Proceedings of the International Conference on Industrial Engineering and Operations Management

Publisher: IEOM Society International, USA DOI: 0.46254/NA10.20250216
Published: June 17, 2025

Development of Agentic Workflows with LangGraph for Software Development Life Cycle Automation

Shriraj Mandulapalli and Emilio Hernandez
Department of Computer Science
Florida Polytechnic University
Lakeland, FL, USA
smandulapalli3528@floridapoly.edu, ehernandez2128@floridapoly.edu

Wayne Jordan
Department of Computer Science
Florida Polytechnic University
Lakeland, FL, USA
whall3063@floridapoly.edu

Alireza Chakeri
Department of Computer Science and Engineering
University of South Florida
Tampa, FL, USA
achakeri16@gmail.com

Luis G Jaimes
Department of Computer Science
Florida Polytechnic University
Lakeland, FL, USA
ljaimes@floridapoly.edu

Abstract

This paper explores the development of agentic artificial intelligence workflows utilizing the open-source LangGraph framework. With the adoption of Large Language Models (LLMs) for various applications like text generation, summarization, image analysis, and code generation, many industries are looking for ways to utilize the power of these models to automate necessary tasks efficiently. LangGraph enables the creation of these multi-agentic workflows by providing an elaborate method for controlling agent interactions and execution flow. We present a methodology for software development lifecycle (SDLC) automation by leveraging LangGraph and discuss how agentic workflows can enhance efficiency and scalability through AI-driven automation.

Keywords

AI Agents, Large Language Models, LangGraph, Software Development Life Cycle.

1. Introduction

Artificial Intelligence (AI) has advanced rapidly in recent years. The conception of the Artificial Neural Network (ANN) has allowed incredible innovations that provide the ability to perform extremely well in computer vision, natural language processing, and other machine learning tasks. One example is the Multi-Modal Large Language Model (MM-LLM), an architecture capable of handling many different input modes and performing a wide range of tasks (Zhang et al. 2024).LLMs fall under the generative artificial intelligence topic and are based on stacking the transformer architecture, a contemporary model famous for its self-attention mechanisms, with input and output parsing to ensure appropriate queries and responses. They are meant to excel at text generation and are trained with large amounts of text to ensure relevant and interpretable responses (Vaswani et al, 2017). These models are at the core of creating agents.

With this creation, AI agents have swept through the tech world, redefining innovation and automation at a rapid pace. Even non-tech companies are attempting to utilize AI Agents within their workflow. With this surge of agents bringing in a new era of intelligent automation, multiple frameworks have been developed to assist with the creation of AI agents. LangChain is a framework designed to build applications driven by LLMs. It is a widely used framework for autonomous agent creation due to its Python compatibility and extensive documentation (Topsakal and Akinci 2023). To further enhance the design of complex agentic applications, a library built on top of LangChain, called LangGraph, enables the development of scalable agentic applications and the creation of expressive, customizable agent workflows through the initialization of cyclic graphs. The library also has an essential debugging tool called LangSmith, which provides a log of the events within each stage of the graph and important metrics (LangChain 2025a).

The Software Development Life Cycle (SDLC) is the process by which many modern software applications are developed. Through a continuous cycle of six stages, including planning, analysis, design, implementation, testing, and maintenance, software products can be developed, revised, and maintained. Throughout the process, several teams specialized in different areas perform specific tasks at each development phase. However, due to the vast number of people and processes involved in the traditional SDLC, developing software can be complex and time-consuming. In this paper, we elaborate on how multi-agent workflows can reduce the time and resources needed for SDLC by automating the portions of this process. profiles, and dynamically adapting educational content. By comparing traditional rule-based systems, instructor-led customization techniques, and the latest AI-enhanced methods, this survey provides a comprehensive overview of the current landscape, highlighting the opportunities, challenges, and future directions for personalized learning in STEM.

