IEEE/ACM
2025 INTERNATIONAL
CONFERENCE ON
COMPUTER-AIDED
DESIGN
ICCAD
44th Edition

GenEDA: Towards Generative Netlist Functional Reasoning via Cross-Modal Circuit Encoder-Decoder Alignment

Wenji Fang, Jing Wang, Yao Lu, Shang Liu, Zhiyao Xie

Hong Kong University of Science and Technology

ICCAD @ Munich, Germany

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

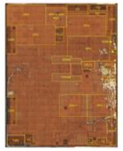
Background and Related Works

Challenges in Delivering Better Chips

Increasing **IC design complexity**

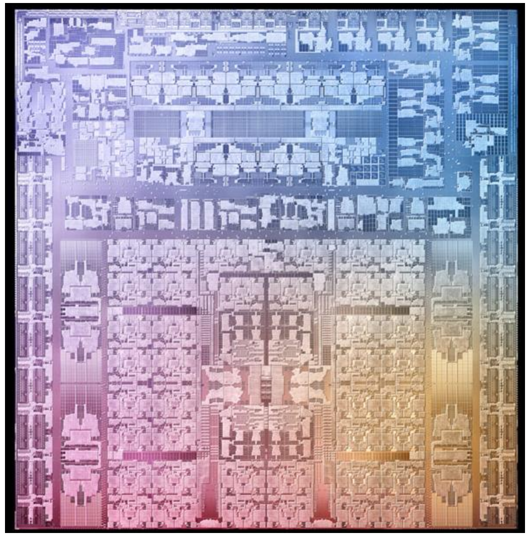
- Increasing **IC design cost**
- Increasing **time to market**

IC complexity



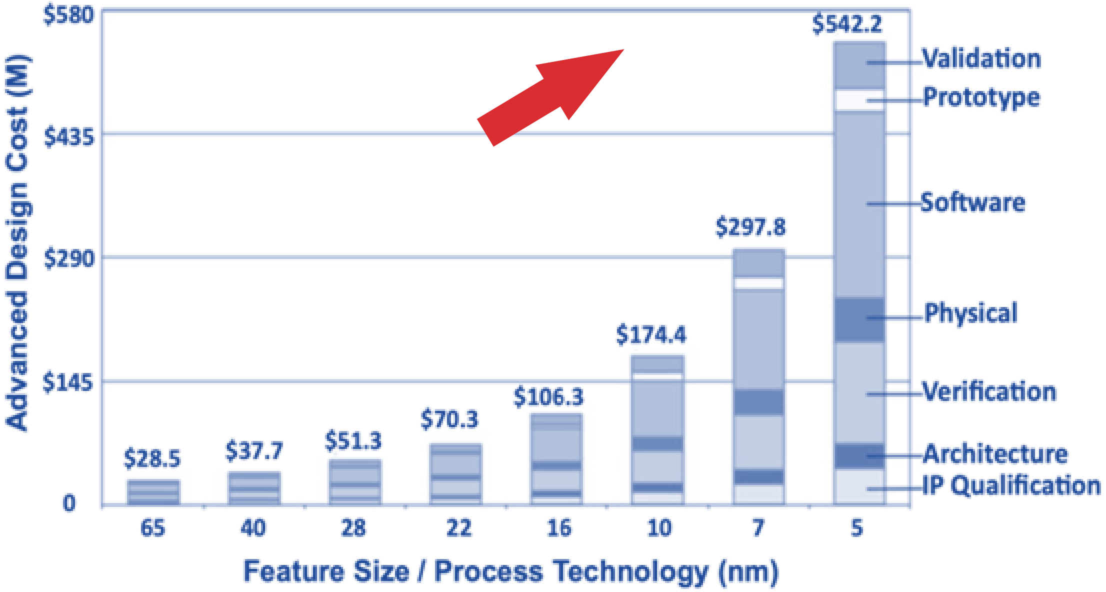
Apple A11
4B transistors

Apple A15
15B transistors



Apple M3 Max
92B transistors

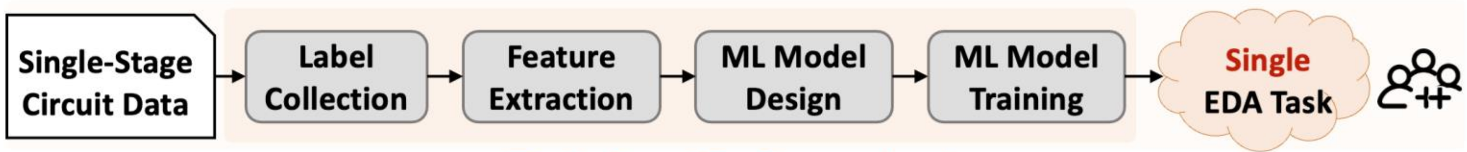
IC Design Cost is Skyrocketing



(Not including manufacturing)

