Exploring Code Language Models for Automated HLS-based Hardware Generation: Benchmark, Infrastructure and Analysis

ASP-DAC 2025

Jiahao Gai, Hao (Mark) Chen, Zhican Wang, Hongyu Zhou, Wanru Zhao, Nicholas Lane, Hongxiang Fan

Outline

The Era of Generative AI

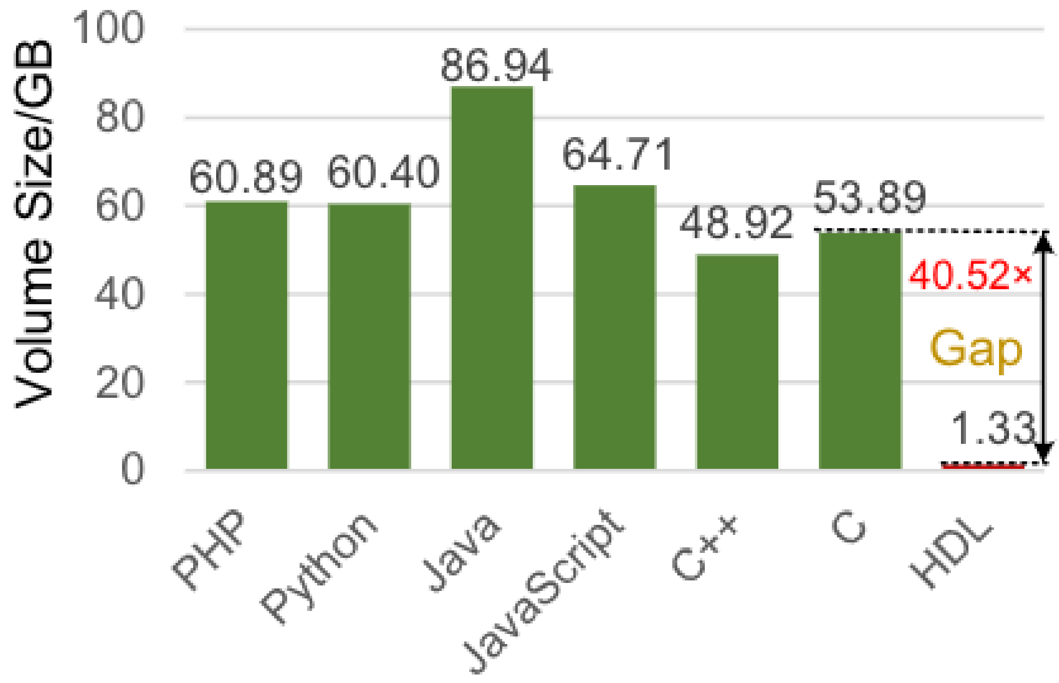
- LLM-assisted code generation: Github Copilot[1] , Deepmind’s AlphaCode[2]
- Over **50** pre-trained models and more than 170 programming language datasets released
- Automated Hardware Design Generation: Verilog, SystemVerilog

[1] Chen, Mark, et al. "Evaluating large language models trained on code." *arXiv preprint arXiv:2107.03374* (2021).

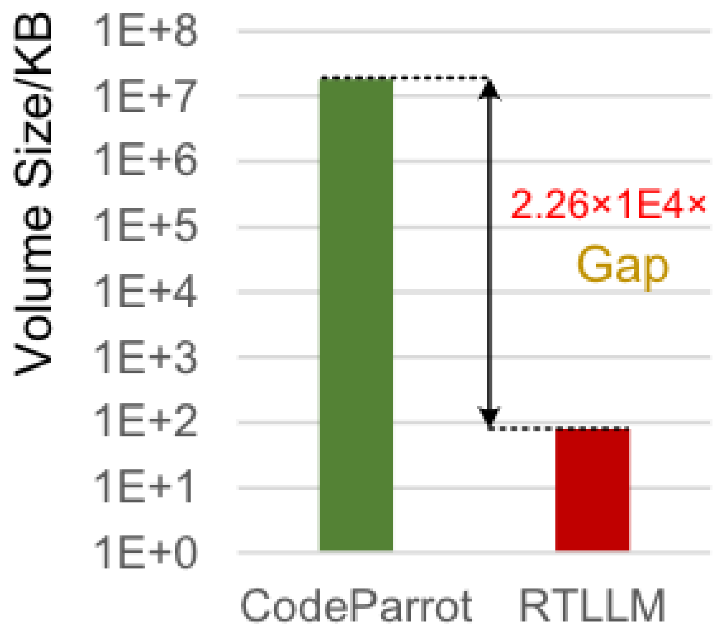
[2] Li, Yujia, et al. "Competition-level code generation with alphacode." *Science* 378.6624 (2022): 1092-1097.