2. Literature Review

Recent advancements in LLM implementation frameworks such as LangChain and LangGraph have significantly impacted AI agent systems and their applications. LangChain provides abstractions, making the chaining and orchestration of multiple language model tasks easier, while LangGraph extends these capabilities through advanced graph-based workflows. LangChain is a framework meant to build single agents with the normal components implement an agent. More precisely, a single agent has some form of reasoning, memory, feedback, prompting, and tool-use provided to an LLM in order to perform a specific task . 2025). LangGraph specifically allows for flexible state representation and execution logic by compiling agents into structured workflows composed of nodes and edges, creating an efficient model for multi-agent collaboration. This has allowed the enhancement of many machine learning tasks, like machine translation (Wang and Duan 2024).

Employing multiple AI agents enhances task efficiency by allowing agents to focus on specific aspects of complex problems. The utilization of specialized prompts for agents ensures clarity in their roles and responsibilities, enabling more accurate and efficient problem-solving capabilities within graph-based structures. This, along with other features of LangGraph, allows for decomposition of complex tasks in order to achieve goals, improving generalization while retaining adaptability of a system 2024). Furthermore, state creation within LangGraph provides the necessary scaffolding for tracking task progression and decision-making processes, supporting dynamic agent interactions within graph workflows (LangChain 2025).

An essential aspect of LangGraph is its support for graph creation and workflow management, involving clearly defined nodes and edges. Nodes represent individual agents or tasks, while edges denote data flow and task dependencies, enabling structured collaboration among multiple agents. Compilation processes within LangGraph translate these logical structures into executable workflows, significantly enhancing modularity and clarity in agent

interactions. Persistent memory and checkpointing mechanisms further empower these graph structures, providing resilience, state persistence, and efficient recovery mechanisms essential for robust long-term workflows (Mishra and Dubey 2013).

Applying these capabilities to automate the SDLC demonstrates significant potential. Some difficulties of the SDLC include slow development cycles and inflexible models (Lazovsky et al, 2024). These challenges can be easily resolved by agentic workflow implementation to automate sections of the SDLC that are bottlenecks, or even the entire process for a desired model. One example is how a recent study emphasized automated bug fixing leveraging LangGraph-based agents, highlighting the feasibility and efficiency of AI-driven software maintenance tasks (Wang and Duan, 2025). Architecturally, the integration of LLMs into software engineering has been explored through various model architectures, including encoder-only, decoder-only, and encoder-decoder structures, each showing distinct advantages for specific coding tasks. A systematic review of existing literature shows the substantial potential of LLMs to enhance software development efficiency, automate coding activities, and improve overall code quality (Hou et al. 2024).

This paper proposes utilizing LangGraph to orchestrate a multi-agentic workflow, where each step is tasked with autonomously executing different phases of the SDLC. By combining SDLC with an agentic workflow, we aim to increase efficiency while exploring different architectures and their complexities. The integration of LangChain and LangGraph represents a robust step forward in the utilization of AI agents in complex workflows. The ability to orchestrate multiple specialized agents, combined with flexible state management, persistent memory, and comprehensive graph workflows, significantly advances the capabilities and applications of AI in software engineering. Continued research and development in this area hold the promise of further enhancing automation, accuracy, and efficiency within a diverse set of industries where implementation is relied on as a solution.

3. System Design

LangChain is an open-source software library designed to provide developers the ability to incorporate LLMs into AI applications. It provides a variety of components, such as prompts, memory, chains, and agents, which allows the creation of efficient and complex AI systems. It provides seamless compatibility and integration for any model from leading providers such as OpenAI, Anthropic, DeepSeek, Grok, Nvidia, Mistral, and more. Additionally, LangChain enables developers to define a system prompt, which is a structured set of instructions that affects the agent's behavior and responses, ensuring the agent focuses on the task (Lazovsky et al, 2024). Furthermore, LangChain also provides the agent memory mechanisms, which allow the agent to recall information from the previous interactions. With all these components tied together, LangChain serves as an essential library for the creation of robust, scalable, and powerful AI Agents.

