OPL4GPT: An Application Space Exploration of Optimal Programming Language for Hardware Design by LLM

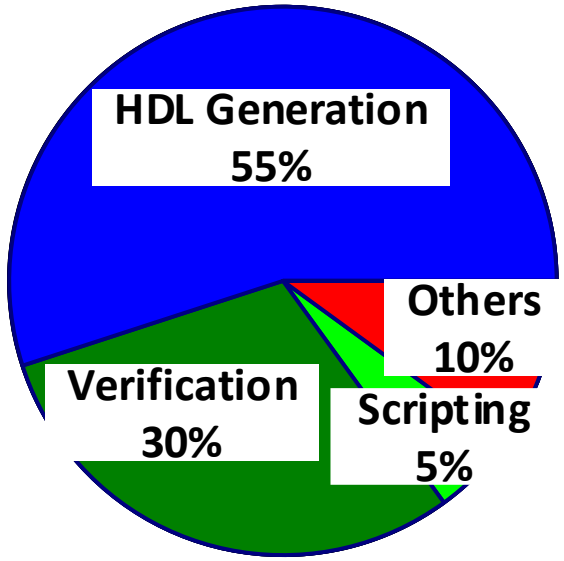
Kimia Tasnia, Sazadur Rahman
Department of Electrical and Computer Engineering,
University of Central Florida

Outline

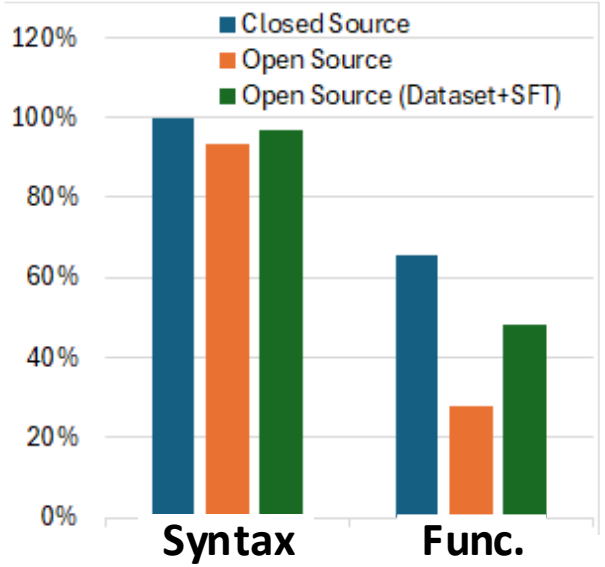
- Motivation
- Designing an SoC with LLM
  - Conversational flow to explore the optimal language for hardware design
  - OPL4GPT Framework
- Modular Prompting and Profiling
  - One case study- RSA Cryptography
- Experimental Results
  - Experimental Setup
  - Comparison between C/C++ and Verilog code
- Conclusion

Intro: Emerging Design Automation

- LLM is showing promising results to advance automated HDL generation
  - Existing works remain applicable mostly to Verilog code generation…
- However, LLM generated codes are falling short to meet syntax/functionality.
  - Requires multiple iterations to debug and converge to the optimal code



