AI Wake Up Call Challenge

Joey Ah-kiow
ja4844@nyu.edu
New York University

Baleegh Ahmad
ba1283@nyu.edu
New York University

Vineet Bhat
vrb9107@nyu.edu
New York University

Luca Collini
lc4976@nyu.edu
New York University

I. Competition Proposal

This challenge aims to use generative AI to develop a hardware accelerator for Keyword Spotting (KWS) on the Caravel System-on-Chip. Generative AI large language models (LLMs) have historically generated better-quality code in high-level software languages than hardware description languages used for hardware Register-Transfer Level (RTL) design. We use this intuition to motivate the use of high-level synthesis (HLS) for this challenge. HLS is a design process that accepts an algorithmic implementation, typically in languages like C, C++, or SystemC, and generates synthesizable RTL. It also provides the ability to quickly explore the optimization design space (e.g., pipelining, loop unrolling) through pragmas. We will explore using an LLM to translate an existing KWS Machine Learning (ML) model implemented in a high-level software language into HLS-synthesizable C code and leverage an HLS tool, like Catapult, to generate Verilog. Using LLMs in this context may alleviate the need to develop time-consuming compiler-like algorithmic approaches and bridge the gap between software and hardware generation. Afterward, we will explore developing a full end-to-end design generation pipeline that will intake a natural language description of the problem (e.g., "design a keyword spotting application that uses machine learning to..."), generate high-level code that can be functionally validated and used to train the model, and then translate the high-level code into RTL. Finally, we will also investigate the development of an automated framework to minimize human interaction.

II. Soft to hard: Translating software to hardware

The first step of our solution is to translate existing high-level code, implemented in a language like C, into HLS-synthesizable code using an LLM. HLS has many design limitations that are not present in regular code that will need to be addressed. This includes dynamic memory allocation (i.e., `malloc()`, `calloc()`, `free()`, and `realloc()` function calls) and other external function calls. Furthermore, the performance of HLS-generated RTL is highly dependent on optimized input code through the use of pragmas to specify optimizations (e.g., pipelining), the removal of complex dependencies, and the use of appropriate data types (e.g., integer, fixed-point, floating-point, etc.). Once this process is completed, an HLS tool like Catapult can generate functionally correct RTL.

III. End-to-End Design Pipeline

After we have implemented and validated our translation flow on an existing solution, we will explore the use of an LLM to design a machine learning model for the challenge with edge inference in mind and complete the design space exploration to evaluate performance-cost tradeoffs. At this stage, cost is defined as model size and complexity and does not consider post-place-and-route power and area. The goal will be to create an end-to-end pipeline capable of generating an "optimal" solution for the provided functional requirements (e.g., "create a Keyword Spotting machine learning model capable of detecting a sequence of three words at 80% accuracy) and optimization targets (e.g., "you should minimize the model's size as much as possible"). The generated high-level code will then be used to train the model to detect specific keywords (e.g., "one two three") and, finally, translated into HLS-synthesizable code using the process developed in section II.

IV. (Semi-)Automated Generation Framework

Finally, we will implement a framework that will automate the evaluation of the LLM output and provide feedback to the LLM to minimize human intervention. The goal is to build on the end-to-end design pipeline described in section III to automatically detect and resolve any functional errors in the model and use HLS-synthesis metrics such as pipeline initiation interval (II) to optimize the final result.