Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Truong, Linh; Vukovic, Maja; Pavuluri, Raju

On Coordinating LLMs and Platform Knowledge for Software Modernization and New Developments

Published: 22/11/2023

Document Version
Early version, also known as pre-print

Please cite the original version:
Truong, L., Vukovic, M., & Pavuluri, R. (2023). *On Coordinating LLMs and Platform Knowledge for Software Modernization and New Developments*.

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

On Coordinating LLMs and Platform Knowledge for Software Modernization and New Developments

Hong-Linh Truong
Department of Computer Science, Aalto University
Finland
linh.truong@aalto.fi

Maja Vukovic, Raju Pavuluri
IBM Thomas J. Watson Research Center
USA
maja@us.ibm.com,pavuluri@us.ibm.com

ABSTRACT

Emerging generative and fine-tuning LLMs services have been widely benchmarked and used for various software development tasks. These LLMs services are powerful but have different output qualities for software development tasks and may not be able to deal with complex development tasks in edge-cloud software modernization and new developments due to their generative capabilities and lack of up-to-date (domain) knowledge. Many queries and solutions related to target platforms, deployment configurations, policies, data regulation, observability, to name just a few, are not well integrated with these LLMs, but are accessed by the developer through other sources. In this work, we discuss situations where the gaps between the needs and the offerings from LLMs can be compensated by Platform Knowledge, which captures knowledge about, e.g., software, service and infrastructure catalogs, architectural decision records and code patterns. We propose **COLLMS** – a framework for coordinating LLMs services and Platform Knowledge. At the starting point of the framework, we will discuss challenges for achieving the coordination centered around Platform Knowledge, LLMs management and integration, quality-aware coordination of LLMs, and observability and knowledge updating.

ACM Reference Format:
Hong-Linh Truong and Maja Vukovic, Raju Pavuluri. 2023. On Coordinating LLMs and Platform Knowledge for Software Modernization and New Developments . In *Proceedings of Author version (Preprint).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Rapid changes in edge-cloud computing continuum platforms and corresponding application models for data intensive workloads have required a huge amount of work in edge-cloud software modernization as well as new developments. Edge/Cloud modernization focuses on techniques, methods and tools that help the developer to transform their existing code/applications to (new, native) edge-cloud native environments and software systems [28, 32, 40]. The modernization reflects the goal to achieve target artifacts for edge-cloud native environments – modernizing software/code, as well as the new, improved engineering process to achieve the target artifacts – modernizing development processes.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Preprint, November 2023, Espoo, Finland
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Although the use of LLMs for edge/cloud modernization and code development productivity has been recently investigated and applied intensively [21, 23], there exist several open questions on how to use LLMs and other assistant services for edge-cloud applications and services development. We believe that coordination is one of the key challenges that need to be addressed for utilizing LLMs and other services for modernization and new developments. Many works have been focused on training foundation models, fine-tuning models, and developing prompt engineering workflows and templates for different development scenarios. However, the state-of-the-art has reported that to use an LLM for the right development tasks, one must carefully invoke the LLM in the right way, especially carefully coordinating steps of invoking the LLM. Furthermore, task flows have also recently been investigated for invoking external knowledge sources to provide contextual information for invoking LLMs, such as using LLMs and vectorization in Retrieval Augmented Generation (RAG) [27]. Given many choices of LLMs in the ecosystem, we have observed that LLMs offerings under the service model can be selected and combined with others [31]. Thus, coordination of LLMs is of paramount importance but current coordination techniques for software modernization and new developments have several limitations:

- focus on individual LLMs: this does not reflect the diverse types of tasks and services we need for software modernization and new developments
- focus on limited quality aspects, e.g., reliability or costs: this does not support the end-to-end needs for several related tasks of the software development.
- lack the integration with many important services and platform knowledge: this does not connect common tools and knowledge sources that the developer has access to in the developer environment (e.g., on-premise tools and private data sources), specially suitable for software modernization and new developments.