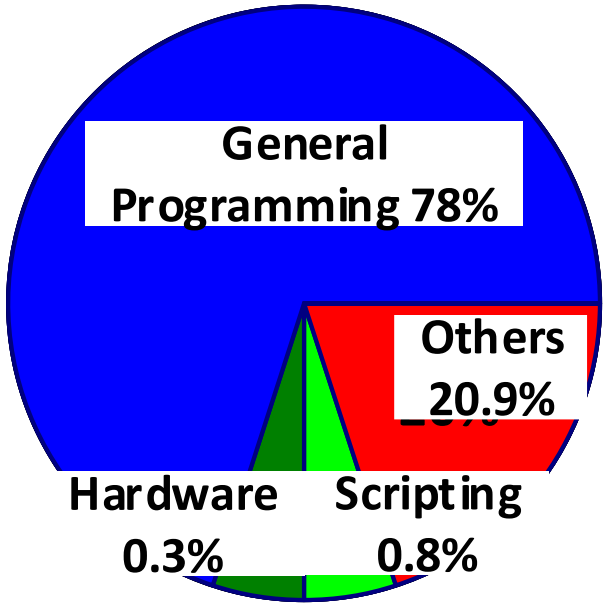
Survey of ~120
Research Papers

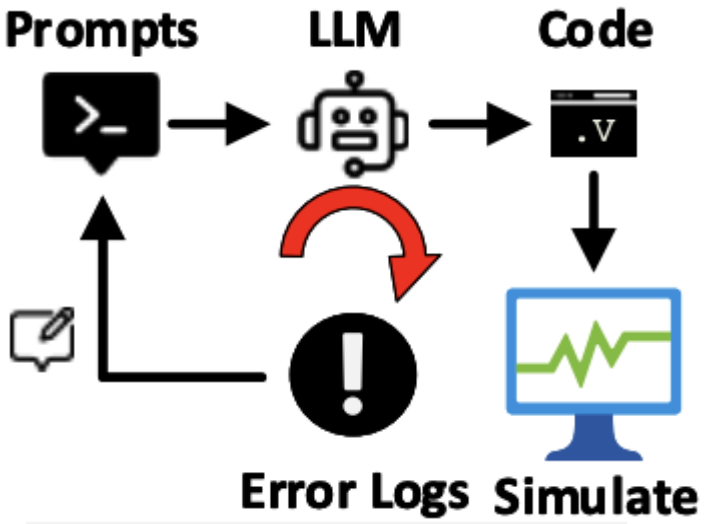


Pass Rate in
HDL coding

Problem Statement: Lack of Diverse Database

- How these generative AI models learned to code and debug?
  - They mostly included `GitHub` code repositories in the training database
  - `GitHub` database is biased to general purpose languages languages, e.g., `C++/Python`
- Moreover, simulating, debugging and iterating Verilog codes is time consuming!





Motivation: Appl. Specific Language

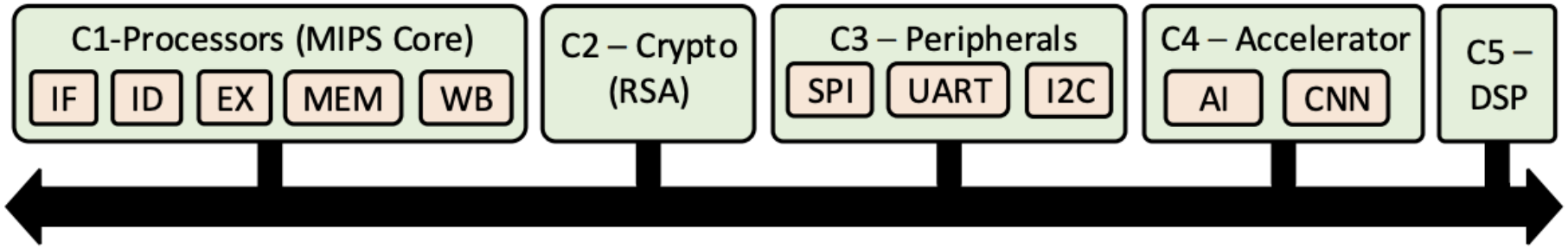
- Both register transfer level (RTL) and High-level synthesis offer unique benefits, hence our application space exploration for optimal language
  - **Register Transfer Level (RTL)**
    - Customization capability
    - Hardware resource constrained
    - Leverage legacy systems and interfaces
    - Suitable for fine-grained and critical applications
  - **High-level Synthesis (HLS)**
    - Reduces design complexity
    - Enables algorithmic optimization
    - Faster time-to-market
    - Applicable for complex and data intensive designs, e.g., cryptographic algorithms, AI accelerators, and DSP blocks

Research Objective

Assess LLM's capability in **application specific** and **optimal programming language** generation for hardware design