Challenge 1: Data Availability of HDL

- C++ = 40.52 times HDL
- Python = 2.26 * 10^4 times HDL



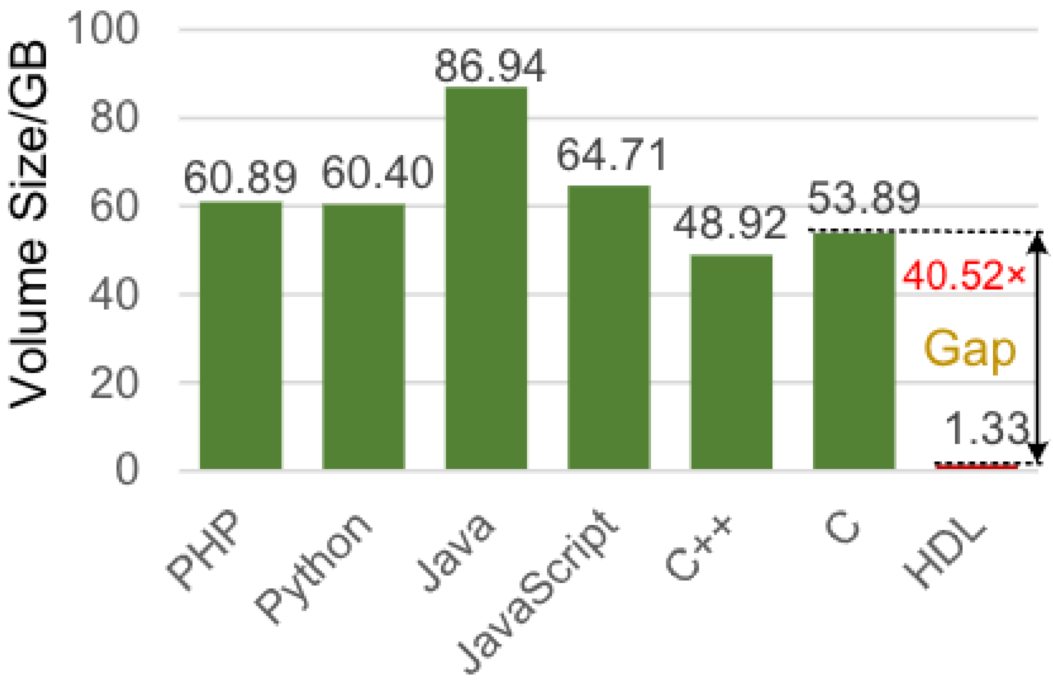
(a) Starcoder Dataset



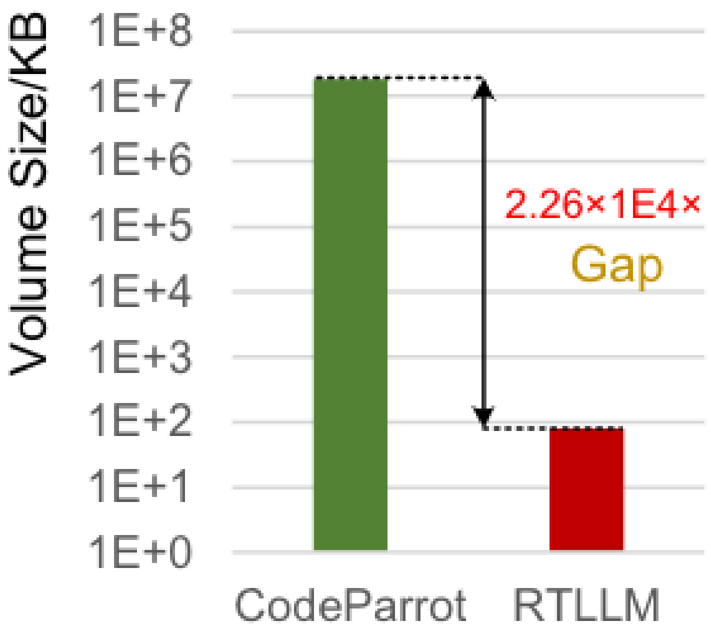
(b) CodeParrot vs RTLLM

Challenge 2: Difficulty in Transferring Pretrained Knowledge

- Most code LLMs pre-trained on software programming language
- Different from HDL



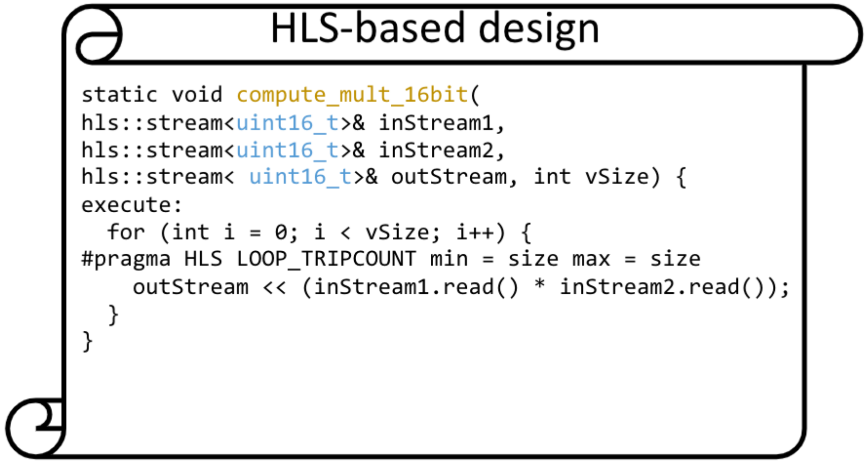
(a) Starcoder Dataset



(b) CodeParrot vs RTLLM

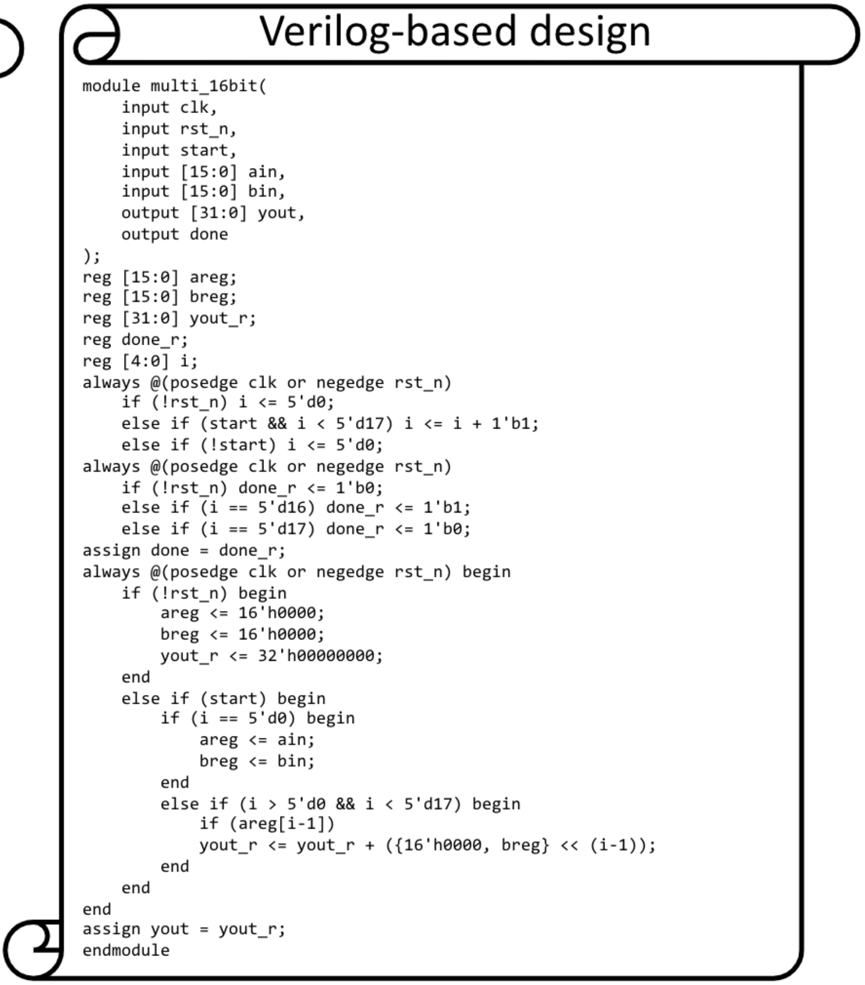
Challenge 3: Cost of Generation

- HDL implementations require 3~4 times more tokens than HLS



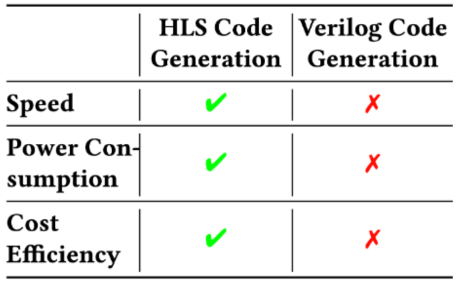
HLS-based design

```
static void compute_mult_16bit(
hls::stream<uint16_t>& inStream1,
hls::stream<uint16_t>& inStream2,
hls::stream< uint16_t>& outStream, int vSize) {
execute:
  for (int i = 0; i < vSize; i++) {
#pragma HLS LOOP_TRIPCOUNT min = size max = size
    outStream << (inStream1.read() * inStream2.read());
  }
}
```