We propose to consider the coordination problems as one of the key requirements for employing LLMs, fine-tuning models and other important domain-specific services for software modernization and new developments. We present further related work and background in Section 2. We describe our coordination framework and challenges in Section 3, as well as a short discussion of the prototype. Section 4 concludes our work.

2 BACKGROUND AND RELATED WORK

2.1 LLMs for software development

Many LLMs and fine-tuning models have been used, developed and provided for software development. The recent survey [21]

has indicated many of them, such as Code Llama [34], StarCoder, etc. For certain tasks in software development, many LLMs can help. Each LLM service has different pros and cons. LMOps [11][26] focuses on building applications using LLMs and Generative AIs. Many papers have been introduced for combing different LLMs to build applications. Modernization and new development are part of application development that can benefit from LLMs. However, most LMOps papers do not focus on tasks for software component development. Our goal in this work is not to provide a new LLM or fine-tuning models for software development but examine the challenges of coordinating them for software development.

2.2 Coordination of LLMs

AutoMix with its verifier [31] uses a small language model to get an initial answer and then check the answer with a verifier to decide if the query should be given to the black-box LLM. It is not designed for software engineering but since it uses black-box LLMs, it could be tested for software development. Retrieval Augmented Generation (RAG) and Self-RAG [16] can use external sources to augment prompts for querying LLMs. There exist several tutorials on how to combine vector database for retrieving contexts to be used for querying LLMs, such as AnyScale [1]. Although, RAG-based techniques are generic and can be used to combine with information from domain and platform knowledge, currently, RAG is investigated and applied to a single LLM. In our work, by introducing and utilizing Platform Knowledge and coordinating multiple services, we aim at integrating RAG methods with many types of domain-specific, up-to-date knowledge for multiple LLMs.

Langchain [8], Semantic Kernel [13] and their prompt engineering techniques support the coordination of LLMs to certain degree. Llamaindex [10] allows to configure and query different LLMs. However, in our view, they just support basic "local" coordination through manual switching LLMs in a group, whereas the software development would require coordination workflows of many groups for different tasks. How to combine LLMs and provide a "langchain" for edge-cloud software modernization or new development is still open. CodeRabbit [2] supports code review by calling many LLMs and do the postprocessing of results returned by LLMs. It uses multiple LLMs but with a static configuration. There exist works that break an input into multiple sub inputs and invoke LLMs for sub inputs, such as [30, 37]. They are tested with image/text and video tasks, not for software development. Moreover, due to their domains, they use and implement a fixed coordination model. We do not focus on automatic task decomposition in this work and our goal is to provide customized coordination workflows.

2.3 Platform knowledge for Edge/Cloud software modernization and developments

We focus on edge/cloud software modernization and new developments as they are currently demanding, with many challenges and knowledge that require substantial support. The tools, methods and techniques for edge/cloud modernization and new developments need to capture key characteristics of the applications to be developed and the business use cases and requirements. Notable architectural styles and application models in the concern of the developer in the cloud/edge environments are microservices, serverless and workflows of serverless functions, batch operation workflows, and stream processing. Software development for these styles and models in edge-cloud environments requires various additional tasks and knowledge. Examples are knowledge about target infrastructures, platforms to be used, accepted software versions and licenses, issues related to loosely communication patterns, policy as code, and observability. Collectively, we consider them available in and supported by tools/services in *Platform Knowledge*.

Many refactoring techniques for modernization of applications (e.g., refactoring monolithic applications to microservices) are developed based on static code analysis and dependencies, and dynamic service/API invocations [24]. However, such techniques do not incorporate a wide set of requirements for edge/cloud modernization and new developments in the view of the developer, such as how to incorporate deployment strategies, data regulation, and observability requirements in today's edge-cloud environments. A very recent paper discussed the role of architecture knowledge and generative AI for software development [33]. Our coordination approach will consider Platform Knowledge, which also incorporates architecture knowledge obtained from public or private sources relevant to the development. Another important aspect is that, currently, new developments in edge-cloud environments require the integration of different types of capabilities within the application, including AI/ML inference, stream processing, time series anomaly detection, real-time data quality control, to name just a few. To date, design suggestions and code generation for such capabilities and integration are not well supported by LLMs.