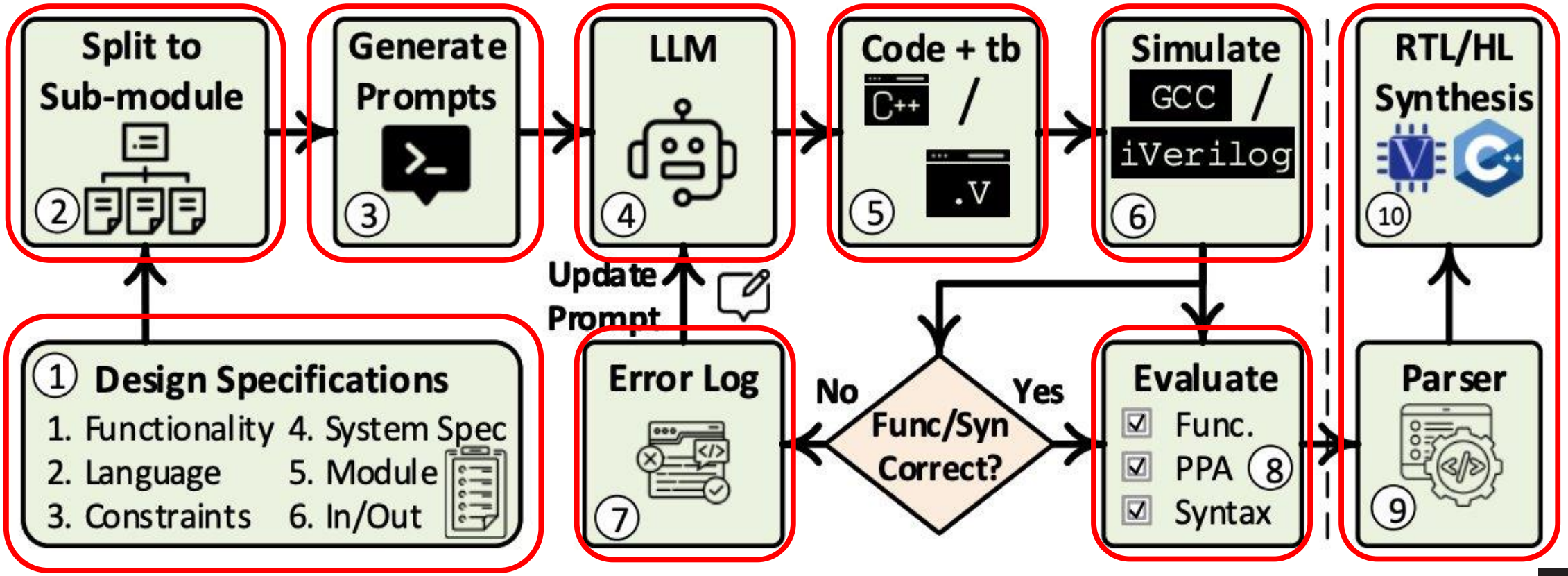
Designing an SoC with LLM

- To perform a comprehensive investigation of LLM's capability in generating Verilog (RTL) and C++ (HLS) codes



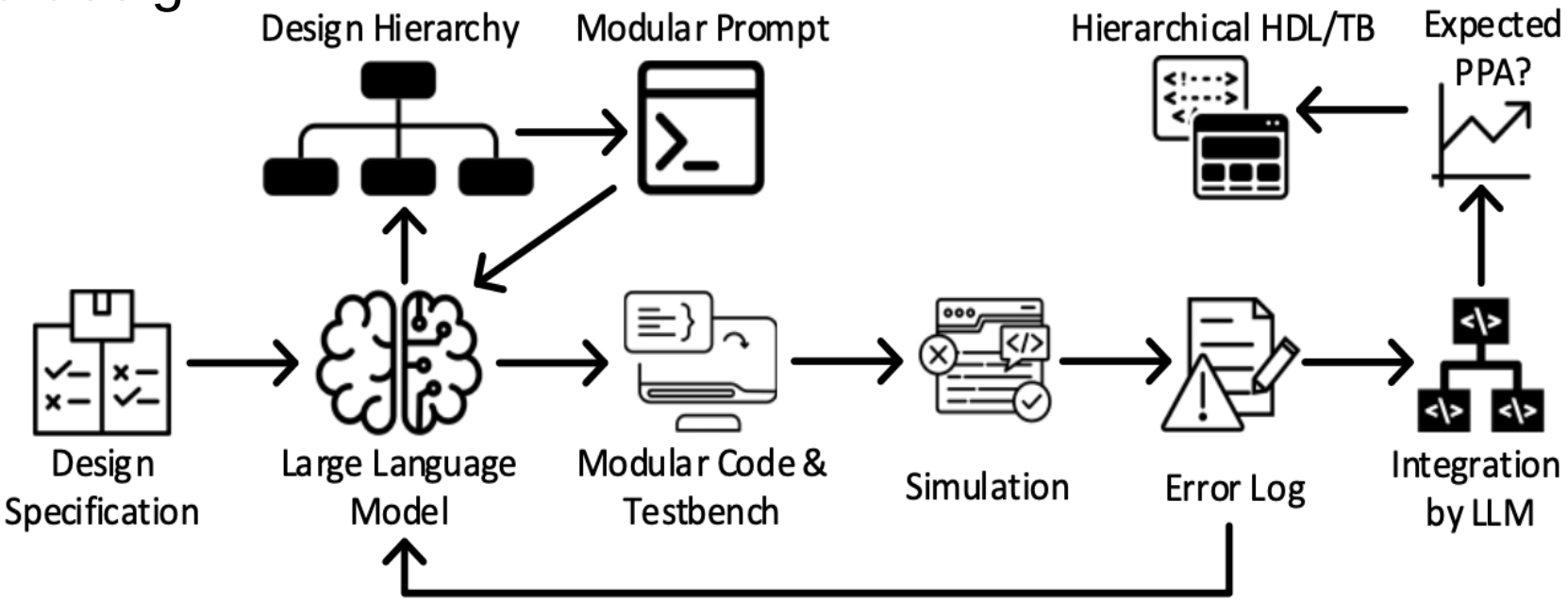
- A simplified SoC across **5 categories** of applications-
  - Processors - 16-bit MIPS
  - Crypto primitives - RSA
  - Peripherals - SPI, I2C, and UART
  - Hardware Accelerators - AI, and CNN, and
  - DSP - filters