This paper focuses on LangGraph, a framework built on top of LangChain that allows scalable agent orchestration. LangGraph enhances the development of scalable AI-integrated applications using a state-based graph structure, where each node functions as a primary agent or tool and each edge defines the pathway that the agent follows. A state is a data structure that represents the schema of the graph, acting as the input to all the nodes and edges (LangChain, 2025). A node is a computational function that contains the logic of an agent. This function will be executed based on the state's content. Each node receives the state as an input, performs a computation or side-effect, and finally returns an updated state. An edge connects nodes, defining an execution sequence. Edges can be linear or conditional branches if the system requires a condition to be met before moving to the next node. Simply put, nodes handle the computational workload and edges dictate the workflow. By composing multiple nodes and edges, LangGraph enables the ability to construct complex workflows that alter the state over time.

By combining both libraries together, we can develop sophisticated and complex AI agents while orchestrating their communication and interactions to automate the SDLC. LangChain facilitates the development of multiple agents, each specializing in executing a specific task through the use of prompt engineering, a tailored system prompt, and efficient LLM integration. Leveraging LangGraph, we can construct an agentic workflow where each agent and function represents a node and the connections between them are configured through edges, allowing us to orchestrate complex and effective agentic workflows.

By using LangChain and an LLM, we can develop AI agents specializing in specific parts of the SDLC using refined system prompts. Utilizing prompt engineering, which involves crafting precise instructions to elicit specific responses

from the LLM, we can fine-tune an agent's behavior to produce accurate and relevant results. Each agent will contain a specialized system prompt that defines its role in the SDLC. These can represent key positions that partake in the SDLC, such as project managers, business analysts, software architects, user interface and user experience designers, quality assurance engineers, and software developers. By distributing these critical and unique roles to multiple agents, we create a scalable and modular approach to SDLC automation. We can utilize LangGraph to orchestrate a workflow where each agent functions as a node in a directed graph. This architecture allows seamless collaboration between the agents, crafting an intelligent workflow for automating the SDLC.

With LangGraph, the process begins by initializing a state, which is a shared data structure that represents the current snapshot of the system. This state consists of the graph's schema, which serves as the input and the output of the entire graph. Once the state is defined, we instantiate the graph and store it in a desired variable. The next steps will be to add nodes and edges based on the type of SDLC model. LangGraph has predefined nodes declared as START and END, which can be used to define the starting and ending nodes through an edge. After a graph with nodes and edges is constructed, the next important step is to compile the graph. The graph must go through compilation, which is a simple yet critical step that provides a couple of basic checks. This process ensures no orphaned nodes or empty edges exist, maintaining an acceptable graph structure. The graph must compile before it is ready for invocation, which can be done by calling the compile method. Inside this method, we can specify specific runtime arguments that further enhance the power of the graph and refine execution behavior. If the compilation is a success, then an agentic workflow was successfully created.

The compile method accepts various runtime arguments that empower the functionality of the graph. One of the most powerful arguments is checkpointer, which creates a built-in layer of persistence. The checkpointer argument saves a checkpoint of the graph's state at every super-step, which represents a single iteration over the graph nodes. These checkpoints are saved to a thread that is initialized in the configuration of the graph and can be accessed after the graph execution. Threads provide access to many powerful and essential capabilities such as memory retention, fault tolerance, and human-in-the-loop interactions. Each thread functions as a unique identifier assigned to each checkpoint saved by a checkpointer. By incorporating the checkpointer argument in our compile method, we can attach a configuration when invoking the graph for the first time.

The SDLC has multiple models that provide a structured methodology for software development. Common SDLC models include waterfall, V-shaped, incremental, and spiral. Each of these models enhances the software development process systematically and ensure completion of a project and high-quality software (Madupati, 2024). Although each model differs in execution, a common attribute of all these models is that they follow some variation of an iterative or sequential approach, providing a sequence of activities for software and system engineers to follow and guiding them through a robust and efficient software development process. Despite the differences in execution, all SDLC models generally consist of the following key phases (Madupati, 2024).

1. Requirement Analysis - Understanding the problem by gathering software requirements.
2. Design - Creating a structured plan for a software solution, consisting of architectural and system designs.
3. Implementation - Developing the software according to the requirements and design specifications.
4. Testing - Verifying and validating the software.
5. Deployment - Releasing the final product to users or environments.
6. Maintenance - Updating, improving, and monitoring the software after deployment.

