BetterV: Controlled Verilog Generation with Discriminative Guidance

Zehua Pei[1], Hui-Ling Zhen[2], Mingxuan Yuan[2], Yu Huang[2], Bei Yu[1]

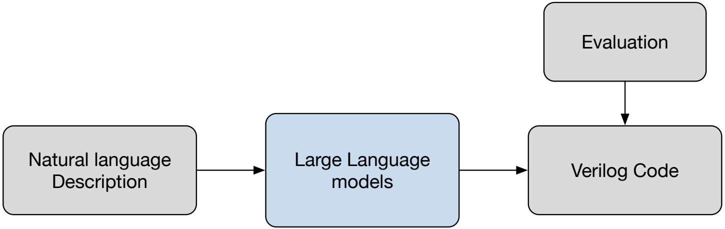
[1]The Chinese University of Hong Kong
[2]Noah's Ark Lab, Huawei, Hong Kong SAR

Outline

LLMs based Verilog Generation

Given the natural language descriptions as input, the large language models (LLMs) try to output the Verilog code. The generated Verilog is expected to be syntactically and functionally correct.

- Syntactic correctness. The Verilog obeys the rules and structure defined by the Verilog language specification.
- Functional correctness. The Verilog satisfies the requirements from the natural language descriptions.

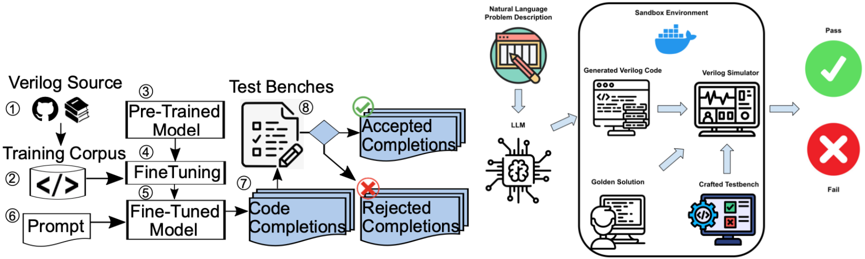


The Flow of LLMs based Verilog Generation

Verilog generation Literature

Existing Verilog generation works focus on fine-tuning the LLMs with customized datasets and developing evaluation benchmarks.

Some **problem-sets** are constructed as requirements to the generation, and some **testbenches** are used to evaluate the functionality[12].



(a) *NYU*[1]

(b) *NVIDIA*[2]

[1] Shailja Thakur et al. (2023). "Benchmarking Large Language Models for Automated Verilog RTL Code Generation". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1–6.

[2] Mingjie Liu et al. (2023). "VerilogEval: Evaluating Large Language Models for Verilog Code Generation". In: *arXiv preprint arXiv:2309.07544*.

Existing Challenges

Three challenges need to be addressed to enhance the performance and practicability of LLM based Verilog generation:

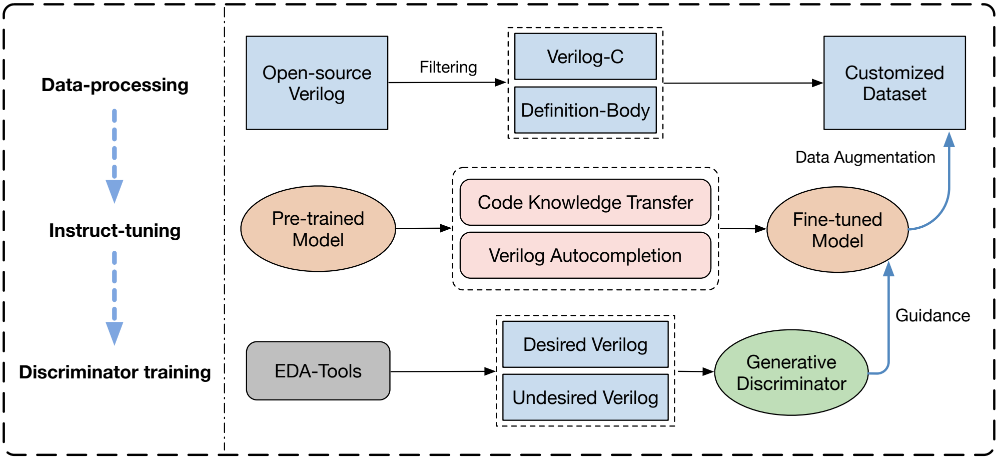
- **Complicated requirements of Hardware designs**: The complex and strict requirements of hardware designs restrain LLMs from learning and understanding the knowledge related to Verilog.
- **Limited Verilog resources**: There are limited Verilog resources available globally, which often leads to problems of overfitting and data bias during LLM fine-tuning.
- **EDA downstream tasks**: The Electornic Design Automation (EDA) downstream tasks should be further considered. However, it is difficult for LLM to understand the downstream tasks, which involve customized and complex definitions.

