2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Generative AI based RTL Code Generator

Hareesh Perumal S
Infineon Technologies
Bangalore, India

Gandem Kranthi Kiran
Infineon Technologies
Bangalore, India

Pathak Vishnu Dutt
Infineon Technologies
Bangalore, India

Ashutosh Bajpai
Infineon Technologies
Bangalore, India

Dr. R. K. Sharma
NIT Kurukshetra
Haryana, India

***Abstract*- Recent progress in artificial intelligence has steered researchers to use natural language instructions to create Register Transfer Level (RTL) code, like Verilog, by harnessing the power of Large Language Models (LLMs). In the realm of automated RTL code generation, the current landscape is dominated by methodologies that are heavily reliant on proprietary Large Language Models (LLMs) such as ChatGPT. On the other hand, open-source LLMs, despite being customized for such tasks, have not met the performance benchmarks set by their commercial counterparts and it is much costlier as well to train these medium to large sized open-source LLMs on downstream task like code generation. To address this problem, our research introduces a novel framework that finetunes a very small language model of size less than 7 billion parameters with high quality instruct dataset samples, generated through distillation techniques, that help the small model with huge enhancements in reasoning and coding capabilities. The experimental results also prove, with respect to the task of RTL code generation, our small language model is capable of rivaling GPT3.5 and GPT4, whose size are (100x-1000x) more than our model.**

Keywords— Small Language models, Register transfer language, Instruct dataset, Knowledge distillation, Domain Adaptive Pre-Training (DAPT).

I. INTRODUCTION

Over the past decades, electronic design automation (EDA) algorithms and tools have significantly increased the productivity of chip industry. Combined with the exponential increase in transistor density as predicted by Moore's Law, EDA has enabled the development of complex SoC designs with many functions and billions of transistors. Electronic design automation (EDA) is a set of integrated circuit (IC) design software programs and services that work together in a design flow to conceptualize and analyse circuit designs. In a digital hardware design flow, designers write code in a Hardware Description Language (HDL) such as Verilog or VHDL to specify the hardware architecture and behaviour assisted by the EDA tools. This process is time consuming and error prone as the transistor density increases, which has led to recent slowdown of Moore's Law, pressurizing EDA tool workflow and raking up the need of further improved automation of design flow.

In recent years, large-scale language models (LLMs) such as GPT [2] have demonstrated remarkable performance in natural language processing (NLP). Inspired by this progress, researchers have begun to consider the adoption of LLMs in agile hardware design. More recently, researchers have been exploring how to apply AI to EDA algorithms and chip design processes to further improve chip design productivity [3][4][5]. However, many of the time-consuming chip design tasks that involve interfacing with natural and programming languages have yet to be automated. GitHub's Copilot was found to be creating security bugs during simple Verilog code completions [10]. Indeed, early academic work [11][12][13] explored the application of LLM to generate RTL that can generate small design modules and generate scripts for EDA tools.

While general LLMs trained on large amounts of Internet data show remarkable capabilities in generative AI tasks across a variety of domains (as shown by Bubeck et al.in [9]), recent studies such as BloombergGPT [15] and BioMedLLM [16] have shown that domain-specific LLM models can outperform general models on domain-specific tasks. In the area of hardware design, [6][12] showed that an open-source LLM (CodeGen [18]) optimized with additional Verilog data can outperform state-of-the-art OpenAI models. The research team of NVIDIA proposed its own labeled training dataset and benchmark [17], then finetuned its own new model. This

might had been the first non-commercial model [14] that claims comparable performance with GPT-3.5, but neither the training dataset nor fine-tuned LLM model had been released in the public.

A Customized LLM solution with modest parameter counts of only 7B achieving better performance than GPT-3.5 through an innovative approach of training using a custom Verilog instruct dataset of 27,000 samples produced groundbreaking results [23], but still was not able to improve performance of SLMs (Small Language models) whose model parameter count < 7B. In this work, we address this issue, by finetuning small Language model of parameter count <7B on RTL generation task and improve its performance multifold, to rival the likes of proprietary models like ChatGPT-4 and ChatGPT-3.5 despite being (100x -1000x) small.