3 COLLMS: COORDINATING LLMS AND PLATFORM KNOWLEDGE

3.1 COLLMS framework overview

Figure 1 presents an overview of **COLLMS** that captures (i) common key tasks/stages of the software development, and (ii) required LLMs and other services to be coordinated for development tasks.

3.1.1 LLMs and fine tuning LMs services. Many LLMs/AI services can be selected, integrated and customized based on the need of the development. This is also subject to the business requirement for which **COLLMS** is deployed. Thus, we aim at supporting the selection of LLMs and fine-tuning models and other related services based on the configuration of, especially, the development context (e.g., task types, see Section 3.1.3). Furthermore, many services can be brought into the framework that are not necessary available from the outside of the deployment of the framework for a particular development environment. For example, a specific development team/company has many specific services used for the development that can be integrated into **COLLMS** for specific tasks. Given different groups of LLMs for specific tasks and different services in a group for tasks, we can employ different strategies to use LLMs, e.g., single, quorum, or parallel usage. These strategies can be configured and decided based on runtime quality.

3.1.2 Platform Knowledge. Platform Knowledge collectively includes specific ML models/services, software and service catalogs, data catalogs, architectural decision records, accepted solutions, and common development tools and services. It includes various

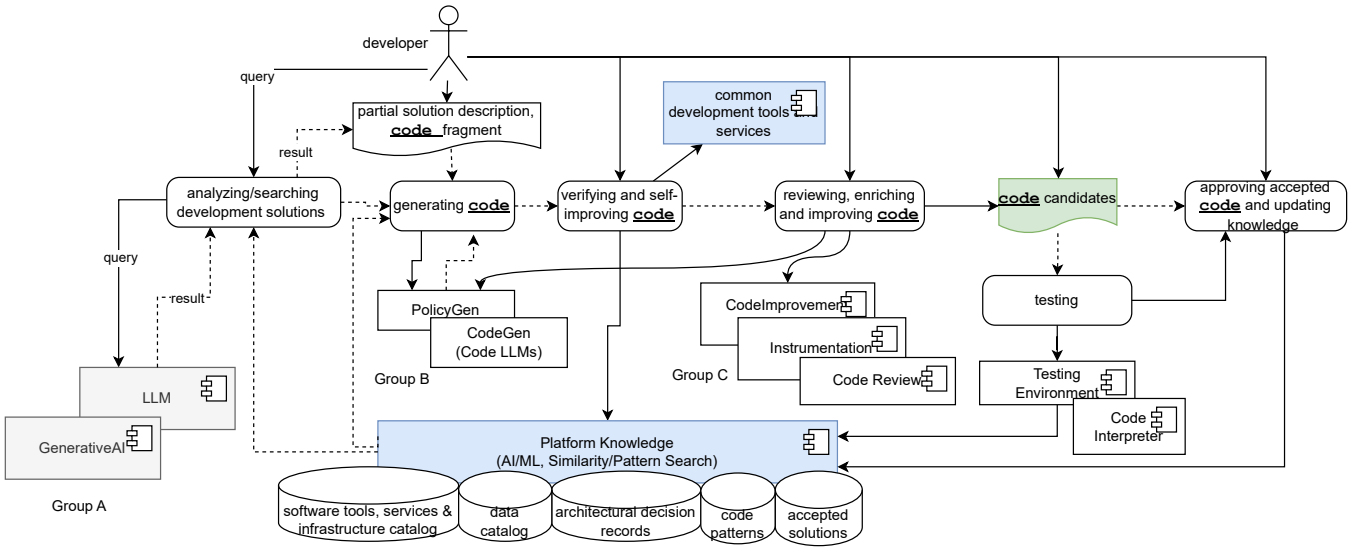

Figure 1: Overview of COLLMS tasks, LLMs and Platform Knowledge. code represents the example of tasks for developing code.

utilities and knowledge that provide some internal, private knowledge but also up-to-date knowledge for the modernization, which will be targeted to software systems and architectures constrained by business, e.g., edge-cloud native environments in our focus.