Our solutions

In this work, we attempt to address these challenges by proposing a framework, **BetterV**.

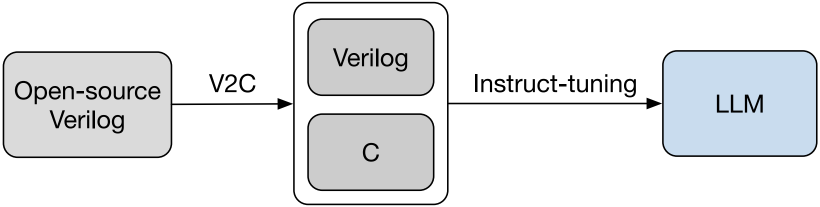
- **Code Knowledge Transfer**: We design a novel instruct-tuning process to aligns Verilog to C, which helps transfer the knowledge of LLMs on general code to Verilog.
- **Discriminative Guidance**: We utilize a generative discriminator to guide the LLMs to generate or modify Verilog implementations directly from natural language, towards specific optimization on downstream tasks.
- **Data Augmentation**: We implement a simple but effective solution to augment data for Verilog scarcity.

Framework Overview



The overview of BetterV

We firstly use the tool **V2C** to convert the Verilog into C. Then the Verilog-C pairs are used as dataset for LLMs **instruct-tuning**, which drives the alignment from C to Verilog.



The pipeline of Code Knowledge Transfer

Code Knowledge Transfer-2

System Prompt:
You are an experienced Verilog engineer.

Instruction:
You are supposed to translate the following Verilog into C program.

```
module d_latch(
    input d,
    input en,
    input rstn,
    output reg q
);

always @ (en or rstn or d)
    if (!rstn)
        q <= 0;
    else
        if (en)
            q <= d;
endmodule
```

Answer:

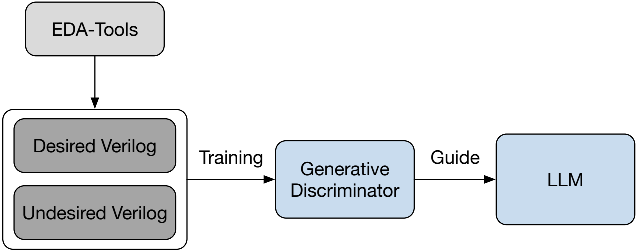
```
#include <stdio.h>
#include <stdbool.h>

void d_latch(
    int d,
    bool en,
    bool rstn,
    int *q) {

    if (!rstn) {
        *q = 0;
    } else if (en) {
        *q = d;
    }
}
```

An example of Instruction of Code Knowledge Transfer

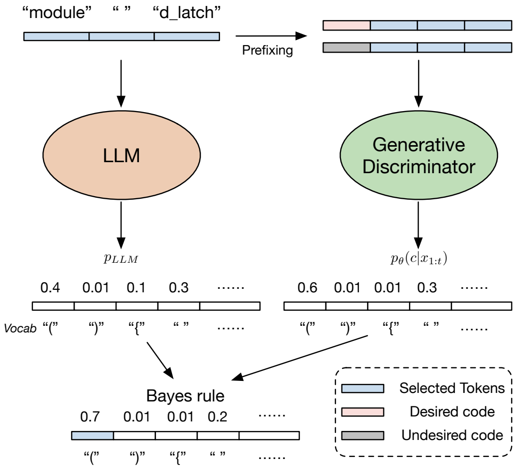
We employ generative discriminator to guide LLMs on specific Electronic Design Automation (EDA) tasks, which will give optimization on the Verilog implementation.