Since the SDLC phases are either strictly sequential or iterative structure, we can conceptualize each phase as a step or a node within a graph. By treating the phases as nodes, we can define edges between them to replicate an SDLC model in a way that LangGraph can process. If each node is an AI agent that excels in a specific phase of the life cycle, then we can orchestrate an agentic workflow by defining edges to dictate the transition from one phase to another. This approach introduces automation for the process of the SDLC. The overall state of the graph will serve as a dynamic data structure, containing information that each AI agent can access and build upon. LangGraph allows us to automate both iterative and sequential models while ensuring a robust and well-structured development process.

3.1 Waterfall SDLC Model

The Waterfall SDLC model follows a sequential structure, making it perfect for replication within LangGraph. Each phase will be represented as a node, with pre-defined START and END nodes marking the graph's entry and completion, as illustrated in Figure 1.

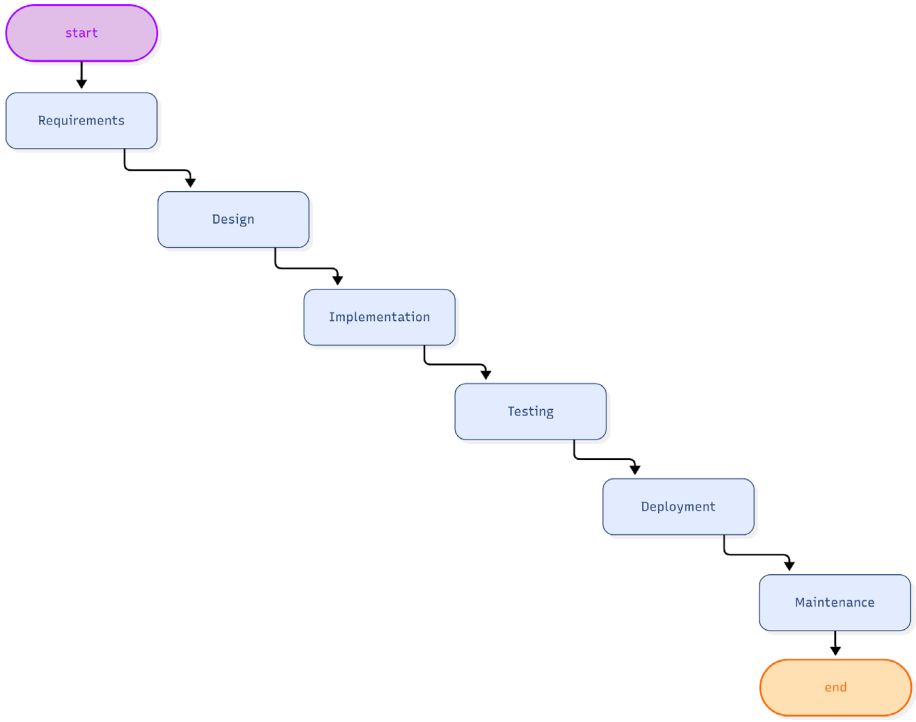

Figure 1. LangGraph Representation of the Waterfall SDLC Model

3.2 Incremental SDLC Model

The Incremental SDLC model follows a sequential structure but introduces extra complexities by building upon previous implementations. Due to the nature of this model, an initial build will go through the life cycle, after which subsequent builds expand upon the previous ones. LangGraph can facilitate this because of its persistent memory, retaining the context of the previous build across different iterations, then making adjustments and enhancements based on the context. To accurately represent the workflow, a conditional edge, represented by the dotted lines, is required to complete this graph. LangGraph has a built-in conditional edge method, which provides the ability to give an edge a specific condition. In this case, we can proceed to Build 2 once Build 1 is complete, and it provides the implementation results. The feedback from the implementation phase will be used to show the requirements phase that we can proceed to the next build. This iterative process can occur n number of times until we reach the end node, as illustrated in Figure 2.

