ON THE WAY TO UNIVERSAL EXECUTOR / НА ШЛЯХУ ДО УНІВЕРСАЛЬНОГО ВИКОНАВЦЯ

Svystunov A. O. / Свистунов А.О.

Київський національний університет імені Тараса Шевченка / Taras Shevchenko National University of Kyiv

Україна, 01033, місто Київ, вул. Володимирська, 60

Email: antonsvystunov@knu.ua, +380970246540

Code generation and program synthesis are more of ways of trying to replace humans in software development. Generated programs are still Turing-complete and are limited to specific software development patterns from LLMs' training set. At the same time, by using LLMs, we have a robust common sense reasoning tool that can infer knowledge and work with natural input way better than traditional approaches. Regarding the AI OS concept, LLM can be considered a processor unit for future applications. But while LLM works better with intensional means, classic Turing machines are still more reliable in tasks where determinism is crucial. The work is dedicated to the universal executor architecture concept - an agentic system that can utilize both LLM-based inferences and computing by acquiring and memorizing programs.

Large language models (LLMs) have trailblazed many research directions on AI agents and natural language processing. They are becoming a subject not only for data science but for cybernetics, a new way to control information systems. The latest studies often cover how LLMs can be utilized as controllers (ReAct [1]) or processors (AI as an OS approach [2]) to solve problems that are stated in natural language. A significant part of those studies is dedicated to the problem of code or program generation, which is considered a utility to optimize software development and a missing building block to connect new probabilistic and intensional and old deterministic and extensional ways of defining the goal of computing. At the same time, LLMs' code generation and program synthesis skills are still limited, making them non-reliable for real-world scenarios. To address this issue, the concept of a universal executor is proposed.

Program generation, in a broad sense, is not only code generation. In terms of communicative informatics, a program is a special informational object that provides a goal of processing in one of two ways: as a sequence of steps to transform inputs into outputs or as a set (partial) of requirements regarding the desired output. It is natural to consider both ways of program definition and select the appropriate performer (platform, executor) based on the nature of requirements. The universal executor would need to combine these two approaches for program definition both for the generation of programs and executor selection.

The universal executor processing pipeline starts with a *peripheral agent*, which aims to communicate with the initiator, a human, or another agentic system. UE accepts problem definition and input data that can be mixed in natural language. The challenge is that the problem definition may need to be completed. To complete the definition, the universal executor may need to look up some information system (web or internal) to fetch the missing contextual data. This task is for the *data agent,* a RAG by domain, to define the domain-level entities and their relationships.

After UE has all of the domain data (model), it has to classify the task to select the target executor by invoking the *task classifier agent*. For example, summarization or reasoning are tasks that LLMs can handle; at the same time, some dynamic programming problems may require classic computation. Classification can be done by a finite set of rules or by returning to the system's previous knowledge to find the exact or similar "programs" to apply to input data. Memory is an essential part of the UE concept. A list of verified plans and programs can drastically improve processing time and correctness. It is suggested that verified plans and programs in a search index that can be filled out independently are used. This approach may be called JIT-generation, referring to just-in-time compilation for code interpreters. For LLM-based tasks, programs can be stored a serialized agentic pipeline or as a set of tools or code snippets for compute-based. If the matched program is missing some steps, UE must generate a new one and store it in memory.

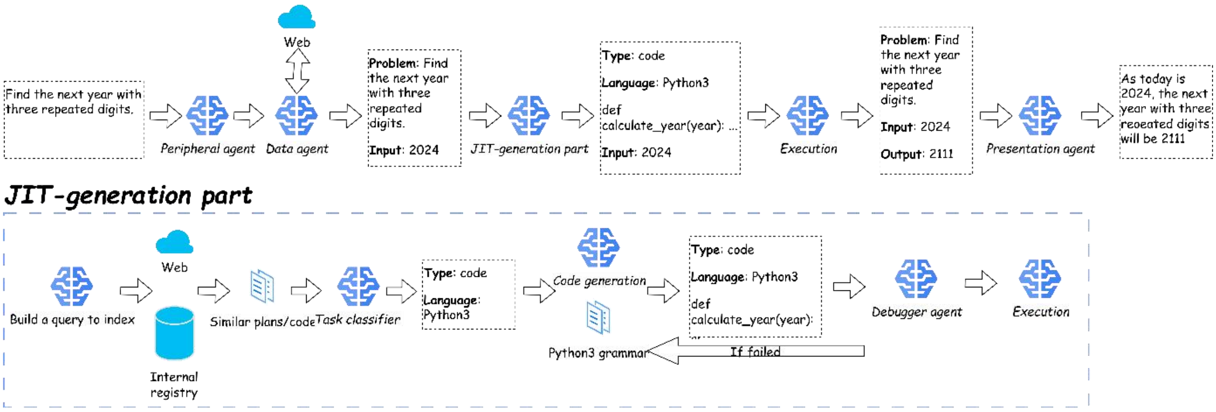

Figure 1. Overview of universal executor concept processing flow.

When generating a compute-based program (code), UE must select the target language and platform. LLMs may not be "proficient" in all of the existing programming languages. In this case, UE will need to balance model creativity and the syntactic and semantic correctness of the output program. A grammar augmentation technique can address syntactic correctness issues, for example, by using SynCode's approach [3]. By predefined context-free grammar, output logits of the language model can be masked based on the next allowed terminal symbol of the language defined by a grammar. However, this approach may lead to the model being too conservative, and it is not guaranteed that the generated sequence is a correct terminal sequence, as it guarantees only partially correct sequences, the one that has at least one continuation to be inferenced to a terminal sequence. Thus, grammar augmentation should be used to re-write a code after the plain code generation step.