Contributions of our work can be summarized below:

- We introduce an instruction dataset generation framework which helps in generating quality instruct dataset from initial unlabeled data, that helps in improving the reasoning and chat capabilities of LLMs that are even less than 7B parameter in size and help them rival their bigger counterparts in terms of performance.
- Utilizing the concept LLM-as-a- Mentor, our work introduces a novel way of RTL generation, where we chain our finetuned LLM with open-source LLM like mixtral-8x7b, to achieve incredible improvements in accuracy. Here the finetuned model gives the shape and functionality to the code and the chained LLM corrects the finetuned model's response both logically and functionally.
- We have implemented series of advance techniques in filtering, deduplication, cleaning and tokenizing dataset, that helps creating a cleaner & higher quality dataset to prevent overfitting.

II. METHODOLOGY

A. *Proposed Workflow:*

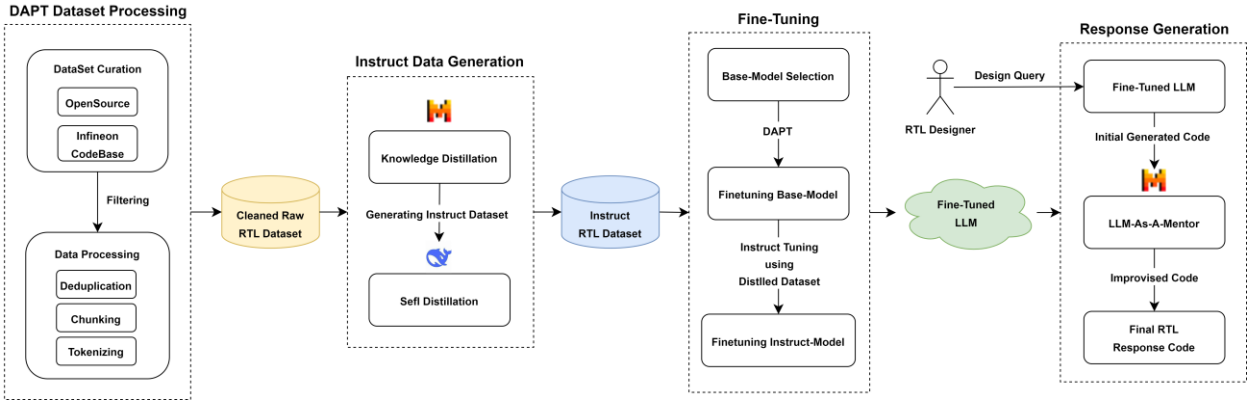

Fig. 1. End-2-End Worflow

Complete proposed workflow is depicted by the fig.1, it commences with the selection of the most prolific pre-trained model within the category of Small Language Models (SLMs), defined by their parameter count of fewer than 10 billion. This chosen model is then subjected to domain-adaptive continuous pretraining with raw data that is extracted and processed by series of advanced techniques, followed by which the model is aligned with an

instruction dataset generated by a novel framework based upon dataset distillation, improvising the model for the specialized task of RTL code generation. This finetuned model which is good at hardware domain comprehension and RTL code generation combines with an open-source LLM as Mixture of Experts (MoE) to generate state of the art RTL designs. As per an industry standard benchmark our model's performance rival the performance of proprietary models like ChatGPT-4 and ChatGPT-3.5 despite being (100x -1000x) small.