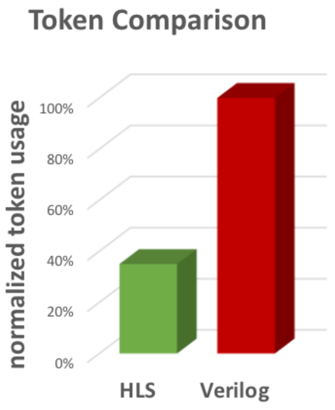


Verilog-based design

```
module multi_16bit(
    input clk,
    input rst_n,
    input start,
    input [15:0] ain,
    input [15:0] bin,
    output [31:0] yout,
    output done
);
reg [15:0] areg;
reg [15:0] breg;
reg [31:0] yout_r;
reg done_r;
reg [4:0] i;
always @(posedge clk or negedge rst_n)
    if (!rst_n) i <= 5'd0;
    else if (start && i < 5'd17) i <= i + 1'b1;
    else if (!start) i <= 5'd0;
always @(posedge clk or negedge rst_n)
    if (!rst_n) done_r <= 1'b0;
    else if (i == 5'd16) done_r <= 1'b1;
    else if (i == 5'd17) done_r <= 1'b0;
assign done = done_r;
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        areg <= 16'h0000;
        breg <= 16'h0000;
        yout_r <= 32'h00000000;
    end
    else if (start) begin
        if (i == 5'd0) begin
            areg <= ain;
            breg <= bin;
        end
        else if (i > 5'd0 && i < 5'd17) begin
            if (areg[i-1])
            yout_r <= yout_r + ({16'h0000, breg} << (i-1));
        end
    end
end
assign yout = yout_r;
endmodule
```



	HLS Code Generation	Verilog Code Generation
Speed	✔	✗
Power Consumption	✔	✗
Cost Efficiency	✔	✗



A Code LLM for HLS Generation

- Challenge 1&2: HLS shares main semantic/syntax with C/C++, which makes knowledge transfer possible and reduces dataset requirements

- Challenge 3: HLS generation is more cost-efficient at inference time

- Dataset + Model + Generation Framework

Research Questions

- Whether the existing public data is enough for the training HLS-Gen LLM?

- What performance can be achieved using existing public data?

- Can advanced techniques, such as CoT, help HLS-Gen?

Outline

Format of Dataset

- Input: Natural language description from developer
- Output: HLS design

Template of Design Point

Instruction Prompt: Specify coding language and requirements.

Design Description: High level description of the design details.

Reference Design: Canonical HLS program.

An Example of Design Point

Instruction Prompt: "Generate HLS code with the following instructions:"

Design Description: "This function performs the SYRK (symmetric rank-k) operation on matrices A and C, according to the BLAS parameters. It computes C := alphaAAT + betaC, where A is an 80x60 matrix and C is an 80x80 matrix. Designates the following function for hardware acceleration. Do not automatically pipeline the outer loop to allow for manual pipelining optimization..."

Reference Design:

```
#pragma ACCEL kernel
void kernel_syrk(double alpha,double beta,double C[80][80],double
A[80][60])
{int i;\n int j;\n int k;\n
// => C := alphaAAT + betaC. A is NxM, C is NxN
#pragma ACCEL PIPELINE ...
#pragma ACCEL TILE FACTOR...
#pragma ACCEL PARALLEL FACTOR=auto{1}
for (j = 0; j < 80; j++) {
if (j <= i) {C[i][j] += alpha * A[i][k] * A[j][k];}
}}
```

Dataset Collection

- 52 designs, 42000 HLS programs from HLSyn[3] and ML4Accel[4]

Works	#Designs
Thakur et al. [5]	17
Chip-Chat [6]	8
Chip-GPT [7]	8
RTLLM [8]	30
Ours	52

[3] https://github.com/UCLA-DM/HLSyn

[4] https://github.com/UT-LCA/ML4Accel-Dataset

[5] Thakur, Shailja, et al. "Verigen: A large language model for verilog code generation." *ACM Transactions on Design Automation of Electronic Systems* 29.3 (2024): 1-31.

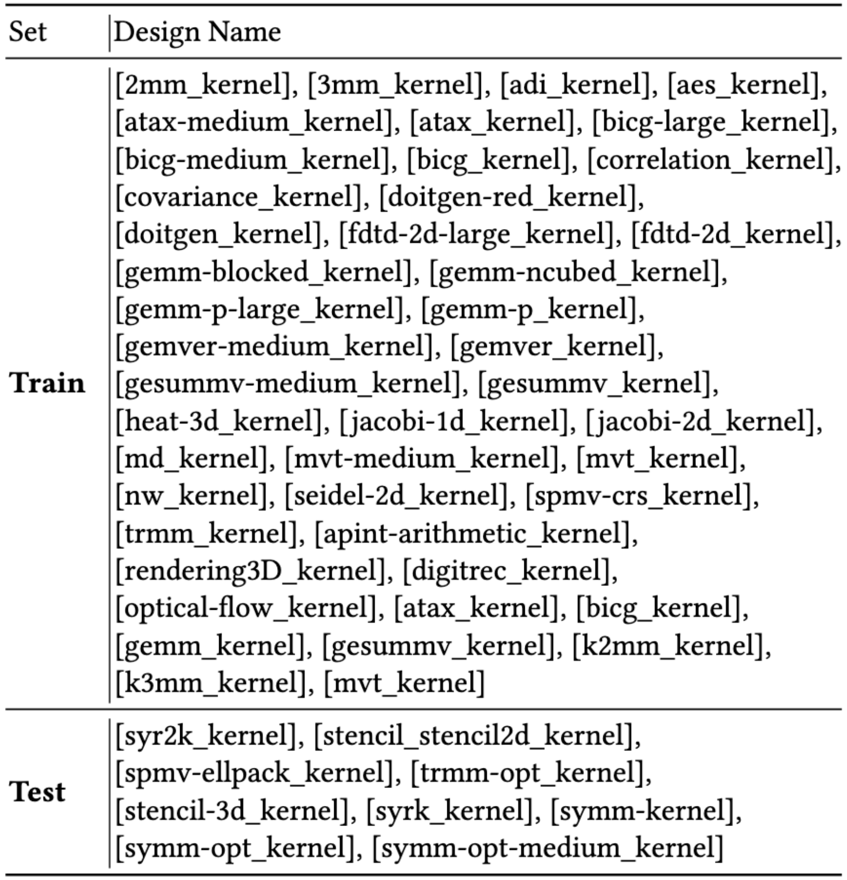
[6] Blocklove, Jason, et al. "Chip-chat: Challenges and opportunities in conversational hardware design." *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023.

[7] Chang, Kaiyan, et al. "ChipGPT: How far are we from natural language hardware design." *arXiv preprint arXiv:2305.14019* (2023).

Lu, Yao, et al.