After the program code is generated, it should be evaluated. The current state-of-the-art approach uses *debugger agents*, like Language Model Debugger (LDB), so that the current best performance in code generation on MBPP and HumanEval benchmarks is held by MapCoder [4] and LDB-based agentic architectures [5]. *The debugger agent* iteratively generates and autocorrects the generated code. This step may lead to autocorrection iterations, and there is no way to guarantee convergence. Code should be executed in an isolated environment with all necessary software installed - libraries, interpreters, etc.- based on the previously selected language and platform.

When generating an LLM-based program, UE should generate prompts and agents to handle the requested transformation. To make those reproducible, prompts may utilize the Plan-and-Solve zero-shot approach [6], which will allow UE to define the clear execution steps and provide more contextual and semantic value for further indexing to memory. A critique agent crew can evaluate the plan. If the result is ambiguous or non-convergent for both generation steps, processing should be terminated, and control returned to the *peripheral agent* for resolution.

The next step for both branches is executing a program on the input data provided and passing the raw outputs to the *presentation agent*. The result's presentation may fully depend on the interface in which the UE-based application is executed. It may be just a text report or a specific adaptive UX element like chat or canvas.

A crucial architectural detail is that the UE should optimize its inference by combining different types and deployment options of LMs to improve performance. If the UE already has the matching program, it performs similarly to modern tool-calling agent workflows.

Conclusion. Universal executor architecture is a proposal on how to build next-gen applications that combine classical compute-based approaches and generative AI capabilities. LLMs have opened a new chapter in programology and provided new unresolved problems on developing, executing, and verifying new types of programs. New steps will include addressing issues like context-window limitations, formal verification of UE programs, inference type, and distributed nature of UE in terms of LMs and executors. Further publications will include details on implementation and benchmark results on MBPP and HumanEval datasets.

References

1. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models.* arXiv. https://arxiv.org/abs/2210.03629
2. Mei, K., Zhu, X., Xu, W., Hua, W., Jin, M., Li, Z., Xu, S., Ye, R., Ge, Y., & Zhang, Y. (2024). *AIOS: LLM Agent Operating System*. arXiv. https://arxiv.org/abs/2403.16971

3. Ugare, S., Suresh, T., Kang, H., Misailovic, S., & Singh, G. (2024). *SynCode: LLM Generation with Grammar Augmentation*. arXiv. https://arxiv.org/abs/2403.01632
4. Islam, M. A., Ali, M. E., & Parvez, M. R. (2024). *MapCoder: Multi-Agent Code Generation for Competitive Problem Solving*. arXiv. https://arxiv.org/abs/2405.11403
5. Zhong, L., Wang, Z., & Shang, J. (2024*). Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step-by-step.* arXiv. https://arxiv.org/abs/2402.16906
6. Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., & Lim, E.-P. (2023). *Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models*. arXiv. https://arxiv.org/abs/2305.04091

МОДЕЛЬ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПЛАНУВАННЯ ТА ОЦІНКИ РИЗИКІВ ПРОЄКТІВ / ARTIFICIAL INTELLIGENCE MODEL FOR PROJECT PLANNING AND RISK ASSESSMENT

Михайлов Н. О. / Mykhailov N.O.
Київський національний університет імені Тараса Шевченка / Taras Shevchenko National University of Kyiv
03022, Київ, пр. Академіка Глушкова, 4д., Факультет комп'ютерних наук та кібернетики
E-mail: nikmikhailov13@gmail.com

Effective project management in the face of constantly changing market demands and technological advancements is essential for company success. Optimizing planning and risk management processes plays a critical role in improving outcomes and reducing implementation timelines. Traditional methods, such as the Critical Path Method and Network Programming, provide effective project structuring but often struggle to address dynamic changes and unforeseen risks. Modern technologies, particularly machine learning, enable the development of adaptive management systems capable of leveraging historical data and real-time updates. These algorithms can accurately predict task durations, assess risks, and optimize resource allocation. This article focuses on designing an adaptive model to enhance planning accuracy and minimize project risks.

Ефективне управління проєктами в умовах постійно змінюваних ринкових вимог та технологічного прогресу є необхідною складовою успішної діяльності компаній. Оптимізація процесу планування та управління ризиками відіграє ключову роль у підвищенні якості результатів і скороченні термінів реалізації. Традиційні методи планування, такі як метод критичного шляху і метод програмування в мережі, ефективно допомагають структурувати задачі проєкту. Однак, через зростаючу складність проєктів, ці методи не завжди можуть врахувати динаміку змін та непередбачені ризики. Застосування сучасних технологій, зокрема машинного навчання, дозволяє розробляти гнучкі та адаптивні системи управління, здатні враховувати як історичні дані, так і нові зміни в проєкті в режимі реального часу. Алгоритми машинного навчання можуть ефективно прогнозувати строки виконання завдань, оцінювати ризики та допомагати в оптимальному розподілі ресурсів. Дана праця зосереджена на розробці адаптивної моделі, що дозволяє покращити точність планування та мінімізувати ризики під час виконання проєктів.

На початковому етапі роботи алгоритму відбувається підключення до JIRA через API для збору даних про завдання зі спринтів. Оскільки JIRA є популярним інструментом управління проєктами, отримані дані достовірно відображають реальні робочі процеси команди. API забезпечує доступ до інформації про попередні спринти, включаючи виконавців завдань, тривалість їх виконання, початкові оцінки часу, пріоритети тощо. Автоматизоване отримання цих даних значно полегшує процес збору інформації для прогнозування, усуваючи необхідність ручного втручання [1].

Оскільки нейронні мережі працюють лише з числовими даними, категорійні дані, такі