OPL4GPT – Conversational Flow



Modular Prompting and Profiling

- Inspired from real world human-assisted RTL design workflow pattern
  - Divide the specification into parts according to the design hierarchy
  - Design a top module to instantiate and connect all the sub-modules
  - Verification follows the same way, first modular-wise verification then verify the integrated design



Case study – RSA Cryptography

- Sub-module 1 - Text to ASCII Conversion:
  - convert the plain text, numbers, and special characters to be encrypted to its corresponding ASCII format
- Sub-module 2 - Generating Public and Private Keys:
  - Choose two large, unique prime numbers p and q

$$n = pq$$

$$\lambda(n) = lcm(p-1, q-1)$$

  - Choose an integer e such that, e and lambda(n) are coprime

$$1 < e < \lambda(n)$$

$$\gcd(e, \lambda(n)) = 1;$$

  - Determine d as-

$$d \equiv e^{-1} \pmod{\lambda(n)}$$

Case Study Continues…

- Sub-module 3 - Encryption and Decryption:
  - Encrypt message m using public key e to generate ciphertext c

$$c \equiv m^e \pmod{n}$$

  - To recover m from ciphertext c, utilize the private key exponent d

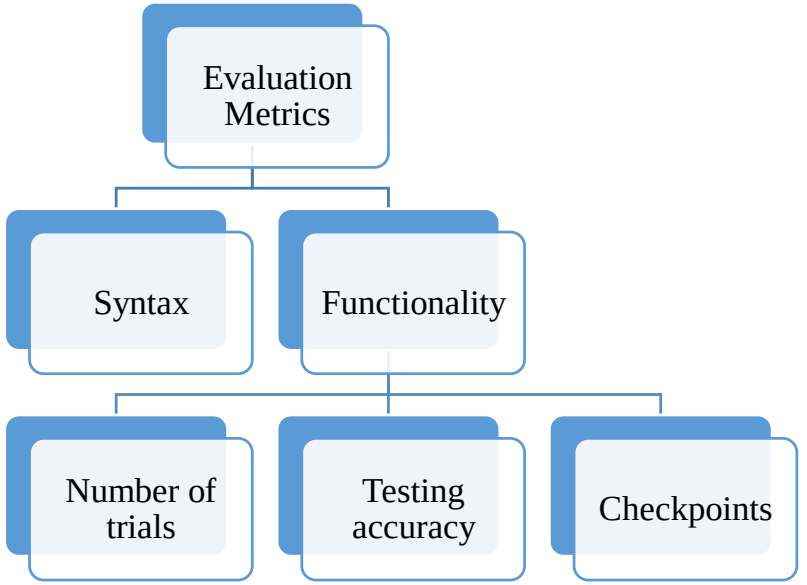
$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

- Verify each sub-module with corresponding testbench
- Integrating these 3 sub-modules with a top module
- Evaluate the results of the integrated RSA cryptomodule to encrypt and decrypt messages

UCF

Experiment Setup and Evaluation Metrics

- Leveraged OpenAI’s GPT-4o
- Evaluation Metrics:
  - Syntax: Assesses the syntactical correctness
  - Functionality: Measures the intended functionality as per the specification of the designed prompts
    - Number of trials required for error-free code
    - Testing accuracy evaluates the functional correctness of the generated code.
    - Checkpoints analyzes the design specific efficiency of the LLM’s capabilities