Dataset Collection

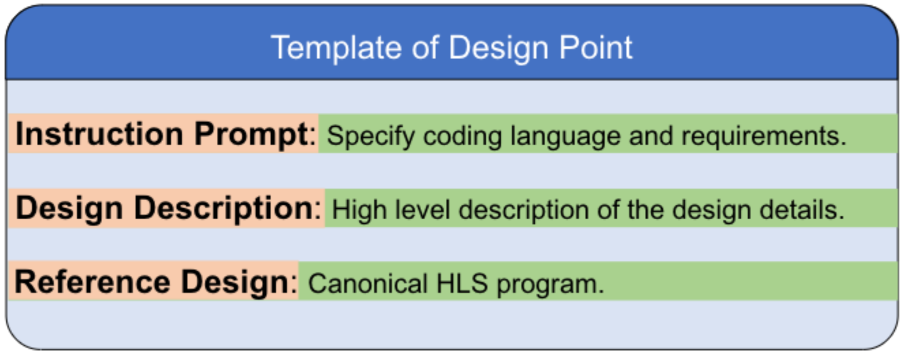
- Category:
  - Linear Algebra
  - Scientific Simulation
  - Statistical Computation
  - Iterative Methods
  - Other Computations



Set	Design Name
Train	[2mm_kernel], [3mm_kernel], [adi_kernel], [aes_kernel], [atax-medium_kernel], [atax_kernel], [bicg-large_kernel], [bicg-medium_kernel], [bicg_kernel], [correlation_kernel], [covariance_kernel], [doitgen-red_kernel], [doitgen_kernel], [fdtd-2d-large_kernel], [fdtd-2d_kernel], [gemm-blocked_kernel], [gemm-ncubed_kernel], [gemm-p-large_kernel], [gemm-p_kernel], [gemver-medium_kernel], [gemver_kernel], [gesummv-medium_kernel], [gesummv_kernel], [heat-3d_kernel], [jacobi-1d_kernel], [jacobi-2d_kernel], [md_kernel], [mvt-medium_kernel], [mvt_kernel], [nw_kernel], [seidel-2d_kernel], [spmv-crs_kernel], [trmm_kernel], [apint-arithmetic_kernel], [rendering3D_kernel], [digitrec_kernel], [optical-flow_kernel], [atax_kernel], [bicg_kernel], [gemm_kernel], [gesummv_kernel], [k2mm_kernel], [k3mm_kernel], [mvt_kernel]
Test	[syr2k_kernel], [stencil_stencil2d_kernel], [spmv-ellpack_kernel], [trmm-opt_kernel], [stencil-3d_kernel], [syrk_kernel], [symm-kernel], [symm-opt_kernel], [symm-opt-medium_kernel]

Scalable Dataset Curation Pipeline

1. Crawl HLS programs from online repo
2. Filter out invalid code samples
3. Generate Design Description using ChatGPT

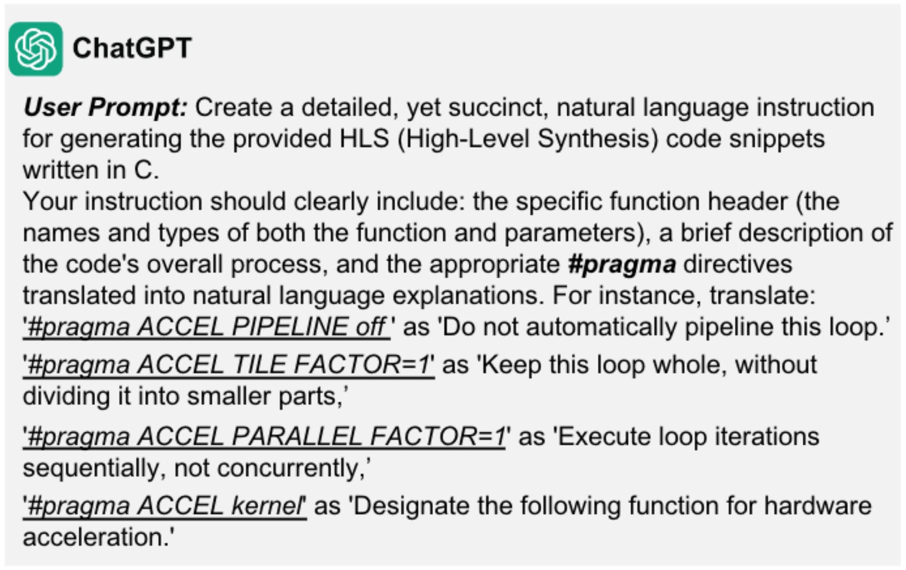


Template of Design Point

Instruction Prompt: Specify coding language and requirements.

Design Description: High level description of the design details.

Reference Design: Canonical HLS program.



ChatGPT

User Prompt: Create a detailed, yet succinct, natural language instruction for generating the provided HLS (High-Level Synthesis) code snippets written in C.
Your instruction should clearly include: the specific function header (the names and types of both the function and parameters), a brief description of the code's overall process, and the appropriate ***#pragma*** directives translated into natural language explanations. For instance, translate:
'*#pragma ACCEL PIPELINE off* ' as 'Do not automatically pipeline this loop.'

'*#pragma ACCEL TILE FACTOR=1*' as 'Keep this loop whole, without dividing it into smaller parts,'

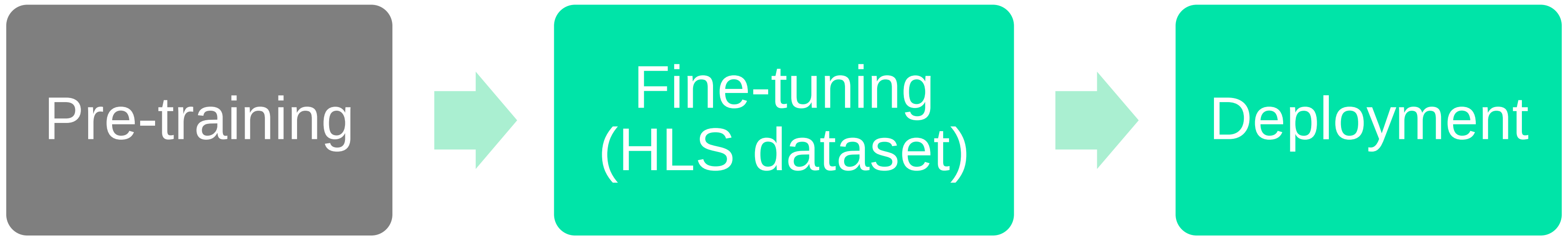
'*#pragma ACCEL PARALLEL FACTOR=1*' as 'Execute loop iterations sequentially, not concurrently,'

'*#pragma ACCEL kernel*' as 'Designate the following function for hardware acceleration.'