Platform Knowledge enables two situations: (i) complementing various types of knowledge missing in LLMs for improving LLM query and quality control and (ii) in-house solutions and services not offered by LLMs. For example, LLMs can provide code suggestions based on out-of-date software libraries due to rapid changes of software and infrastructures for the target environments. By scanning the code or running a quick (compiling) check, the quality of the suggestion can be detected. Another example is that recommendations for new developments that are not solvable by current LLMs should not be queried from LLMs to avoid overhead and wasted cost. While the development and configuration of Platform Knowledge may be similar to common, supporting services for the development, we emphasize the focus on building it suitable for the context of software modernization and the cooperation with LLMs.

3.1.3 Task types and models. In terms of developer's tasks and the development stages, we support customized task models, configured based on the developer need for seeking potential solutions for implementation, generation and testing of programming language code, configuration and deployment policies, and (structured) query languages. By focusing on highly related tasks reflecting a subset but crucial activities that the developer needs, we concentrate on specific development contexts and settings, such as analyzing possible solutions and generating code by LLMs, verifying and improving code by the developer, enriching/improving code with LLMs, verifying and updating knowledge for a specific solution by the developer. Thus, not all types of tasks in software engineering (such as requirement engineering) are in the scope of our work.

From the coordination viewpoint, requests from developers will be provided as inputs for tasks to be coordinated. Our focused task models specify *types of tasks*, e.g., `{searching, generating, enriching}`, *types of target solutions*, e.g. `{code, policy}`, and associated *development contexts*, e.g. `{Java code modernization, Python ML development}`. We define contexts based on the goal of the development, such as refactoring Java code for a large enterprise software, obtaining various metadata about programming languages, types of development and business concerns. The task type, target solution type, and context identify the scope for a developer's query and help to generate suitable concrete queries for LLMs (e.g., with prompts and other templates). The task models also help identify suitable LLMs and other services at runtime. Thus, we can have different strategies for matching of LLMs based on the type of tasks, the complexity and suitability of tasks given available groups of LLMs. Each task is associated with a model for quality of results evaluated by human or software. Our quality of results is defined based on generic metrics like cost, time, reliability, etc.

3.2 Challenging 1 – Platform Knowledge Design

We build Platform Knowledge atop Internal Developer Platform [39], code patterns, similarity search and specific tools/services:

- software framework, services and infrastructure catalog and data catalog: include information that are important for the applications to be developed (not about the LLMs to be used). The information includes possible internal and external software frameworks and services and infrastructures to be used. For example, the application may be constrained with the ecosystem of Kafka, Kubernetes and Snowflake.
- knowledge search capabilities: include various types of search for finding code patterns, policies, architectural design decisions, etc. This can also include other services like specific code generation services with fine-tuning ML models for the application domains.

Clearly, Platform Knowledge here is not like a typical "knowledge base" used for RAG in LLMs; it includes much more tools and is customized for the task model. Thus, the challenge is to associate

knowledge and tools with different contexts of the recommendation in cooperation with LLMs. In solving the gaps between the developer need and LLM supports, we consider different contexts, namely: *business context* (related to business constraints and costs on software), *operational context* (related to runtime quality, scalability, reliability, etc.), and *infrastructure context* (related to underlying deployable infrastructures). Platform Knowledge must arrange and update its various services and knowledge sources about domains/platforms related to the development incrementally. Platform Knowledge also enables the update of the result from the development that cannot be incorporated into LLMs/fine-tuning models due to time and cost constraints (e.g., it does not make sense to retrain/fine-tune LLMs with only little new information). For example, when accepting code candidates, updates about queries and solutions in Platform Knowledge could be useful for next tasks (and later become the source for retraining ML models).