B. Base Small Language foundation model Selection:

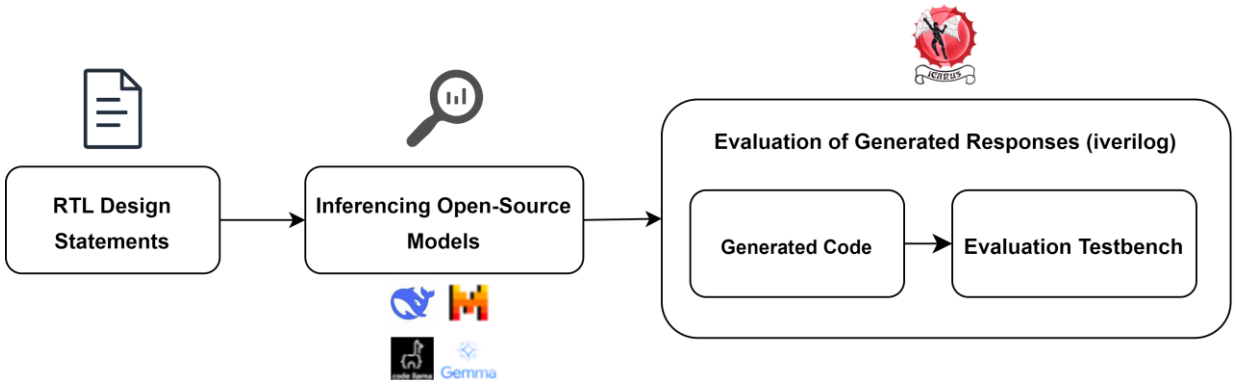

Fig. 2. Base Pre-trained Model Selection Process

We performed extensive research commencing with evaluation of custom set of selected RTL design statements of varying difficulty from the paper [11] using state of the art open source LLMs such as Llama2-70b, Mixtral-8x7b, Mistral-7b, DeepSeek Coder, etc. The performance of these models is then benchmarked for RTL code generation, leading to the identification of the best performing small-sized pretrained model**,** making it best in class and computationally less expensive to train. The evaluation of base models was carried on using Iverilog compiler and based on evaluation standards, that has been portrayed in the table below.

TABLE I. Evaluation Criteria for Base Pre-trained Model Selection

Response Status	Score
Functionally & Syntactically Correct	3
Logically Correct	2
Syntactically Correct	1
Incorrect	0

The score in the given table is used to provide scores for every single response generated by the LLMs as per the conditions mentioned in the table. The best small language foundation model that was selected through above evaluation procedure, is Deepseek Coder-1.3b, it is of only 1.3 billion parameters and its code generation capability compared with other pretrained LLMs is provided in result section in tabular format.

C. Dataset Generation:

1) Domain-Adaptive Pre-training Dataset:

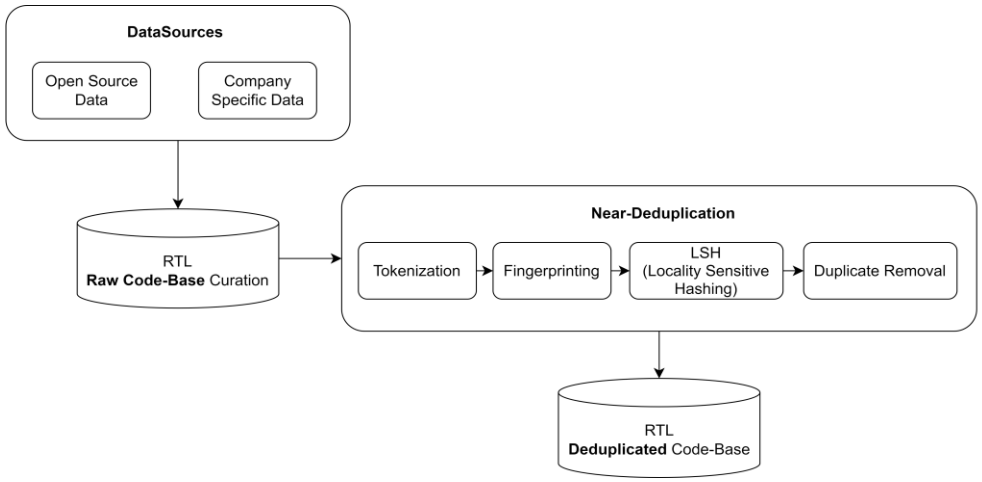

Fig. 3. DAPT Dataset De-Duplication Procedure

DAPT dataset helps the base foundational model to gain understanding of RTL code syntax, code logic coding principles and hardware domain knowledge through domain adaptive pretraining [14]. They are generally unlabeled dataset, and, in this work, we had constructed a huge unlabeled RTL dataset by scraping through internet and gathering RTL files, related design documentations and other miscellaneous hardware domain data. Along with that, most part of our DAPT dataset is internal proprietary data, that had been filtered through file type to create dataset consisting of RTL design, verification, and design requirement files.