Outline

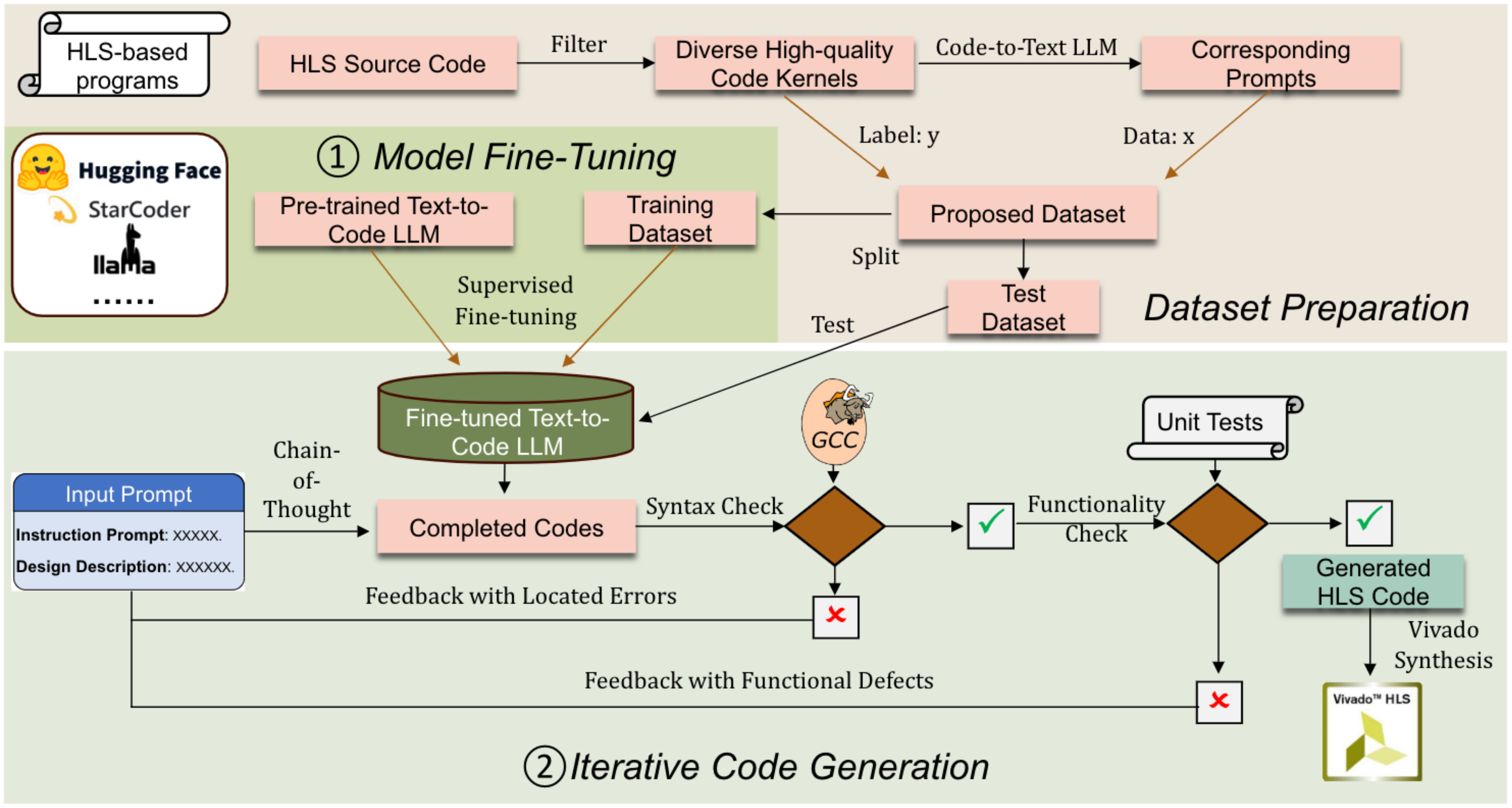
Model Training

- Leverage pre-trained code LLM (CodeLLaMA-7B)
- Parameter Efficient Fine-Tuning: QLoRA
- ~4 hours on 4 Nvidia A40s