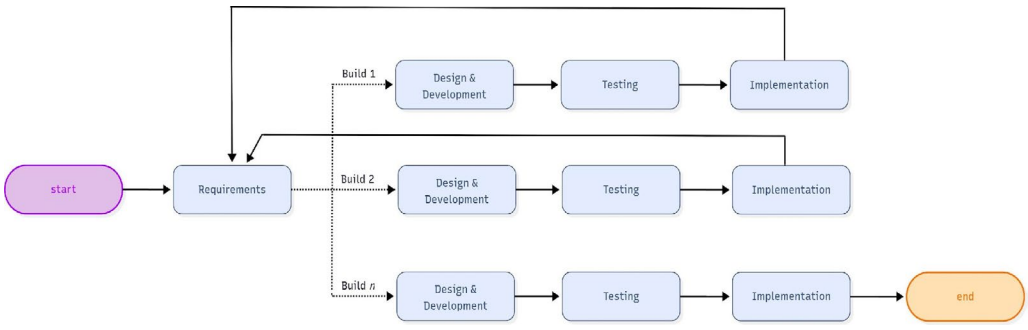

Figure 2. LangGraph Representation of the Incremental SDLC Model

3.3 Spiral SDLC Model

The Spiral SDLC model is inherently iterative by nature, which creates additional complexities when replicating it in LangGraph. To implement iteration, an edge can be defined from the last node back to the starting phase, which will create a loop. To ensure that we can successfully exit out of this loop, a conditional edge, represented by the dotted arrows in Figure 3, must be created and passed some condition, such as a simple counter function that tracks the number of iterations. If the number of iterations has been met, the graph will transition to the END node; otherwise, it will cycle back to the first phase node in the spiral. This is a common approach that demonstrates how LangGraph allows cyclic graph orchestration.

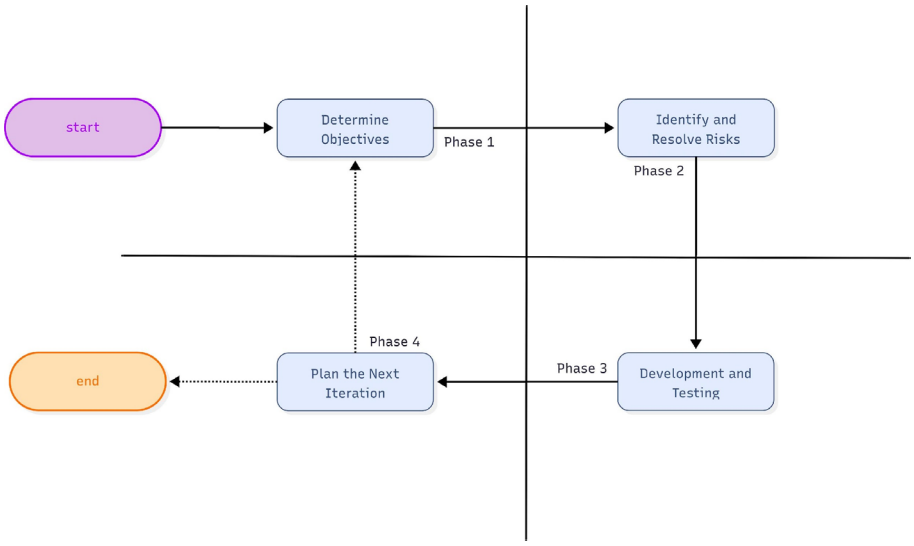

Figure 3. LangGraph Representation of the Spiral SDLC Model

In the real world, many preliminary activities occur before the first phase in the SDLC. Teams may engage in brainstorming sessions, conduct research, or rapidly develop a prototype before going into the requirements phase. All these variations display that strict adherence to an SDLC model isn't always practical. LangGraph excels in these unstable and dynamic environments due to its flexibility in adapting to different workflows in real time. Any change in SDLC can be integrated into the workflow, whether it is an agent who brainstorms, conducts deep research, or starts with prototyping. LangGraph allows for seamless modifications to the SDLC while maintaining a structured workflow.