It is observed that models tend to reproduce training data verbatim when there are numerous duplicates in training dataset. This behaviour not only raises concerns about model integrity but also exposes the model to potential privacy breaches. So, we had removed duplicate data samples with Minhash deduplication in our work [33]. The standard process for MinHash deduplication involves the following steps:

1. Shingling (tokenization) and fingerprinting (MinHashing): This entails converting each document into a set of hashes.
2. Locality-sensitive hashing (LSH): This step involves grouping documents with similar bands together to decrease the number of comparisons.
3. Duplicate removal: In this stage, decisions are made regarding which duplicated documents to retain or discard.

2) Instruct Dataset Generation:

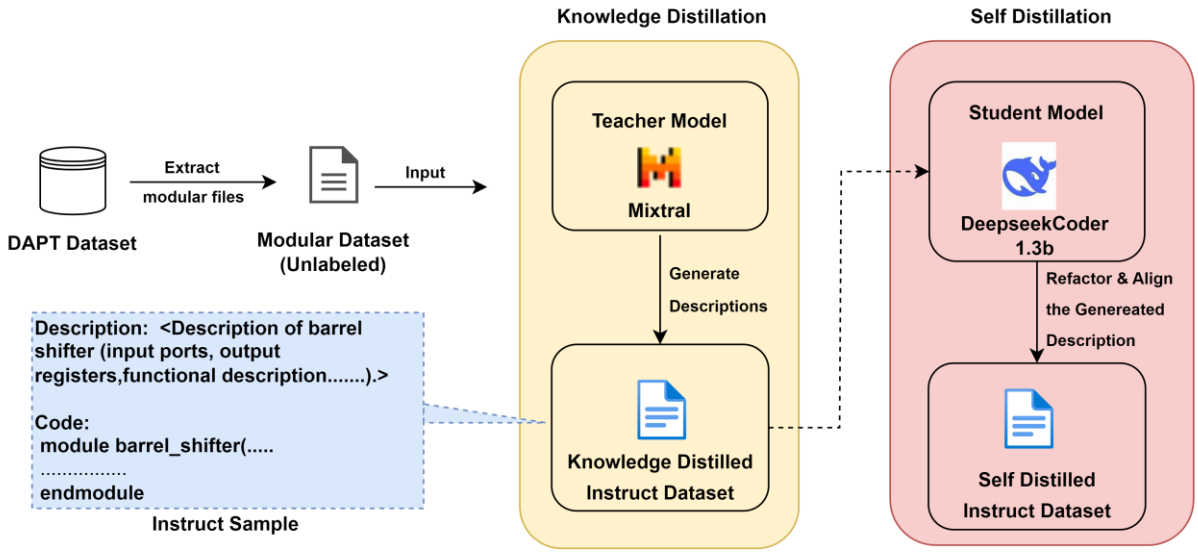

Fig. 4. Instruct Dataset Generation Framework

The bigger open source models such as Mixtral 8x7b, LLAMA-70b etc. possess advanced capabilities and reasoning ability compared to smaller models like LLAMA-7b, Mistral-7b, Deepseekcoder-1.3b etc. They do have better understanding of technical concepts compared to their smaller counterparts and that's why they are able to excel in multitude of task compared to smaller LLMs. To bridge this knowledge disparity gap between larger and smaller models, recently a lot of techniques have emerged, among this knowledge distillation is one of the most effective technique [29]. Knowledge distillation, helps in guiding training process of smaller student LLM with a large pretrained model that acts as the teacher LLM. There are many types of knowledge distillation like response-based distillation, distillation through synthetic data generation, etc. But we opt for the synthetic data generation approach as the training cost of other distillation methods are high and to train our model through instruct dataset generated by this technique.