Results: C/C++ vs Verilog by LLM

Category	Modules	Sub-modules	*C++* Implementation				*Verilog* Implementation			
			Syntax	Functionality			Syntax	Functionality		
			#Trials	#Trials	Testing Acc. (%)	Checkpoints	#Trials	#Trials	Testing Acc. (%)	Checkpoints
Crypto	RSA	Text to ASCII	Zero Shot	Zero Shot	100%	✓Simulated successfully ✓Generated large random primes ✓Generated random pub/pvt Keys ✓Generated random keys	2	3	NA	✓Simulated successfully ✗Generated same primes ✗Incorrect simulation ✗Incorrect response after integration
		KeyGen	Zero Shot	Zero Shot			5	12		
		EncDec	Zero Shot	2			Zero Shot	Inf*		
Accelerators	CNN	Forward Pass Func Test Optimization	3 Zero Shot Zero shot	Zero shot 2 Successful inference	28.6%	✓Simulated successfully	Inf*	Inf*	NA	✗Syntax and compilation error
	AI	sub-mod1	1	Zero shot	100%	✓Implemented scalable/parallel model ✓Accurately implemented/processed weights/cache	Inf*	Inf*	NA	✗Syntax and compilation error
		sub-mod2	Zero shot	2		✓Delegated layers to multi-cores ✓Accurately implemented/processed weights/cache				
Peripherals	SPI	Clock Gen	Zero Shot	Zero Shot	100%	✓Simulated successfully	Zero Shot	Zero shot	80%	✓Simulated successfully
		Master	1	3		✗Manual debugging ✓Transferred/received data	Zero Shot	6		✓Transferred/received data ✗Incorrect initial bits
		Slave	1	2		✓No manual debugging ✓Transferred/received data	Zero Shot	4		✓Transferred/received data ✗Incorrect initial bits
DSP	FIR Filters	BandStop OptFIR	Zero Shot 1	Zero Shot Zero Shot	100%	✓Generated Cutoff freq., hamming window ✓Pipelined MAC, FP arithmetic, loop unrolling	2 1	Zero shot Zero Shot	100%	✓Generated Cutoff freq., hamming window ✓Pipelined MAC, FP arithmetic, loop unrolling
Processor	16-bit MIPS	ISA	Zero Shot	3	100%	✓Successfully compiled/simulated	Zero Shot	4	100%	✓Successfully compiled/simulated
		Register file	1	2			1	3		
		ICache	1	Zero Shot			Zero Shot	Zero Shot		
		ALU	Zero Shot	Zero Shot			Zero Shot	Zero Shot		
		Fetch	1	5			1	6		
		Decode	Zero Shot	2			3	14		
		Exec	Zero Shot	Zero Shot			Zero Shot	Zero Shot		
		DCache	Zero Shot	Zero Shot			Zero Shot	3		
		Mem	Zero Shot	Zero Shot			Zero Shot	2		
		WB	Zero Shot	Zero Shot			Zero Shot	Zero Shot		

*: Could not generate a working code after 15 trials.

Discussion

- The RSA module showed promising performance for the codes generated in C++ with 100% accuracy in contrast to Verilog.
- For CNN and AI accelerator, C++ by LLM outperforms Verilog across all the evaluation metrics including design optimization.
- Both C++ and Verilog achieved 100% test accuracy in generating codes for Processor and DSP block design.
- Both languages passed the syntax test, C++ achieved 100% test accuracy, whereas for Verilog with 80% and more trials.

Conclusions and Future Work

- We developed a solid workflow that utilizes LLMs helping designers select the optimum language based on application
  - Our novel prompt engineering techniques combined with the application space exploration developed the automated OPL4GPT framework
  - LLMs can leverage the advantage of higher abstraction levels in hardware code generation besides focusing only on Verilog
  - C/C++ outperforms in the case of mathematical, algorithm intensive (e.g., RSA), involving inference, pipelining, and parallel circuits (e.g., CNN, AI accelerator)
  - For peripherals and processors with finite state machines, besides Verilog, C++ can also achieve competitive performance
  - For future, we are exploring masking techniques to better prompt engineer and fine-tune LLM in HDL generation

Thank you!

UCF