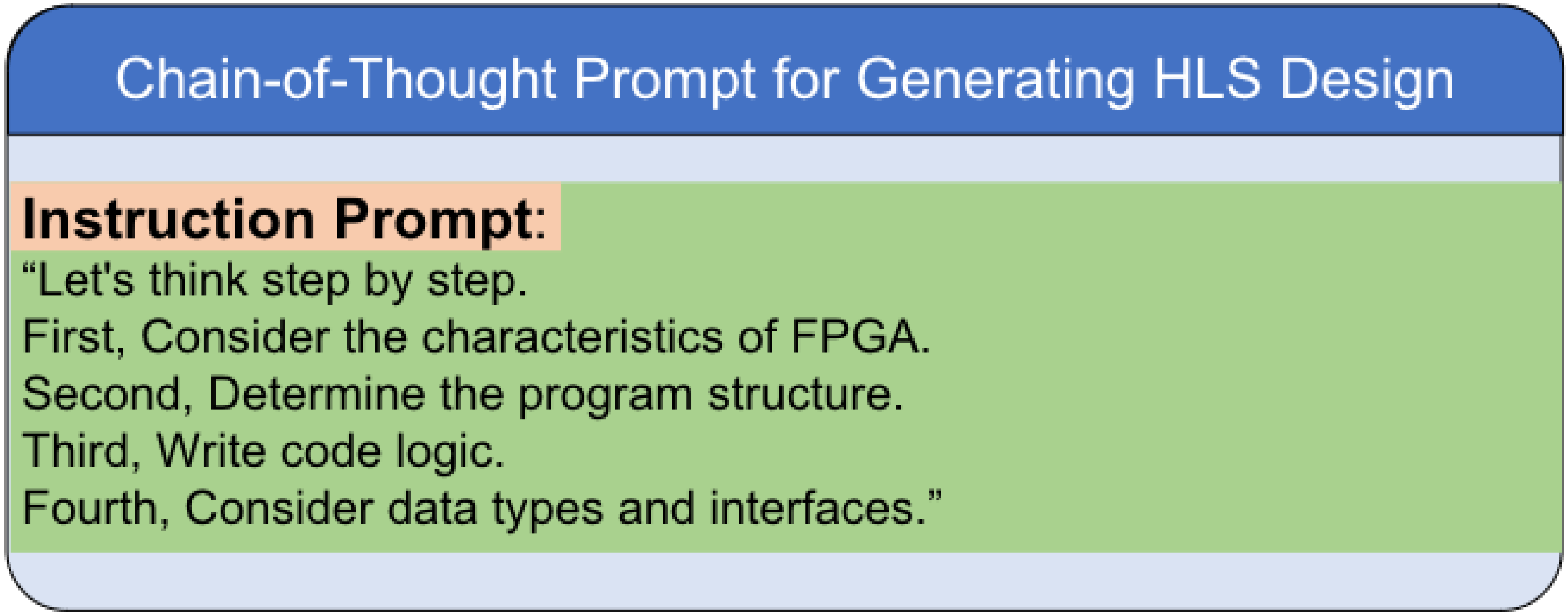
Outline

Framework Overview



Chain-of-Thought Generation

- A popular method to improve output quality

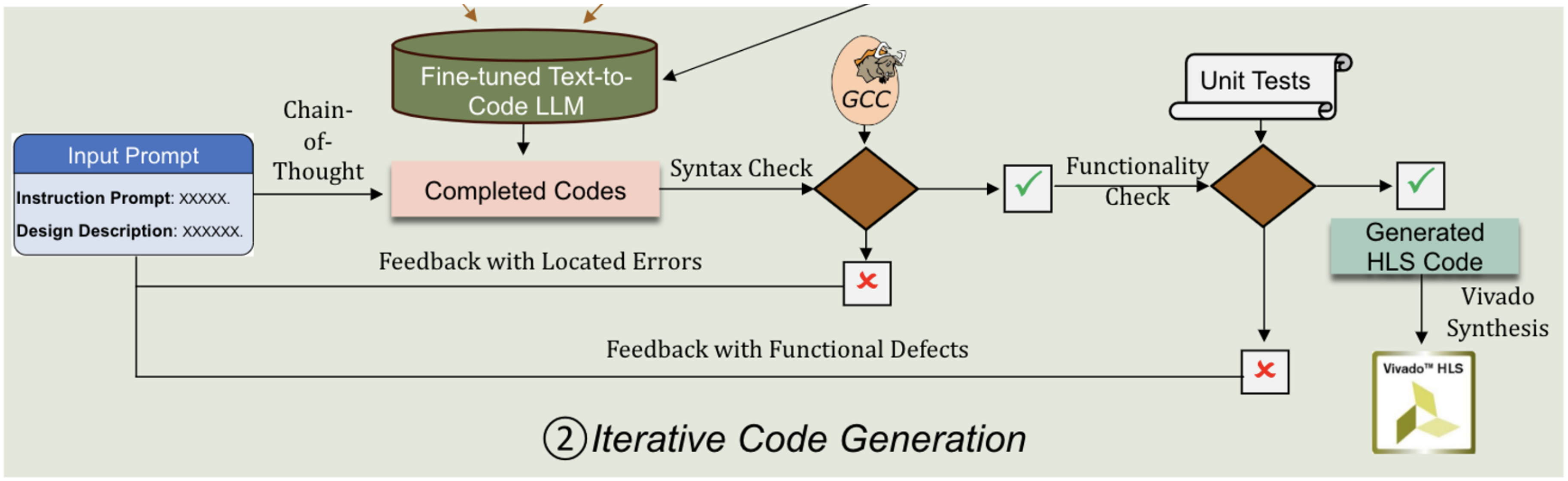


Chain-of-Thought Prompt for Generating HLS Design

Instruction Prompt:
“Let's think step by step.
First, Consider the characteristics of FPGA.
Second, Determine the program structure.
Third, Write code logic.
Fourth, Consider data types and interfaces.”

Two-Step Feedback Loop

- 1st Loop: Syntax check with gcc
- 2nd Loop: Functionality check with unit tests
- Append error information to inputs


Fine-tuned Text-to-Code LLM
Chain-of-Thought
Input Prompt
Instruction Prompt: XXXXX.
Design Description: XXXXXX.
Completed Codes
Syntax Check
GCC
Unit Tests
Functionality Check
Generated HLS Code
Vivado Synthesis
Vivado™ HLS
Feedback with Located Errors
Feedback with Functional Defects
② Iterative Code Generation

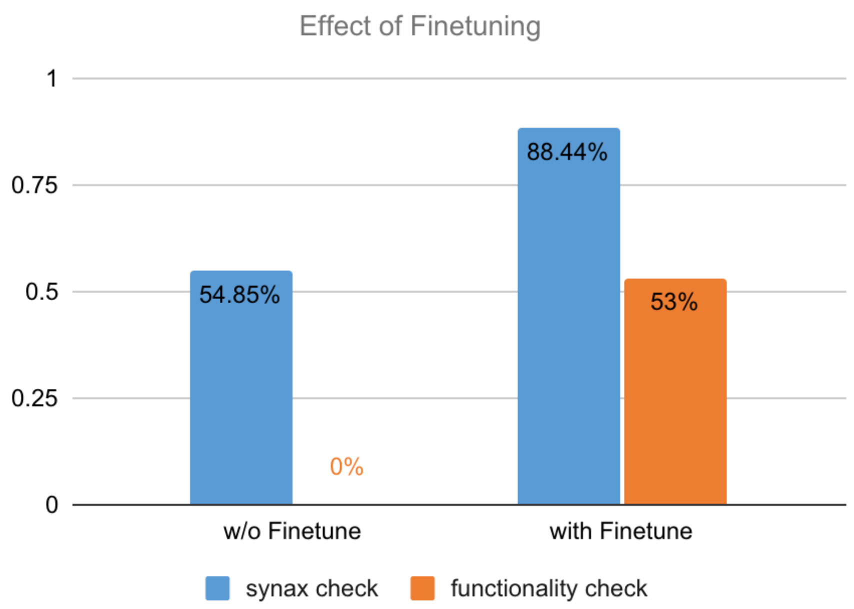
Outline

Evaluation Setup

- Training and test dataset at 4:1 ratio
- CodeLLaMA-7B with QLora
- Metrics:
  - Syntax accuracy: pass@k accuracy
  - Functionality accuracy: pass@k accuracy
  - Hardware performance

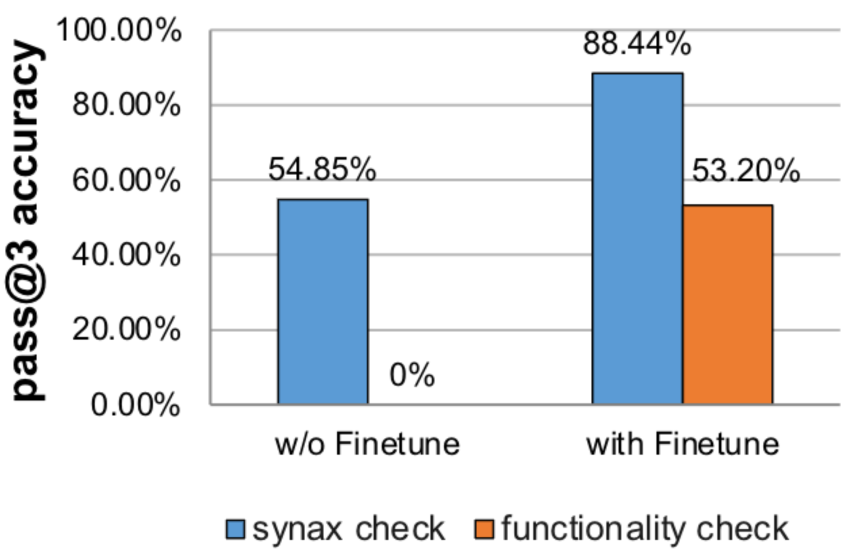
Effect of Fine-tuning

- **Key Results**:
  - Both syntax and functionality correctness improved
  - Functionality accuracy from **0%** to **53.20%**
- **Implications**:
  - HLS generation benefits from pre-training on software programming languages
  - Fine-tuning is essential for functionality correctness

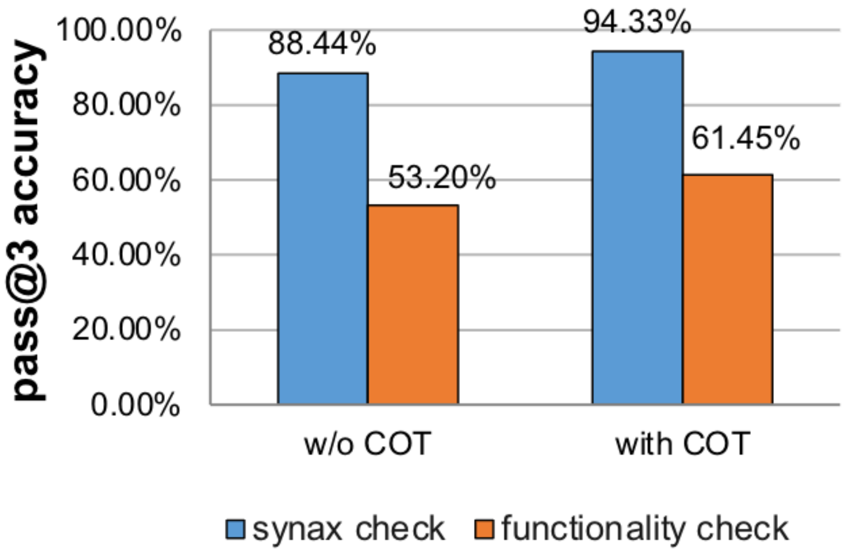


Impact of Chain-of-Thought Prompting

- Noticeable improvement in both syntax and functionality metrics.