So, the complete flow of our instruct dataset generation framework powered by distillation techniques is as depicted by fig.4 above, where first the DAPT dataset is filtered to gather dataset containing only singular module files. Through this filteration, we gathered around 72k non-similar singular module files. After the filteration process, the filtered dataset undergoes transformation through knowledge distillation and self-distillation, which are explained in detail below.

- Knowledge distillation:

  Here the Mixtral 8x7b is used as the teacher model, where the filtered single module files are sent as input to it and is told to generate design description for each of those unlabeled single module files, thus with description, code pairs the initial instruct dataset is generated through knowledge of our teacher model as shown in fig.4

- Self-distillation:

  Self-distillation has been shown to alleviate the issue of catastrophic forgetting [30], which is common when continually fine-tuning models with fewer than 7 billion parameters[32]. By employing self-distillation on the training dataset, we can ensure that the weights of the student large language model (LLM) do not undergo significant alterations post-training. This technique involves having the student LLM independently generate design description on its own, using code-description pairs obtained through knowledge distillation as references. For our fine-tuning process, we have chosen DeepseekCoder-1.3b as the student model to execute self-distillation, ultimately producing the final Instruct dataset.

D. Continual Fine-Tuning:

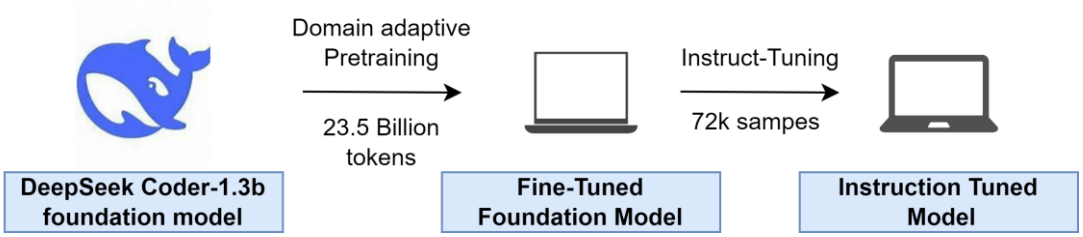

Fig.5 . Continual Fine-Tuning Flow

Continual fine-tuning is performed over the selected base foundation model, DeepSeek Coder-1.3b, that is the base model undergoes Domain adaptive Pretraining first and then instruct tuned to improve alignment, reasoning and chat capabilities of the model. The complete flow of this continual finetuning has been depicted by the fig.5.

[1] Domain adaptive pretraining (DAPT):

At first Domain adaptive pretraining [13] is performed over the selected base model DeepSeek Coder-1.3b with 23.5 billion tokens of RTL data, as seen in the fig.5 above and this quantity of raw RTL data is quite enough for foundational finetuning of our base model to make it expert in RTL code generation and RTL coding practices.

All the model training process was optimized using the DeepSpeed's ZeRO technique [26], incorporating concepts such as data parallelism and flash attention [27] for enhanced efficiency. This Domain adaptive pretraining process is resource intensive and was performed using 4 NVIDIA A30 GPUs, each of 24GB VRAM, with small learning rate of 5e-6 and training is facilitated using Adam optimizer. The train batch size was set at 1, and a context length of training samples were limited to 2048 tokens. For the setup of 4 NVIDIA A30's to train our base model upon 1 billion tokens of RTL data, it took around one day and as we had collected 23.5 billion tokens of RTL data for foundational finetuning, the whole process took around 25 days.