The pipeline of Discriminative Guidance

Discriminative Guidance-2

The weighted decoding powered by Bayes rule is employed to guide the generation.



An example shows the guidance from generative discriminator

Table: Comparison of functional correctness on VerilogEval.

Model	VerilogEval-machine			VerilogEval-human		
	pass@1	pass@5	pass@10	pass@1	pass@5	pass@10
GPT-3.5	46.7	69.1	74.1	26.7	45.8	51.7
GPT-4	60.0	70.6	73.5	43.5	**55.8**	**58.9**
CodeLlama	43.1	47.1	47.7	18.2	22.7	24.3
DeepSeek	52.2	55.4	56.8	30.2	33.9	34.9
CodeQwen	46.5	54.9	56.4	22.5	26.1	28.0
ChipNeMo	43.4	-	-	22.4	-	-
Thakur et al.	44.0	52.6	59.2	30.3	43.9	49.6
VerilogEval	46.2	67.3	73.7	28.8	45.9	52.3
RTLCoder-Mistral	62.5	72.2	76.6	36.7	45.5	49.2
RTLCoder-DeepSeek	61.2	76.5	81.8	41.6	50.1	53.4
BetterV-CodeLlama	64.2	75.4	79.1	40.9	50.0	53.3
BetterV-DeepSeek	67.8	79.1	84.0	45.9	53.3	57.6
BetterV-CodeQwen	**68.1**	**79.4**	**84.5**	**46.1**	53.7	58.2

Table: Synthesis nodes reduction with discriminator.

Problem	Ref	BetterV-base	BetterV	Com Base	Com Ref
ece241_2013_q8	657	333.5	255.3	23.44%	61.14%
m2041_q6	1370	692.7	685.6	1.03%	49.95%
counter_2bc	673	666.2	518.9	22.11%	22.89%
review2015_count1k	487	493.4	402.6	18.44%	17.33%
timer	498	294.3	247.3	15.97%	50.34%
edgedetect2	58	189.9	47.4	75.03%	18.27%
counter1to10	325	266.3	240.3	9.76%	26.06%
2013_q2afsm	826	308.8	296.6	3.95%	64.09%
dff8p	50	42.3	37.8	10.63%	24.4%
fsm3comb	844	167.9	104.4	37.82%	87.63%
rule90	6651	12435.6	4536.9	63.52%	31.79%
mux256to1v	2376	2439.6	557.2	77.16%	76.54%
fsm2	389	186.53	121.9	34.65%	68.66%
fsm2s	396	163.7	144.1	11.97%	63.61%
ece241_2013_q4	2222	1789.5	897.4	49.85%	59.61%
conwaylife	43794	547400.3	27037.4	95.06%	38.26%
count_clock	3187	2497.5	2222.2	11.02%	30.27%
countbcd	1589	932.0	849.3	8.87%	46.55%

Experiment-3

Table: Verification runtime reduction with discriminator.

Design	Ref (s)	BetterV-base (s)	BetterV (s)	Com Base	Com Ref
b03	1.233	1.252	0.857	31.54%	30.49%
b06	0.099	0.083	0.078	6.02%	21.21%
Spinner	1.577	1.343	1.064	20.77%	32.53%
traffic_light_example	0.583	0.497	0.480	3.42%	17.67%
Rotate	1.153	1.126	1.034	8.17%	10.32%

THANK YOU!