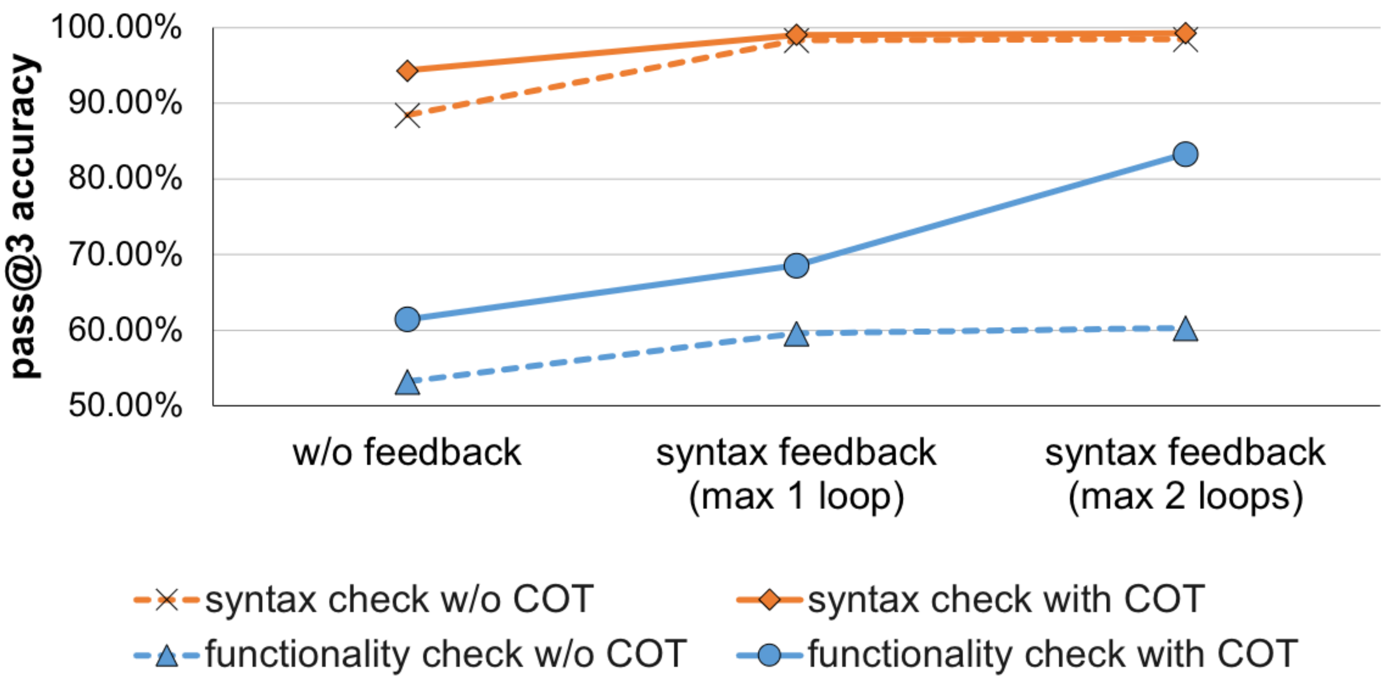
(a) Effect of fine-tuning



(b) Effect of chain-of-thought prompting

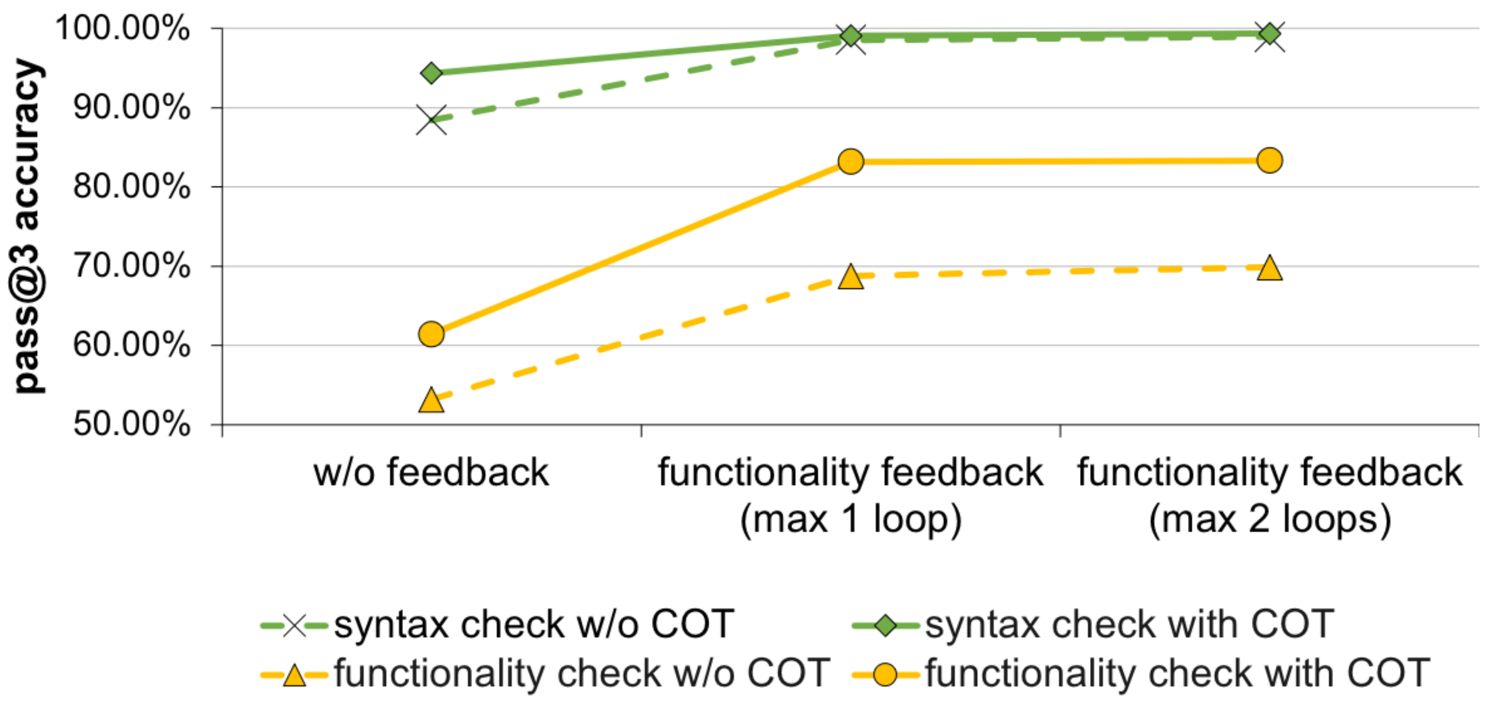
Effect of Syntax Feedback Loops

- First syntax feedback loop yields significant improvements in syntax correctness.
- Second loop shows **diminishing returns** in syntax accuracy.
- Combined with CoT, syntax feedback has a clearer impact on **functionality evaluation**, especially for complex tasks.