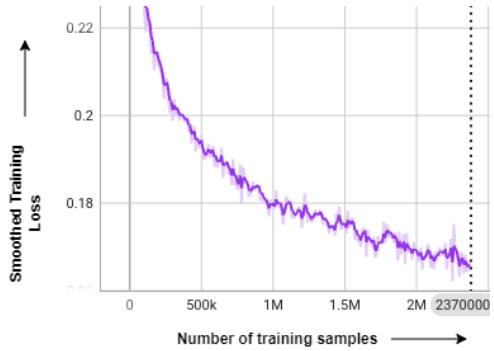

Fig.6 . DAPT Training Loss

[2] Instruct Tuning:

After DAPT we obtain finetuned foundation model which is good at hardware domain comprehension and good at next token prediction (**code completion process**). But it lacks reasoning and chat capabilities which is core quality of any coding assistant. So, to improve reasoning, alignment and chat capability, we use RTL instruct dataset generated by our data distillation framework to instruct tune our foundation model into Instruct model as shown in above fig.6. This instruction dataset contains around 72,000 samples, each instruct data sample contains a design description and its corresponding code, which helps the model to understand, how to respond to any design description query by producing the corresponding code.

E. Enhanced Response Generation through LLM-as-a-Mentor:

LLM-as-a-Mentor is a technique to grade the response of a primary LLM by passing the output of primary LLM to a separate LLM (Mentor/Judge) [28] asking it to mentor the output and modify it as per the needs to get an enhanced result.

Our instruct tuned model, when inferred does generate better quality RTL code and performs on-par with SOTA LLMs like ChatGPT 4 and GPT 3.5 in VerilogEval benchmark, whose results can be seen in Result section. It generates logical part of the RTL code well for complex design, but sometimes, its performance is impeded by lot of syntactical errors in its generated responses. To address this issue and to explore whether the accuracy of our instruct tuned model could be enhanced, we experimented chaining our instruct tuned model with LLMs of high reasoning ability and good Verilog syntax understanding. So, we chained it with Mixtral-8x7b with notion of Mixtral-8x7b to be Mentor LLM as shown in the fig.7.

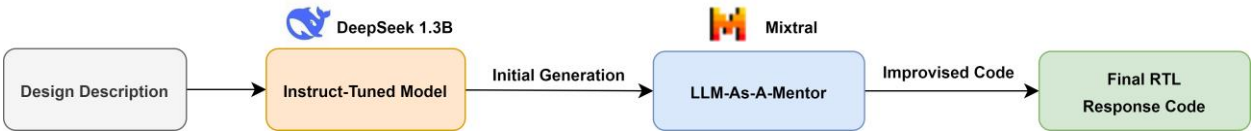

Fig. 7. Response Generation through LLM as Mentor

So now, the response generated by finetuned model is syntactically checked by the Mixtral which is chained to our finetuned model, then the corrected response by Mixtral is passed to verilogEval framework and we got improved results which are displayed in Result section.

III. EXPERIMENTAL RESULTS

Pre-trained models loaded with either fp32 or fp16 precision has been evaluated using design questions from paper [11] and on basis of evaluation standard from Table 1. The table below shows, how they have fared.

Table 2. Performance Comparison of Pre-Trained Base Models

S.No	Model Name	Model Size (billion)	Performance Score (30)
1	LLAMA-2 70B	70	**21**
2	DeepseekCoder-1.3B	**1.3**	19
3	VICUNA - 7B	7	16
4	VICUNA – 13B	13	15
5	Qwen 14b chat	14	12
6	Codellama-34-b-instruct	34	7
7	VICUNA -33B	33	6
8	MISTRAL -7B	7	3

From the above table, we could clearly see Deepseek Coder despite being the smallest model performed better than or same as pre-trained LLMs of more than 10x parameter size.

So, the Deepseek Coder model is selected to be finetuned, and after continually finetuning it through Domain adaptive pretraining [2] and instruct tuning later, we had achieved in making a finetuned small language model that excels in RTL generation and does better than or similar to SOTA models, the performance has been recorded in the below verilogEval benchmark Table 3. This finetuned model is also then chained with the Mentor model Mixtral 8x7b to further improve performance and enhance quality of generated response then this chained approach is evaluated with VerilogEval framework, its performance is also compared along with other popular models as also portrayed in the table below.