3.3 Challenge 2 – Ensemble management for LLMs and fine-tuning models services

The management of LLMs services is complex. It includes (i) input and output representation and mapping, (ii) prompt templates for query enrichment, (iii) different strategies to employ prompt templates and knowledge from Platform Knowledge to improve queries and quality control, e.g., using RAG and composition techniques, and (iv) ensemble execution and aggregation in which, for solving a query, several services are used together. Different types of workflows involve prompt engineering, RAG and LLM services.

One of the first challenges is which LLMs/services should be in a group for a task type. We consider two stages. At the bootstrapping time, a group can be statically selected based on various conditions: preferences of the developer (based on their historical uses or limited access or cost), static AI/ML algorithm evaluations [25], and existing benchmarks (e.g., evaluation of LLMs for testing [35]). Currently, generic benchmarks like HuggingFace [3] or Stanford [4] or other benchmarks like [29] are not suitable. Specific benchmarks of LLMs for relevant tasks are not available but we foresee the design for custom benchmarks that, given a deployment for a specific setting, a fast execution benchmark can be executed to provide the selection of LLMs. However, as surveyed and noted in [5, 15], the quality of LLMs covers much more than metrics that can be automatically measured; many metrics are given by human or measured for human effort. Therefore, it is challenging to develop quality models for LLMs. At runtime, static configuration of a group can be changed. Members can be added and removed based on the quality observed from the monitoring of the task's results. However, we shall note that the number of members of a group maynot be large, thus mostly some suggestions based on historical results can help to reconfigure the group without using complex algorithms. Currently, we are working on different workflows, templates and tools. However, suitable workflows and the best ones cannot be determined in general.

3.4 Challenge 3 – Configuring and runtime quality-aware scheduling of task execution

3.4.1 Customizable and pluggin integration model. At the basic level, we must integrate different LLMs to be able to invoke them. It is actually quite challenging as the path from a developer query to LLM invocation is not trivial. Given an LLM the way to invoke it can be very different dependent on the design and the performance/quality purpose. Figure 2 explains our view on the integration. The way to call an LLM is carried out by an *Agent*, which is following a plugin model and managed by *ExecutionEngine*. Agents can implement different logic. The most simple agent could be just for single-shot, single call but complex agents can be implemented based on React [41], etc. Task query must be mapped to specific, concrete queries accepted by specific agents for LLM.

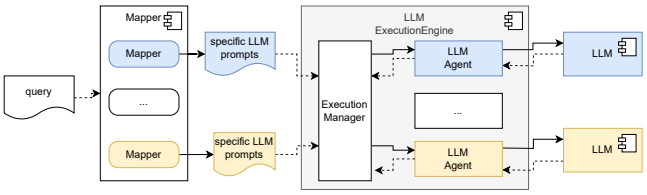

Figure 2: Customizable model for integrating multiple LLMs

3.4.2 Flow patterns in coordination. We design coordination workflows based on common orchestration/choreography patterns but modify them for LLMs with RAG and prompt engineering, considering quality control and fusing/aggregating strategies. Figure 3 presents a high-level design. Two levels must be addressed: at the ensemble and at the individual LLM. For example, given an ensemble, we can have parallel patterns with multiple branches and aggregation or sequence patterns in which output of a model will be used as input for another model. Within each model, we can have different parameter tuning and RAG. The agent model exists in many tools [9, 17, 41] and basic manually workflows for invoking LLMs are supported [6, 8, 12, 13]. We work on low-code and automatic coordination logic, reducing effort done by the engineer.

3.5 Challenge 4 – Quality attributes for tasks and services

3.5.1 Quality of LLM services. Currently, major LLMs are very big and support generative AIs, and it is costly to use such LLMs. Furthermore, due to their very generic features, it is hard to obtain high quality recommendations for complex matters, such as composition and decomposition of complex software. When using a group of LLMs for a task, there are many issues that must be solved:

- runtime performance and reliability: the global configuration for groups and individual services must deal with different structures of LLM invocation and aggregation. For individual LLM services, we must deal with complex parameters, e.g., related to context window, cost and reliability.
- data privacy and security: selection and invocation of services must help the developer to deal with sensitive inputs.