By automating the SDLC, developers can primarily focus on the actual implementation. These agentic AI workflows can manage requirement gathering, design specifications, and architectural planning, ensuring teams have a structured basis before starting software development. This automatic workflow enhances the development process by allowing developers to leverage the generated documents, insights, and prototypes to further refine and accelerate development. This workflow would also improve efficiency and allow for faster project delivery. With the ability to orchestrate complicated workflows, LangGraph can transform modern software engineering by providing a method to automate the SDLC.

4. Discussion

This paper presents three flowchart graphs that illustrate LangGraph representations of SDLC models. Figure 1 depicts the Waterfall SDLC model when implemented using LangGraph. Each phase of the SDLC represents an AI agent that specializes in that domain. The predefined START and END nodes, colored in purple and orange, indicate where the graph starts executing and when it will terminate. This type of model is sequential by nature, beginning with the Requirements phase and moving sequentially to the Design, Implementation, Testing, Deployment, and finally Maintenance. Once the Maintenance phase is over, the next node is END, so the execution concludes. Figure 2 illustrates a LangGraph representation of the Incremental SDLC model. Unlike the linear Waterfall approach, this model incorporates iterative development cycles due to the varying number of builds. The process starts at the

Requirements phase, which is then followed by a conditional edge, detailed by the dotted arrows, that determines which build to proceed with. The workflow will start with build 1 as the base case and continue sequentially to the Design and Development, Testing, and Implementation phases. After this, the content from the implementation phase will be transferred back to the Requirements phase, where the contents will be integrated into the next build. This process iterates until the final build (build n), which will terminate at the END node.

Figure 3 depicts a Spiral SDLC model when implemented using LangGraph. While the flowchart appears cubic, the iterative nature of the workflow creates the cyclic effect of a Spiral model. The process starts at the Determine Objectives phase and sequentially advances to Identify and Resolve Risks, Development and Testing, and Plan the Next Iteration. A critical component of this workflow lies in Phase 4, where a conditional edge dictates the orchestration of the workflow. This condition allows the model to terminate the workflow by moving to the END node when the condition is met or loop back to the initial phase, repeating the process again. This structure mirrors the real behavior of the Spiral SDLC model by leveraging LangGraph's cyclic behavior.
These graphical representations serve as conceptual models to display LangGraph's capability in automating the SDLC using an agentic workflow. Instead of focusing on empirical evaluation, this study focuses on the orchestration of agentic workflows. These figures depict a structural visualization of how the SDLC can be automated using LangGraph.

LangGraph facilitates the integration of subgraphs, which can enhance the efficiency of a desired workflow (LangChain, 2025). One proposed improvement involves converting a singular node in the original graph into a subgraph containing a separate agentic workflow. This approach can improve the modularity, efficiency, and complexity of a phase in an SDLC model. A conceptual example of this enhancement can be applied to the Implementation phase of any SDLC model. The implementation phase could be represented as a subgraph, enabling the integration of an entirely separate agentic workflow, opening the possibility of enhanced results. Within this subgraph, the implementation phase can be broken down into multiple agents, each specializing in distinct areas such as frontend development, backend development, and security integration. With all three of these AI agents collaborating together, the result could ap- pear much more robust and efficient than having a singular agent do all the computation. This proposed improvement further enhances multiple agents collaborating within their own individual workflows and opens up the potential for a more adaptable and scalable system architecture.

To validate each SDLC model, we based our LangGraph representation designs on a comparative study of different SDLC models in unique scenarios (Madupati, 2024). Three of their original models were systematically redesigned to incorporate LangGraph's core components such as START, END, and conditional edges, while preserving the fundamental structure of each SDLC method. These adaptations ensured that none of the original SDLC principles were altered. Each model was verified to function identically to the original, with adjustments made to provide compatibility within LangGraph.

The integration of automated SDLC workflows into software development allows developers to have an AI-based structured plan, helping them focus primarily on implementation and coding. By leveraging LangGraph's capabilities, we can facilitate robust and efficient workflow execution, providing the ability to streamline the beginning of software development. This agentic workflow aims to assist developers with optimizing software development while providing resources, requirements, and prototypes.