Table 3. Performance Comparison of Finetuned Model with VerilogEval Framework

S.NO	MODEL NAME	MODEL SIZE (billion)	Pass@1	Pass@5	Pass@10
1	RTLCoder	7	**58.9**	70	74.9
2	**Our Model + Mixtral -8x7b**	1.3 + 34	52.85	**74.2**	**80.5**
3	GPT-4	1700	47.9	67.8	72.9
4	GPT-3.5	~180	46.7	69.1	74.1
5	NVIDIA ChipNeMo	13	43.4		
6	Mixtral-8x7b	34	40.69	60.11	65.67
7	Our Model	1.3	40.49	54.28	58.731
8	Codegen-16b-multi-sft	16	37.1	55.0	61.1
9	Codegen-2b-verilog-sft	2	35.9	59.0	65.7
10	Deepseek Coder -1.3b	1.3	35.42	45.37	49.29
11	VeriGen	2	20.1	46	55.9

As depicted in the table, we could see that despite being only 1.3 billion parameters in size, the finetuned model performs on par with other models that are of 10x-100x in size. Here the metrics such as Pass@1, Pass@5 and Pass@10 means the model is capable of generating x amount of Verilog designs that is both compliable and functionally correct out of hundred designs in VerilogEval framework within first try (pass@1), within five tries (pass@5) and within ten tries (pass@10) respectively [17].

Instruction: Design a verilog module that implements a three input xor

Response:

```
module my_xor3 (
    output X,
    input  A,
    input  B,
    input  C
);

    wire notA, notB, notC;
    wire and1, and2, and3, and4;
    wire or1, or2, or3;

    // Invert inputs
    not (notA, A);
    not (notB, B);
    not (notC, C);

    // AND gates
    and (and1, A, notB, notC);
    and (and2, notA, B, notC);
    and (and3, notA, notB, C);
    and (and4, A, B, C);

    // OR gates
    or (or1, and1, and2);
    or (or2, or1, and3);
    or (or3, or2, and4);

    // Output
    assign X = or3;

endmodule
```

Fig.8. Finetuned model generated code for 3 input XOR

Instruction: Design a module that implements 32 bit shift register

Response:

```
module shift_reg_32 (
    clk,
    p_load,
    p_data,
    s_in,
    s_out,
    shift_en
);

input clk;
input s_in;
input p_load;
input [31:0] p_data;
input shift_en;
output s_out;

reg [31:0] shreg;

always @(posedge clk) begin
    if (p_load) begin
        shreg <= p_data;
    end else begin
        if (shift_en) begin
            shreg <= {shreg[30:0], s_in};
        end
    end
end

assign s_out = shreg[31];

endmodule
```

Fig.9. Finetuned model generated code for 32-bit Shift Register

Along with the performance score, examples of real time responses that has been generated by finetuned model has been shown in the fig.7 and fig.8 above. In these images, design requirements are passed in as instruction prompt and the code is received as response.

IV. CONCLUSION AND FUTURE SCOPE

So, through the results we are able to conclude that the finetuned model produced through this work performs on par with SOTA LLMs and has edge over the other big models in terms of ease of deployment and less compute resource consumption. Through chaining the finetuned model, we did also observe that the pass@1 rate of our solution had shot up to 52.85 compared to 40.49 that was achieved by the finetuned model alone, as the combined

reasoning of the LLMs has played in good effect**.** So, extending this as inspiration, other prompting and chaining techniques have to be explored in near future to generate Verilog code of different hierarchies when scaling this generative AI powered solution to system or SoC level code generation. As the lines of code would scale up multifold in SoC level, techniques such as LongRoPE [31] have to be implemented to train the LLMs with Verilog codes of that complexity and multiple hierarchies.