Given results from existing benchmarks of LLMs for software development, we start to build a quality model for LLM services based on common quality attributes. However, our challenging issue is to support quality based on the task model. Thus, it requires us to perform finer-grain evaluation based on results from LLMs. Therefore,

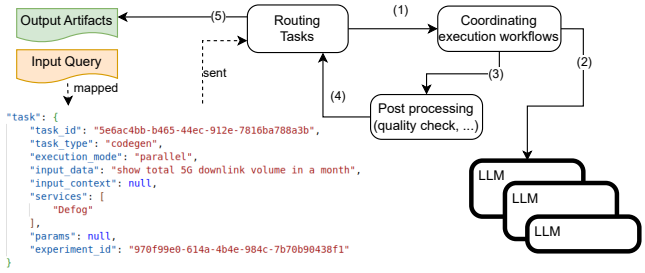

Figure 3: Task workflow at the level of LLM ensemble

novel solutions must be developed for combining incremental quality evaluation of LLM outputs, reported by task quality evaluation or developer, for aggregating and updating LLM quality.

3.5.2 Evaluating recommended outputs. Recommended outputs – solution candidates – must be evaluated. Individual tests are small tests that are carried out for certain code functions or policies achieved through the work in the previous tasks. For this, we need to invoke suitable services to test code/policies and such services are expected to get only a fragment of the code. Therefore, many smalls and lightweight services will be integrated, e.g. policy checkers, and linter services. On the other hand, if the tests are made for a large software, where the candidate solutions are just a part, this can be done via current CI/CD processes. In both cases, it is possible to obtain the errors and use them to revise the task.

3.5.3 Quality of tasks. In terms of cost and runtime, existing schedule methods can be applied. However, one of the key aspects is the reliability and explainability of answers from LLMs and other ML services. Our approach is to use quality attributes associated with ML as constraints to schedule but we need to develop local scheduler to support fusing and RAG for LLMs. Currently, key quality attributes are for data quality, ML model quality, service quality and other common attributes like costs [19, 22, 38].

3.6 Challenge 5 – Observability & Explainability

When the developer approves a result, we can record information about this approval. The traces of how the result is produced can be logged to provide knowledge for future processing. Certain information can be stored back to the Platform Knowledge for improving the knowledge. Although there are many tracing and monitoring have been developed, the end-to-end tracing and logging of cross LLMs and Platform Knowledge tasks have not been studied. We consider that they are related to challenges in task/workflow provenance, observability and explainability when using ML [18, 20, 36]. For example, the follow presents a simplified trail when calling one single LLM generating code from a set of LLMs:

```
{
    "task_instance_id": "354c6c20-66df-4390-...",
    "start_ts": 1698768167.5257006,
    "end_ts": 1698768167.8629,
    "service_name": "InCoder",
    "service_qoa": {
        "responsetime": 0.3371994800399989
    },
    "task": {
        "task_id": "30fbfed7-bd9f-4760-9fcb-...",
        "task_type": "codegen",
        "execution_mode": "parallel",
        "input_data": "def read_data_as_df ....",
        "services": ["StarCoder","CodeLlama", "InCoder"],
        "experiment_id": "473fc800-6c60-449b-..."
    }
}
```

Another challenge is about what kind of knowledge can be stored back. A simple solution is to store back features extracted from the code to be approved and link it to the requirements for supporting search. However, many other information can be stored back for future training. We distinguish two different types of update. The first is the update of accepted solutions. For example, given an approved code, we design the update consisting of (i) `experiment_id` – the id linking different tasks for the same development context, `case_info` – the description/summary of the solution, `input_query` – the original input query, (ii) `output_answer` – the result, and (iv) `ref_doc` – the reference docs which can be links to trace, test, etc. Such updates can be done via APIs or user tools and are stored into vectorized databases so that they can be used for searches in *Platform Knowledge* and for RAG-based queries to LLMs. Second, explainability is crucial, especially if the application to be modernized includes AI capabilities. Thus, using the trails of tasks leading to the result, we may focus on the explainability of the suggested solutions based on different aspects. We update the linkage between the accepted solutions and the goal of performance, accuracy, fairness and sustainability to use it for future needs.