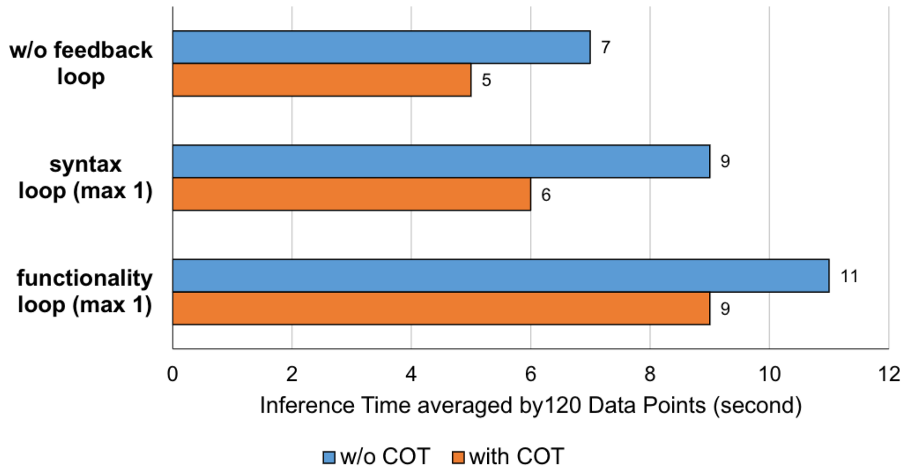
Effect of Functionality Feedback Loops

- Functionality feedback significantly improves **functionality performance**.
- Enhances both **functionality checks** and **syntax accuracy**. Indicates that better functional understanding contributes to improved syntax correctness.



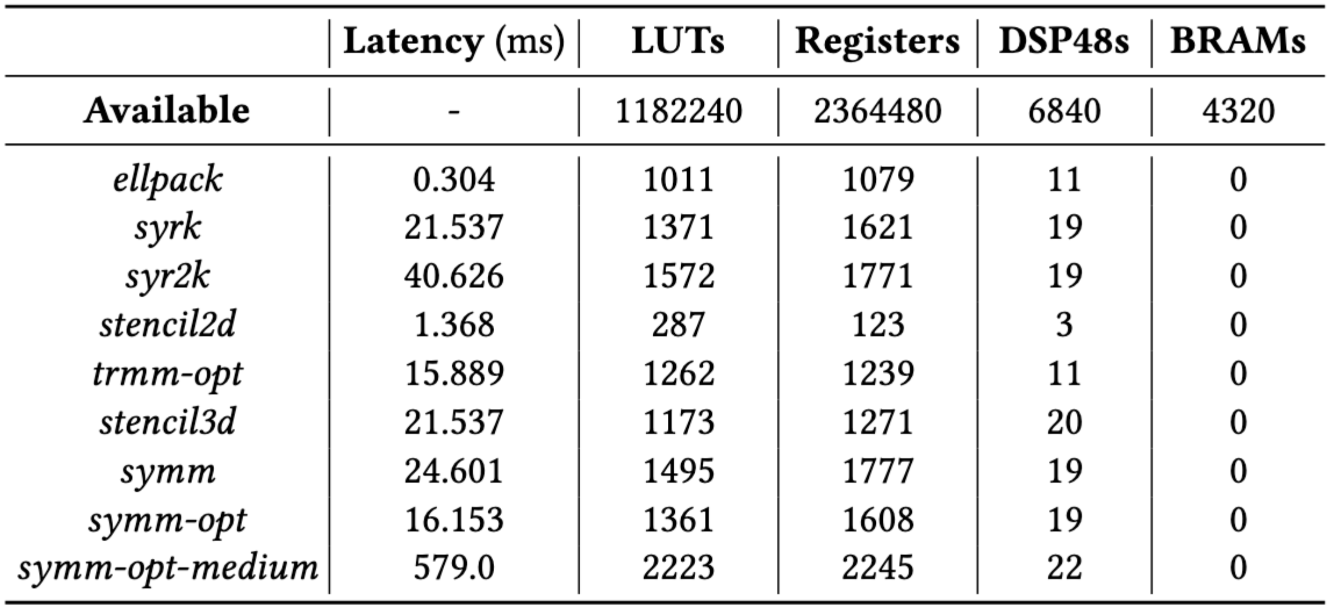
Time Cost Analysis

- Time cost for generating **120 data entries** under different conditions.
- CoT **significantly reduces** inference time.
- **With functionality feedback loop**: Most **time-consuming scenario** due to lower functionality accuracy.



Hardware Performance

- **Target Setup**:
  - Platform: **Xilinx VCU118 FPGA**
  - Clock frequency: **200 MHz**
  - Synthesis tool: **Xilinx Vivado 2020.1**
- All HLS designs show **reasonable performance**.



	Latency (ms)	**LUTs**	**Registers**	**DSP48s**	**BRAMs**
Available	-	1182240	2364480	6840	4320
ellpack	0.304	1011	1079	11	0
syrk	21.537	1371	1621	19	0
syr2k	40.626	1572	1771	19	0
stencil2d	1.368	287	123	3	0
trmm-opt	15.889	1262	1239	11	0
stencil3d	21.537	1173	1271	20	0
symm	24.601	1495	1777	19	0
symm-opt	16.153	1361	1608	19	0
symm-opt-medium	579.0	2223	2245	22	0

Outline

I. Introduction
II. Dataset
III. Model
IV. Framework
V. Evaluation
VI. Discussion

Conclusion

- **Contributions**:
  - Proved the possibility of LLM-assisted **HLS** generation for hardware design.
  - Proposed a **dataset** and **code infrastructures** for developing and evaluating **LLM-assisted HLS design generation**.
  - Integrated advanced techniques such as **feedback loops** and **chain-of-thought (CoT)** reasoning.

- **LLM-assisted HLS** demonstrates strong potential for designing **complex hardware** with high levels of syntax and functional correctness.

Thoughts

Key Factors for Language Selection (HLS vs HDL)

- **Quality of Generated Hardware Design**:
  - **Advantages of HLS**: Shares semantics and syntax with programming languages commonly used in LLM pre-training.
  - Demonstrated potential for **high syntax** and **functional correctness** in hardware designs.
- **Runtime Cost of Hardware Generation**:
  - **Token Efficiency**: HLS-based designs require fewer tokens during code generation, potentially reducing initial computational costs.
  - **Synthesis Costs**: The overall runtime costs associated with HLS synthesis need further analysis.

Future Work

- Investigate more into the hardware performance of LLM-generated HLS
- A comprehensive **quantitative comparison** of runtime costs for HLS and HDL
- Add more design samples to the current dataset

Current Work

- Distributed Training: An alternative way to alleviate data scarcity issue

- Advanced Inference-Time Optimization: Improve the output quality at test-time

THANK YOU!