5. Conclusion

This study demonstrates the feasibility of using LangChain and LangGraph to orchestrate AI agents, creating an agentic workflow for automating the SDLC. Through the replication of Waterfall, Incremental, and Spiral SDLC models within LangGraph, we validated that each model preserves the fundamental structure while leveraging the efficiency of an agentic workflow. By integrating key LangChain and LangGraph components such as agents, conditional edges, node and edge creation, and persistence into the core structure of each model, this research provides a foundation for AI-assisted automation of the SDLC, where each agent specializes in a particular phase of the SDLC. Additionally, we explore different SDLC model architectures replicated with LangGraph, displaying the complexities and flexibility of each model. The representations highlight how different models would be orchestrated automatically, providing a deeper understanding of how LangGraph can be tailored to specific requirements. The unique contribution of this research is derived from demonstrating how an automatic SDLC workflow can provide streamlined project planning and resources that developers can utilize to accelerate development. By incorporating AI agents into the

SDLC, this research opens up the potential for enhancing modern software engineering methodologies, providing developers with the ability to optimize their workflow through intelligent automation.

Disclosure of Interests

The author has no competing interests to declare that are relevant to the content of this article.

References

Duan, Z., Wang, J., Exploration of llm multi-agent application implementation based on langgraph+crewai, arXiv preprint, arXiv:2411.18241, 2024.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang., H., Large language models for software engineering: A systematic review, preprint, arXiv: 2308.10620, 2024.

LangChain., How to use subgraphs. Available: https://langchain ai.github.io/langgraph/how-tos/subgraph/, accessed on March 29, 2025

Sasson Lazovsky, Gal; Raz, Tuval; Kenett, Yoed N., The art of creative inquiry—from question asking to prompt engineering. The Journal of Creative Behavior, vol. 59, no 1, p. e671, 2025.

Madupati, Bhanuprakash. AI's Impact on Traditional Software Development, arXiv preprint arXiv:2502.18476, 2025.

Mishra, Apoorva; Dubey, Deepty. A comparative study of different software development life cycle models in different scenarios. International Journal of Advance research in computer science and management studies, vol. 1, no 5. 2013.

Topsakal, Oguzhan; Akinci, Tahir Cetin. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. En International Conference on Applied Engineering and Natural Sciences. pp. 1050-1056, 2023.

Vaswani, Ashish, et al. Attention is all you need. Advances in neural information processing systems, vol. 30, 2017.

Wang, Jialin; Duan, Zhihua. Agent AI with LangGraph: A Modular Framework for Enhancing Machine Translation Using Large Language Models. arXiv preprint arXiv:2412.03801, 2024.

Wang, Jialin; Duan, Zhihua. Empirical Research on Utilizing LLM-based Agents for Automated Bug Fixing via LangGraph. 2025.

Wang, Lei, et al. A survey on large language model based autonomous agents. Frontiers of Computer Science, vol. 18, no 6, p. 186345, 2024.

Zhang, Duzhen, et al. Mm-llms: Recent advances in multimodal large language models. arXiv preprint arXiv:2401.13601, 2024.

Biographies

Shriraj Mandulapalli is an undergraduate student of Computer Science at Florida Polytechnic University with strong research interests in machine learning, deep learning, and the applications of Artificial Intelligence.

Emilio Hernandez is an undergraduate student of Computer Science at Florida Polytechnic University with strong research interests in agentic AI, deep learning, and the applications of Artificial Intelligence.

Wayne Jordan Hall is an undergraduate student of Computer Science at Florida Polytechnic University with strong research interests in agentic AI, deep learning, and the applications of Artificial Intelligence.

Dr. Alireza Chakeri is a software engineer with Google.

Dr. Luis Jaimes is an assistant professor in the Department of Computer Science at Florida Polytechnic University. Jaimes' research interest lies in intelligent mobility and crowd-sensing, pervasive and mobile computing, and cyber-physical systems for mobile Health. He is the director of the ubiquitous sensing and smart mobility lab. Jaimes research is currently supported by the National Science Foundation (NSF), and the Institute of Health Informatics at Florida Polytechnic University.