LEVIATAN – Leveraging Open-source AI for Embedded Software Development

C. Aguet, R. Berguerand, J. Beysens, D. Hoover, J. Lahera Perez, R. Safai-Naeeni, A. Rubio

This project aims to implement a large language model (LLM) agentic solution tailored for embedded code development. By utilizing open-source, self-hosted models, it ensures the use of confidential data without privacy concerns. Additionally, an automatic pipeline has been developed to facilitate the computation of metrics, assessing the performance of the solution and enabling its benchmarking with other solutions on the market. Ultimately, this project improves embedded software development engineers' productivity by focusing their effort on the most meaningful tasks.

Dedicated models and tools, such as GitHub Copilot[1] or Codestral[2], have already been developed for large language model (LLM) code generation. However, these tools are primarily designed for high-level programming and may not meet the specific challenges of embedded programming, including hardware dependencies, low-power consumption, and low-memory footprint. This project addresses this aspect by implementing an agentic solution on top of existing LLMs to enhance embedded software generation. By integrating autonomous agents, the solution enables to independently perform tasks, make decisions, take actions, and interact with their environment without constant human intervention. Similar to HumanEval, a benchmark dataset for assessing LLMs coding skills, the first phase of this project aimed to create a comparable dataset of embedded software challenges, named EmbeddedEval. These challenges involved writing drivers (e.g., UART, Timer, GPIO) typically found in microcontrollers and specifically targeted two families of microprocessors frequently used at CSEM: Nordic nRF52 and STM32. An autonomous pipeline was then created to automate code development tasks. It processes challenge descriptions, which include function definitions specifying input parameters and intended functionality. The prompt is enhanced with common information, such as format, coding language (C, C++), and platform requirements.

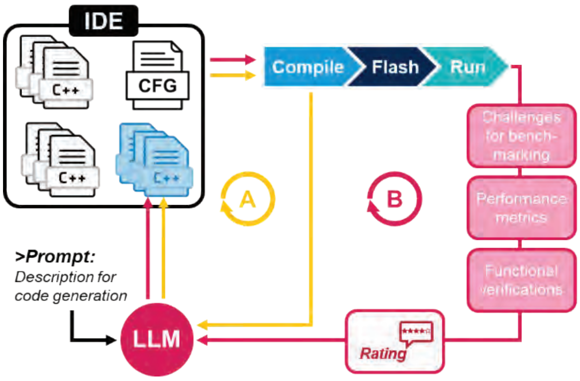

Figure 1: Automated pipeline.

To generate structured data from LLM outputs, the Pydantic[3]. library is utilized. This allows relevant data retrieval (such as the code or the number of errors) to be directly used in subsequent steps of the pipeline without the need for manual parsing. After generating the code, it is inserted into a template project for compilation. If compilation fails, the LLM parses the output to list the errors and suggest corrections; then attempts to fix them. This process is repeated until the code compiles successfully or a defined threshold is reached, marking the pipeline as failed.

If the firmware is successfully generated, it is automatically flashed onto a test board. Dedicated unit tests verify the code for correctness, with results retrieved via a serial interface. If tests fail, the output is used to regenerate the code. To enhance code generation, retrieval-augmented generation (RAG) is employed. In the first phase, semi-structured data (such as PDFs) are parsed to extract and separate the text and tables present in the documents. An LLM is then used to generate summaries for each part, facilitating better indexing and retrieval process. These summaries are processed to generate embeddings, which are stored in a database. During the pipeline's code generation phase, the RAG vector base is queried to retrieve relevant information from datasheets, which is then added to the prompt.

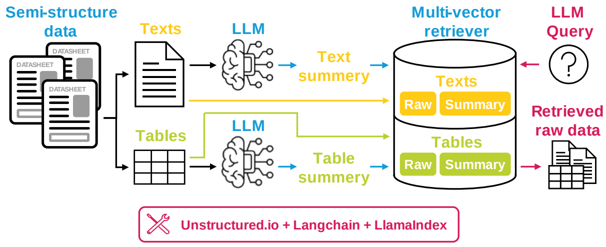

Figure 2: Retrieval-augmented generation (RAG) workflow[4] [5].

To ensure privacy when using confidential data, LLMs are hosted on a local server, preventing external companies from accessing queries. The Ollama[6] tool is used on this server to host open-source models. As a final step of this still ongoing project, the agentic solution will be benchmarked against existing solutions. A scoring formula inspired by Pass@k[7] was designed, considering the number of iterations needed by the agent and the quality of the generated code in terms of runtime latency and memory consumption.

In conclusion, LEVIATAN has laid the groundwork for advancing embedded code generation by leveraging the capabilities of LLMs. This development enabled the creation of the EmbeddedEval dataset, an evaluation framework designed to benchmark embedded code generation solutions through specific challenges. We have developed an agent capable of iterating and refining its own code outputs, utilizing a combination of tools and RAG modules, supported by relevant datasheets. The pipeline covers the full process, from code generation to compilation, flashing, testing, and rating of final outputs. Looking ahead, we envision further enhancements in the agent's autonomy, and we see opportunities for a more seamless deployment of these solution in a hosted environment.

[1] GitHub Copilot: https://github.com/features/copilot

[2] Codestral: https://mistral.ai/fr/news/codestral/

[3] Pydantic: https://docs.pydantic.dev/latest/

[4] Unstructured.io: https://unstructured.io/

[5] Langchain: https://www.langchain.com/

[6] Ollama: https://ollama.com/

[7] M. Chen, *et al*., arXiv preprint arXiv:2107.03374 (2021).