ensuring seamless integration of LLMs with existing tools, databases, and APIs.
Fairness	Addressing biases to ensure equitable and ethical usage.

architectures using caching, parallelization, and hardware acceleration, as demonstrated by HuggingFace's Transformers library [25], ensure low-latency inference and efficient resource utilization. *Scalability* addresses the challenges of large model sizes and high concurrent query volumes. Solutions like OpenAI's GPT-4 [3] APIs use distributed clusters and horizontal scaling to manage millions of daily requests, ensuring robust performance under load. *User interaction* focuses on seamless, intuitive interfaces that abstract LLM complexity while providing mechanisms for feedback and control over probabilistic outputs. Microsoft's GitHub Copilot exemplifies this through its integration with IDEs, enhancing usability and developer productivity. *Security* and *privacy* are essential to safeguard against threats such as unauthorized access, malicious prompts, and data breaches. Measures like input validation, encryption, and secure deployment pipelines are critical, with OWASP highlighting risks like prompt injection. Federated learning, as used by Google on Android devices, ensures compliance with privacy regulations like GDPR by minimizing data exposure. *Modularity* simplifies updates and enhances scalability by breaking LLM functionalities into reusable components. AWS Sagemaker exemplifies this with modular deployments of preprocessing and inference components, enabling flexible system maintenance and scaling. *Customizability* supports domain-specific adaptations without disrupting core architecture through methods like prompt engineering and fine-tuning. OpenAI's GPT fine-tuning API allows tailoring models for use cases such as customer support and legal text generation. *Interoperability* ensures seamless integration with existing tools, databases, and APIs, avoiding adoption barriers. Salesforce's Einstein GPT demonstrates this by using adapters to interact smoothly with CRMs and workflows. *Fairness* focuses on mitigating biases to ensure equitable and ethical usage. Google's AI ethics team addresses this through bias detection and mitigation strategies during model training, promoting fairness and inclusivity.

III. PROPOSED SOFTWARE REFERENCE ARCHITECTURE

In this section, we present the preliminary functional software reference architecture (RA) for LLM-integrated systems. Following Garces et al. [12], preliminary indicates that the RA synthesizes insights from existing systems while proposing

generalized principles for future design. Based on Behere and Törngren [6], functional emphasizes a black-box view of the system, detailing entities and interactions. This aligns with the functional view in ISO/IEC/IEEE 42010:2022, which standardizes the architectural description of software-intensive systems [15].
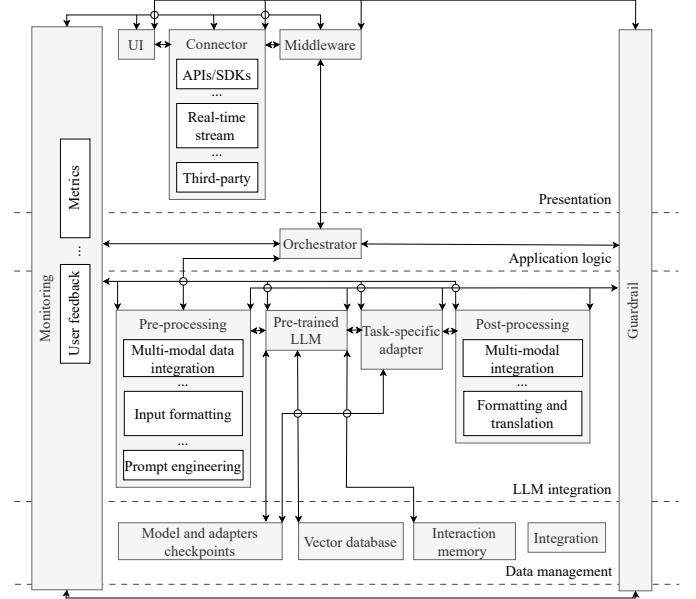

Fig. 1. Preliminary functional software RA for LLMs-integrated Systems

Figure 1 shows the RA, organized into four layers: *Presentation*, *Application logic*, *LLM integration*, and *Data management*, represented by dotted black lines. Each layer contains functional components (grey boxes) and optional subcomponents (white boxes), representing example functionalities with ellipses indicating possible extensions. Two sidecars—*Monitoring* and *Guardrail*—span multiple layers. Connections are shown with directional arrows, and overlapping connections are marked by black-bordered white circles for clarity. Each component operates as an independent service to enable modular updates and scaling of LLM functionalities.