3.7 Towards the Prototype

We are working on the initial prototype. We utilize various existing LLMs services; many from Hugging Face Hub [7]. Some of specific services are implemented as a proof-of-concept and reused from other sources. Langchain [8] and Semantic Kernel [13] are used as the base for experimental prompts and orchestration. For Platform Knowledge, we use Weaviate [14] for vectorization database. We use in-house code and other data to populate the Platform Knowledge. Deployment customization can be achieved from two perspectives: (i) the underlying Platform Knowledge and LLMs for specific deployments, (ii) the focused tasks and environments. In the first case, underlying Platform Knowledge and LLMs are selected for environment-specific deployments, such as pay-per-use LLMs can be combined with private ones and services in Platform Knowledge are customized based on the used environments. In the second case, focused tasks and contexts are customized, for example for Java development or Python development, for target big enterprises or for small enterprise use cases.

4 CONCLUSION

In this paper, we identified key issues in coordinating LLMs and other services for software development. Key issues raised from the gaps between LLMs support and the need for edge-cloud environments require the combination of LLMs with many other services to enable productive development. We outlined our design and presented our current development. The future is to focus on the development of main features discussed in this paper.

REFERENCES

[1] 2023. https://www.anyscale.com/blog/a-comprehensive-guide-for-building-rag-based-llm-applications-part-1
[2] 2023. https://coderabbit.ai/docs/introduction/
[3] 2023. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
[4] 2023. https://crfm.stanford.edu/helm/v0.2.2/
[5] 2023. https://www.microsoft.com/en-us/research/group/experimentation-platform-exp/articles/how-to-evaluate-llms-a-complete-metric-framework/
[6] 2023. Azure AI Studio. https://azure.microsoft.com/en-us/products/ai-studio/
[7] 2023. Hugging Face Hub. https://huggingface.co/docs/hub/index
[8] 2023. LangChain. https://www.langchain.com/
[9] 2023. Langchain Agents. https://python.langchain.com/docs/modules/agents/
[10] 2023. LlamaIndex. https://docs.llamaindex.ai/en/stable/index.html
[11] 2023. LMOps. https://github.com/microsoft/lmops
[12] 2023. Retool AI. https://retool.com/products/ai
[13] 2023. Semantic Kernel. https://github.com/microsoft/semantic-kernel
[14] 2023. Weaviate. https://weaviate.io/
[15] Anthropic. 2023. https://www.anthropic.com/index/evaluating-ai-systems
[16] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv:2310.11511 [cs.CL]
[17] AWS. 2023. Agents for Amazon Bedrock. https://aws.amazon.com/bedrock/agents/
[18] Larissa Chazette and Kurt Schneider. 2020. Explainability as a Non-Functional Requirement: Challenges and Recommendations. *Requir. Eng.* 25, 4 (dec 2020), 493–514. https://doi.org/10.1007/s00766-020-00333-1
[19] Khan Mohammad Habibullah, Gregory Gay, and Jennifer Horkoff. 2023. Non-Functional Requirements for Machine Learning: An Exploration of System Scope and Interest. In *Proceedings of the 1st Workshop on Software Engineering for Responsible AI* (Pittsburgh, Pennsylvania) *(SE4RAI '22)*. Association for Computing Machinery, New York, NY, USA, 29–36. https://doi.org/10.1145/3526073.3527589
[20] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A Survey on Provenance: What for? What Form? What From? *The VLDB Journal* 26, 6 (dec 2017), 881–906. https://doi.org/10.1007/s00778-017-0486-1
[21] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE]