V. REFERENCES

[1] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M. Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. 2019. HardFails: Insights into Software-Exploitable Hardware Bugs. 213–230. https://www.usenix.org/conference/usenixsecurity19/ presentation/dessouky

[2] OpenAI. 2023. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774 (2023).

[3] B. Khailany et al., "Accelerating chip design with machine learning," IEEE Micro, vol. 40, no. 6, pp. 23–32, 2020.

[4] H. Ren and M. Fojtik, "Invited- nvcell: Standard cell layout in advanced technology nodes with reinforcement learning," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021.

[5] R. Roy et al., "PrefixRL: Optimization of parallel prefix circuits using deep reinforcement learning," in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021

[6] W.-L. Chiang et al., "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," March 2023. [Online].

[7] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023

[8] Rolf Drechsler, Ian G. Harris, and Robert Wille. 2012. Generating formal system models from natural language descriptions. In IEEE Int. High Level Design Validation and Test Workshop (HLDVT). 164–165.

[9] Christopher B. Harris and Ian G. Harris. 2016. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. In Design, Automation Test in Europe Conf. Exhibition (DATE). 966–971

[10] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan- Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In 2022 IEEE Symposium on Security and Privacy (SP). 754–768. https://doi.org/10.1109/SP46214.2022.9833571 Nvidia

[11] S. Thakur et al., "Benchmarking large language models for automated verilog rtl code generation," in 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023, pp. 1–6.

[12] J. Blocklove et al., "Chip-chat: Challenges and opportunities in conversational hardware design," 2023.

[13] Z. He et al., "Chateda: A large language model powered autonomous agent for eda," 2023

[14] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. 2023. ChipNeMo: Domain- Adapted LLMs for Chip Design. arXiv preprint arXiv:2311.00176 (2023)

[15] S. Wu et al., "Bloomberggpt: A large language model for finance," 2023.

[16] M. LLC. (2022) Biomedlm: a domain-specific large language model for biomedical text. [Online]. Available: https://www.mosaicml.com/blog/introducing-pubmed-gpt

[17] M. Liu et al., "VerilogEval: evaluating large language models for verilog code generation," in 2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2023

[18] E. Nijkamp et al., "Codegen: An open large language model for code with multi-turn program synthesis," ICLR, 2023

[19] S. Gururangan et al., "Don't stop pretraining: Adapt language models to domains and tasks," 2020.

[20] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. arXiv preprint arXiv:2305.02309 (2023)

[21] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! arXiv preprint arXiv:2305.06161 (2023)

[22] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. arXiv preprint arXiv:2310.06825 (2023)

[23] Liu, S., Fang, W., Lu, Y., Zhang, Q., Zhang, H., & Xie, Z. (2023). Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open- source dataset and lightweight solution. *arXiv preprint arXiv:2312.08617*.

[24] Nadimi, B., & Zheng, H. (2024). A Multi-Expert Large Language Model Architecture for Verilog Code Generation. *arXiv preprint arXiv:2404.08029*.

[25] Hayati, S. A., Jung, T., Bodding-Long, T., Kar, S., Sethy, A., Kim, J. K., & Kang, D. (2024). Chain-of-Instructions: Compositional Instruction Tuning on Large Language Models. *arXiv preprint arXiv:2402.11532*

[26] Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020, November). Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-16). IEEE.

[27] Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io- awareness.

Advances in Neural Information Processing Systems, *35*, 16344-16359.

[28] Zhu, L., Wang, X., & Wang, X. (2023). Judgelm: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*.

[29] Xu, X., Li, M., Tao, C., Shen, T., Cheng, R., Li, J., ... & Zhou, T. (2024). A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*.

[30] Yang, Z., Pang, T., Feng, H., Wang, H., Chen, W., Zhu, M., & Liu, Q. (2024). Self-Distillation Bridges Distribution Gap in Language Model Fine-Tuning. *arXiv preprint arXiv:2402.13669*.

[31] Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., ... & Yang, M. (2024). Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*.

[32]Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., & Zhang, Y. (2023). An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

[33] https://huggingface.co/blog/dedup