The Presentation layer facilitates user interaction and visualization, acting as the entry point for external systems and users. It includes a *UI* component for interfaces like web apps, mobile applications, and chatbots, such as ChatGPT's web interface. The *Connector* bridges LLMs with external services, supporting real-time streams (e.g., WebSocket) and third-party integrations, such as Slack's integration with ChatGPT for task management. The *Middleware* handles request validation, transformation, and logging, maintaining conversational context similar to OpenAI's Session Management Layer. Communication within this layer typically uses secured HTTP/HTTPS or GraphQL, while event-driven middleware ensures responsive interactions. The Application logic layer manages task and data flow within the system. Its core component, the *Orchestrator*, dynamically determines workflows based on user inputs, such as summarization or translation. For example, Jasper AI's Content Workflow demonstrates this functionality. The Orchestrator supports asynchronous, event-driven workflows to ensure efficient execution and scalability.

The LLM integration layer is the system's core, handling input and output processing. The *Pre-processing* component formats and enriches inputs with tasks like tokenization, prompt engineering, and multi-modal integration, as seen in Google's Vertex AI Pipelines. The central inference relies on the *Pre-trained LLM* and *Task-specific adapters* for fine-tuned, domain-specific responses, such as ChatGPT's adapters for FAQs or personalized support. The *Post-processing* component refines outputs (e.g., formatting, translation) for user-friendly delivery, exemplified by DALL-E's Caption Formatter, which converts image outputs into natural language descriptions. Components communicate via RESTful APIs, gRPC for low latency, or shared caching mechanisms. The Data management layer ensures efficient data handling. *Model and adapter checkpoints* store pre-trained models and fine-tuned adapters for seamless updates. A *vector database* enables retrieval-augmented generation (RAG) for knowledge-grounded outputs, improving accuracy, as seen in Pinecone's integration with Notion AI. The *Interaction memory* component maintains context across sessions. Long-term memory stores persistent user data (e.g., profiles, preferences, and error history), while short-term memory retains session-specific chat context, as exemplified by ChatGPT's Memory Feature. *Integration* components connect with external APIs, knowledge bases, and tools, ensuring system interoperability. The *Monitoring* sidecar spans all layers, collecting metrics like latency, throughput, and system health, along with user feedback. For example, Azure OpenAI's Metrics Dashboard monitors inference latency, token usage, and error rates to identify bottlenecks and improve performance. The *Guardrail* sidecar enforces security and compliance across layers, validating user inputs and ensuring adherence to regulations like GDPR and the EU AI Act. For instance, Microsoft Copilot's Compliance Checks prevent outputs containing confidential or inappropriate content.

Intra-component communication ensures seamless interaction across all layers. The Presentation layer communicates with the Application logic layer via secured HTTP/REST or GraphQL calls, returning responses in JSON. The Application logic layer connects to the LLM integration layer through synchronous (REST/GraphQL) and asynchronous (e.g., RabbitMQ, Kafka) methods for real-time tasks and batch processing. The LLM integration layer exposes APIs through REST or gRPC, with outputs serialized as JSON or Protocol Buffers (Protobuf) for high performance. The Data management layer provides data to the LLM integration layer via REST APIs or database queries, storing embeddings, interaction histories, and structured data for context-aware operations. The Monitoring sidecar collects metrics asynchronously using message brokers like Kafka and aggregates them into a central system. Alerts and anomalies are communicated back to the Application logic layer via REST APIs or notification services. The Guardrail sidecar validates inputs inline at the Presentation layer and enforces security through synchronous REST APIs or policy hooks.

IV. VALIDATING THE PROPOSED REFERENCE ARCHITECTURE

This section evaluates how the proposed RA addresses the identified concerns and validates its applicability through three open-source LLM-integrated systems in computer vision, text processing, and coding.

A. Proposed reference architecture Vs. architectural concerns

Table IV-A demonstrates how the proposed reference architecture (RA) addresses key architectural concerns through a modular, layered design with cross-layer components.