[22] J. H. Husen, H. Washizaki, H. Tun, N. Yoshioka, Y. Fukazawa, H. Takeuchi, H. Tanaka, and K. Munakata. 2023. Extensible Modeling Framework for Reliable Machine Learning System Analysis. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*. IEEE Computer Society, Los Alamitos, CA, USA, 94–95. https://doi.org/10.1109/CAIN58948.2023.00022
[23] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770 [cs.CL]
[24] Anup K. Kalia, Jin Xiao, Chen Lin, Saurabh Sinha, John Rofrano, Maja Vukovic, and Debasish Banerjee. 2020. Mono2Micro: An AI-Based Toolchain for Evolving Monolithic Enterprise Applications to a Microservice Architecture. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1606–1610. https://doi.org/10.1145/3368089.3417933
[25] Adriano Koshiyama, Emre Kazim, and Philip Treleaven. 2022. Algorithm Auditing: Managing the Legal, Ethical, and Technological Risks of Artificial Intelligence, Machine Learning, and Associated Algorithms. *Computer* 55, 4 (2022), 40–50. https://doi.org/10.1109/MC.2021.3067225
[26] Eero Laaksonen. 2023. LLMOps: MLOps for Large Language Models. https://valohai.com/blog/llmops/
[27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.
[28] Feifei Li. 2023. Modernization of Databases in the Cloud Era: Building Databases That Run Like Legos. *Proc. VLDB Endow.* 16, 12 (aug 2023), 4140–4151. https://doi.org/10.14778/3611540.3611639
[29] Junling Liu, Chao Liu, Peilin Zhou, Qichen Ye, Dading Chong, Kang Zhou, Yueqi Xie, Yuwei Cao, Shoujin Wang, Chenyu You, and Philip S. Yu. 2023. LLMRec: Benchmarking Large Language Models on Recommendation Task. arXiv:2308.12241 [cs.IR]
[30] Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2023. ControlLLM: Augment Language Models with Tools by Searching on Graphs. arXiv:2310.17796 [cs.CV]
[31] Aman Madaan, Pranjal Aggarwal, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Mausam, and Manaal Faruqui. 2023. AutoMix: Automatically Mixing Language Models. arXiv:2310.12963 [cs.CL]
[32] Vikram Nitin, Shubhi Asthana, Baishakhi Ray, and Rahul Krishna. 2023. CARGO: AI-Guided Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 20, 12 pages. https://doi.org/10.1145/3551349.3556960
[33] I. Ozkaya. 2023. Can Architecture Knowledge Guide Software Development With Generative AI? *IEEE Software* 40, 05 (sep 2023), 4–8. https://doi.org/10.1109/MS.2023.3306641
[34] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]
[35] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2023. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. arXiv:2302.06527 [cs.SE]
[36] Shreya Shankar and Aditya G. Parameswaran. 2022. Towards Observability for Production Machine Learning Pipelines. *Proc. VLDB Endow.* 15, 13 (sep 2022), 4015–4022. https://doi.org/10.14778/3565838.3565853
[37] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. arXiv:2303.17580 [cs.CL]
[38] Hong-Linh Truong and Tri-Minh Nguyen. 2021. QoA4ML - A Framework for Supporting Contracts in Machine Learning Services. In *2021 IEEE International Conference on Web Services (ICWS)*. 465–475. https://doi.org/10.1109/ICWS53863.2021.00066
[39] Kaspar von Grünberg. 2021. What is an Internal Developer Platform. https://humanitec.com/blog/what-is-an-internal-developer-platform
[40] Daniele Wolfart, Wesley K. G. Assunção, Ivonei F. da Silva, Diogo C. P. Domingos, Ederson Schmeing, Guilherme L. Donin Villaca, and Diogo do N. Paza. 2021. Modernizing Legacy Systems with Microservices: A Roadmap. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '21)*. Association for Computing Machinery, New York, NY, USA, 149–159. https://doi.org/10.1145/3463274.3463334
[41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629* (2022).