TABLE II
PROPOSED REFERENCE ARCHITECTURE VS. ARCHITECTURAL CONCERNS

Concern	How the concern is satisfied
LLM integration	The LLM integration layer encapsulates LLM functionalities as modular services, accessible via APIs for easy integration and independent updates.
Data handling	Data pre-processors ensure efficient formatting and batching; adapters integrate external data sources; vector databases support retrieval-augmented generation.
User interaction	The Presentation layer provides user-facing interfaces and unified access to underlying services through API gateways, ensuring smooth interactions and validation.
Performance and scalability	Distributed processing and asynchronous workflows optimize performance under high loads; auto-scaling and caching ensure efficient handling of concurrent requests.
Security, privacy, compliance and fairness	Secure communication with TLS; authentication managed at Integrator; Guardrail enforces compliance policies, logs, and mitigates risks like malicious prompts and breaches.
Modularity	Layers separate concerns; microservices allow independent updates and scaling of functionalities.
Customizability	Fine-tuning and inference pipelines are modular, enabling domain-specific adaptations and reusable components without disrupting the core architecture.
Interoperability	Standardized APIs and integration adapters ensure seamless interaction with external systems, databases, and workflows.

LLM integration is handled by the LLM integration layer, which modularizes pre-processing, inference, and post-processing using microservices and proxy patterns. These components operate independently, accessible via standardized APIs, ensuring seamless integration, scalability, and maintainability. Data handling is supported by the data management layer, which uses vector databases for retrieval-augmented generation (RAG) and embedding-based operations, enabling low-latency access to external knowledge. The interaction memory component maintains session context, enhancing personalization and enabling context-aware user interactions. User interaction is addressed in the presentation layer, which provides adaptable interfaces, such as web apps and chatbots, to abstract LLM complexity. Integrators and middleware ensure smooth, real-time communication with external systems by managing request validation, transformation, and session handling. Performance and scalability are achieved through event-driven communication, distributed processing, and horizontal scaling. Asynchronous workflows optimize resource utilization, while caching mechanisms and task-specific optimizations improve response times for high-throughput applications. The Monitoring and Guardrail sidecars address security, compliance, and system optimization. Monitoring collects runtime metrics like latency and error rates, enabling real-time performance tracking and proactive improvements. Guardrails enforce compliance with privacy regulations (e.g., GDPR) and mitigate risks such as malicious prompts, ensuring secure and ethical operations. Fairness and privacy are reinforced through bias detection and regulatory compliance enforced by the Guardrail sidecar. Monitoring further supports fairness by detecting anomalies and enabling continuous improvements. Modularity and interoperability are

Component	MaxKB	Continue	InternVL
UI	UI in Vue.js	Continue Sidebar, Chat, In-editor UI	UI in Streamlit Python
Connector	RESTful APIs, LangChain Connectors	VSCode API, Web View, LLM Providers APIs	RESTful APIs
Middleware	Request Validation, Input Transformation in Django	–	Request Validation, Input Transformation
Orchestrator	Workflow Engine	VSCode Extension	Workflow Engine
Pre-processing	Context Retrieval, Prompt Reformulation, Tokenization, Image Transformation in Langchain	Context Retrieval and Prompt Reformulation	Caption Generation, Image Transformation, Tokenization
Pre-trained LLM	Multiple LLMs	Multiple LLMs	Multiple LLMs
Task-specific adapter	–	–	LoRA
Post-processing	Output Parsing, Filtering, and Formatting in LangChain	Output Filtering, Code Change Integration	Output parsing (Multi-modality), Output Formatting
Model and adapters checkpoints	Load LLMs via LangChain or Locally via Ollama	Load LLMs from providers'APIs or Locally via Ollama	Load LLMs from providers' APIs
Vector Database	Knowledge Vectors with pgvector	Codebase Vectors	–
Interaction Memory	Session Storage, Conversational Memory (PostgreSQL)	Cache Completion	Session Storage, Conversational Memory
Integration	RESTful APIs, Customized Functions & tools	RESTful API (e.g., OpenAI, Anthropic, Ollama)	RESTful API (e.g., OpenAI-compatible)
Monitoring	System Usage, User Feedback	Telemetry (PostHog), User Feedback	User Feedback
Guardrail	–	–	Safeguard for content moderation

strengths of the LLM integration layer, where independent pre-processing, inference, and post-processing components allow seamless updates and domain-specific adaptations, such as fine-tuning or prompt engineering, without affecting the core system. Customizability is achieved via modular task-specific adapters and fine-tuned models, enabling tailored outputs for diverse domains while preserving system stability, efficiency, and reliability.

B. Proposed reference architecture Vs. existing LLM-integrated systems

Table IV-A maps three open-source LLM-integrated systems—spanning text processing, coding, and computer vision—to the proposed preliminary functional software RA.

Max Knowledge Base (MaxKB) combines LLMs with external knowledge retrieval to deliver accurate, domain-specific responses [2]. It provides a Vue.js-based user interface for interaction and supports workflow orchestration through drag-and-drop components. MaxKB integrates with third-party tools for dynamic task execution and uses PostgreSQL with pgvector to store document embeddings for RAG. Documents are segmented and stored as embeddings to enable efficient retrieval and Dialogue logs maintain session histories and user feedback, enabling query analysis over time ranges (e.g., past 7, 30, or 180 days) for refining AI responses. Built on LangChain [9], MaxKB integrates multiple LLM providers, such as Llama 3, Qwen 2, OpenAI, Claude, and dynamically combines top-k retrieved segments into contextually relevant prompts, aligning with the LLM Integration Layer for pre-processing and inference.

Continue, a coding assistant integrated with VSCode and boasting nearly 20,000 GitHub stars [1], exemplifies modular LLM integration. It offers three interfaces: a sidebar for configuration, a chat for LLM interactions, and in-editor tools for tasks like code autocompletion and bug fixing. Continue connects to LLM providers (e.g., OpenAI, Anthropic, Google Gemini) and supports local LLMs via Ollama. The Orchestrator in the Application Logic Layer activates workflows like docstring generation or code autocompletion. The plugin indexes the project codebase into a vector database that is leveraged by a context retrieval module to augment user prompts with contextually relevant code. It supports multiple LLMs for code autocompletion, chat, code editing, and code retrieval. LLMs outputs are filtered to remove redundancy before merging accepted changes into the codebase, aligning with the RA's Post-processing. Telemetry powered by PostHog tracks user interactions, while feedback mechanisms allow issue reporting, aligning with the Monitoring Sidecar.

InternVL supports multi-modal tasks such as text generation, reasoning, and dialogue, processing images, videos, and text [21]. Its Streamlit-based user interface allows local and web-based interaction, while RESTful APIs support external integrations, mapping to the Presentation Layer. The middleware handles input validation, transformation (e.g., tokenization, image resizing), and logging, ensuring session context for multi-turn conversations. The Orchestrator dynamically selects workflows for text, images, or videos, enabling tasks like captioning, multi-image reasoning, and video processing. The Pre-processing component handles tokenization and image transformations, while Post-processing refines outputs for integration. InternVL uses pre-trained multimodal LLMs (e.g., InternVL2-76B) with support for fine-tuning, including parameter-efficient methods like LoRA, aligning with the LLM Integration Layer. Interaction memory preserves session

context for multi-turn conversations, enhancing personalization. LMDeploy enables OpenAI-compatible API deployment, supporting interoperability. Telemetry tracks user interactions (e.g., upvotes, flags) for performance improvement, while Guardrails enforce content moderation, ensuring outputs adhere to ethical and safety guidelines.

V. FINAL DISCUSSION AND FUTURE STEPS

Compared to existing work, our study provides a novel contribution by proposing a preliminary functional software RA for LLM-integrated systems. Unlike related work, such as foundation model-based systems [14], our RA explicitly addresses unique architectural concerns of LLM-driven systems, including modular pre-processing, dynamic task orchestration, and cross-layer monitoring. While our work is grounded in software architecture literature, established principles, and extensive experience in LLM research, we acknowledge limitations in the scope and systematicity of our review. However, given the limited availability of peer-reviewed studies on LLM-integrated systems, this is unlikely to introduce significant bias, as additional findings would more likely complement rather than challenge our proposed RA. The proposed RA synthesizes insights from existing systems while also proposing generalized principles for future LLM-integrated system designs. It is important to note that current systems may not necessarily include all the identified RA components (e.g., Task-specific adapters and Guardrail components), which is typical for RAs, as demonstrated in previous research on domains like DTs [11]. Additionally, there may be architectural concerns not yet considered, which is a natural aspect of the early-stage development of such frameworks. We anticipate that further research and practical applications will help identify and address these gaps. To address the above limitations, we are conducting a systematic literature review, including grey literature, to capture broader perspectives and emerging insights into this rapidly evolving field. The proposed RA was validated against three open-source LLM-driven systems—MaxKB, Continue, and InternVL. However, further evaluation across real-world systems spanning diverse domains is essential to ensure broader coverage and uncover any overlooked challenges. As part of future work, we also plan to conduct focus group validations and expert surveys to gather qualitative feedback on the RA's design and its practical applicability in integrated system architectures. We also aim to extend this RA by incorporating dedicated architectural views, as suggested by the ISO/IEC/IEEE 42010 standard, such as the technical view, and views addressing unique concerns of LLM-integrated systems, including compliance and fairness. This extension may ultimately form a comprehensive architectural framework, similar to those introduced for the automotive domain [22].

REFERENCES

[1] "Continue, open-source ai code assistant." https://github.com/continuedev/continue, accessed: 2024-12-09.

[2] 1Panel-dev, "Maxkb: An open-source knowledge base q&a system," https://github.com/1Panel-dev/MaxKB, 2024, accessed: 2024-12-12.

[3] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[4] N. B. Ali and K. Petersen, "Evaluating strategies for study selection in systematic literature studies," in *Procs of ESEM*, 2014.

[5] L. Bass, *Software architecture in practice*. Pearson Education India, 2012.

[6] S. Behere and M. Törngren, "A functional reference architecture for autonomous driving," *Information and Software Technology*, vol. 73, pp. 136–150, 2016.

[7] 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.

[8] Bucaioni et al., "Automatic search, and selection," https://docs.google.com/spreadsheets/d/1Tm3m4iMFQsZMHGfhCH6NjhbIq_UKyk8T/edit?usp=share_link, accessed: 2024-12-02.

[9] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[10] European Union, "The artificial intelligence act," https://artificialintelligenceact.eu, accessed: 2024-12-02.

[11] E. Ferko, A. Bucaioni, P. Pelliccione, and M. Behnam, "Standardisation in digital twin architectures in manufacturing," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. IEEE, 2023, pp. 70–81.

[12] L. Garcés, S. Martínez-Fernández, L. Oliveira, P. Valle, C. Ayala, X. Franch, and E. Y. Nakagawa, "Three decades of software reference architectures: A systematic mapping study," *Journal of Systems and Software*, vol. 179, p. 111004, 2021.

[13] T. Greenhalgh and R. Peacock, "Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources," *BMJ*, vol. 331, no. 7524, pp. 1064–1065, 2005.

[14] T. Händler, "A taxonomy for autonomous llm-powered multi-agent architectures." in *KMIS*, 2023, pp. 85–98.

[15] *ISO/IEC 20233:2019 - Information technology – Cloud computing – Interoperability and portability*, International Organization for Standardization Std., Oct. 2019, accessed: 2024-12-08. [Online]. Available: https://www.iso.org/standard/74393.html

[16] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and software technology*, 2013.

[17] Klarna, "Klarna AI Assistant Handles Two-Thirds of Customer Service Chats in Its First Month," https://www.klarna.com/international/press/klarna-ai-assistant-handles-two-thirds-of-customer-service-chats-in-its-first-month/, accessed: 2024-12-02.

[18] 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," in *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, 2024, pp. 1–6.

[19] Q. Lu, L. Zhu, X. Xu, Z. Xing, S. Harrer, and J. Whittle, "Towards responsible generative ai: A reference architecture for designing foundation model based agents," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2024, pp. 119–126.

[20] R. Nazir, A. Bucaioni, and P. Pelliccione, "Architecting ml-enabled systems: Challenges, best practices, and design decisions," *Journal of Systems and Software*, vol. 207, p. 111860, 2024.

[21] OpenGVLab, "Internvl," 2024, accessed: 2024-12-12. [Online]. Available: https://github.com/OpenGVLab/InternVL

[22] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of volvo cars," *Journal of systems architecture*, vol. 77, pp. 83–100, 2017.

[23] S. Shashidhar, A. Chinta, V. Sahai, Z. Wang, and H. Ji, "Democratizing llms: An exploration of cost-performance trade-offs in self-refined open-source models," *arXiv preprint arXiv:2310.07611*, 2023.

[24] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Procs of EASE*. ACM, 2014, pp. 38:1–38:10.

[25] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[26] C. Xiao, Z. Zhang, C. Song, D. Jiang, F. Yao, X. Han, X. Wang, S. Wang, Y. Huang, G. Lin *et al.*, "Configurable foundation models: Building llms from a modular perspective," *arXiv preprint arXiv:2409.02877*, 2024.

[27] Z. Yang, Z. Sun, T. Z. Yue, P. Devanbu, and D. Lo, "Robustness, security, privacy, explainability, efficiency, and usability of large language models for code," *arXiv preprint arXiv:2403.07506*, 2024.