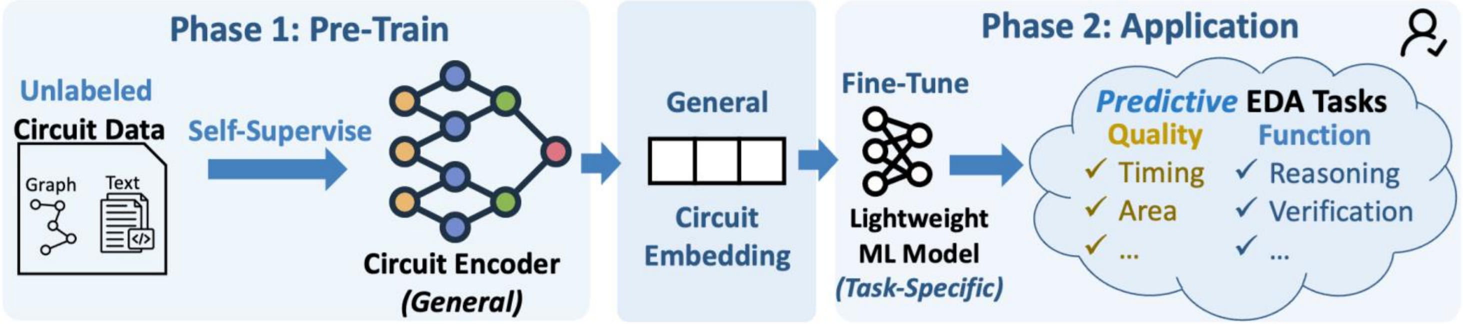
Background: AI for EDA Paradigm Shift

- Supervised learning → Circuit Foundation Models



Single-Stage Circuit Data → Label Collection → Feature Extraction → ML Model Design → ML Model Training → **Single** EDA Task

(a) Type I: Task-Specific Supervised AI for EDA



Phase 1: Pre-Train

Unlabeled Circuit Data (Graph, Text) — Self-Supervise → Circuit Encoder *(General)*

→ General Circuit Embedding →

Phase 2: Application

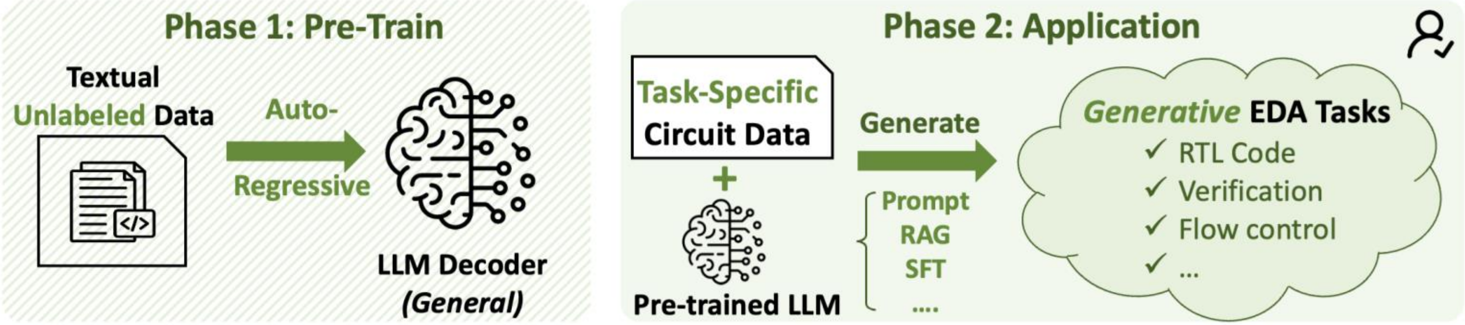
Fine-Tune Lightweight ML Model *(Task-Specific)* →

Predictive EDA Tasks

Quality: ✓ Timing ✓ Area ✓ ...

Function: ✓ Reasoning ✓ Verification ✓ ...

(b) Type II: General Encoder-Based Circuit Foundation Model



Phase 1: Pre-Train

Textual Unlabeled Data — Auto-Regressive → LLM Decoder *(General)*

Phase 2: Application

Task-Specific Circuit Data + Pre-trained LLM — Generate (Prompt, RAG, SFT, ...) →

Generative EDA Tasks

✓ RTL Code
✓ Verification
✓ Flow control
✓ ...

(c) Type II: General Decoder-Based Circuit Foundation Model

Our Survey Paper

Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

Background: Circuit Foundation Models

Type I: **Supervised Predictive** AI Techniques for EDA

Type II: **Foundation** AI Techniques for EDA

(Circuit Foundation Model)

Paradigm 1: **Encoder**-based

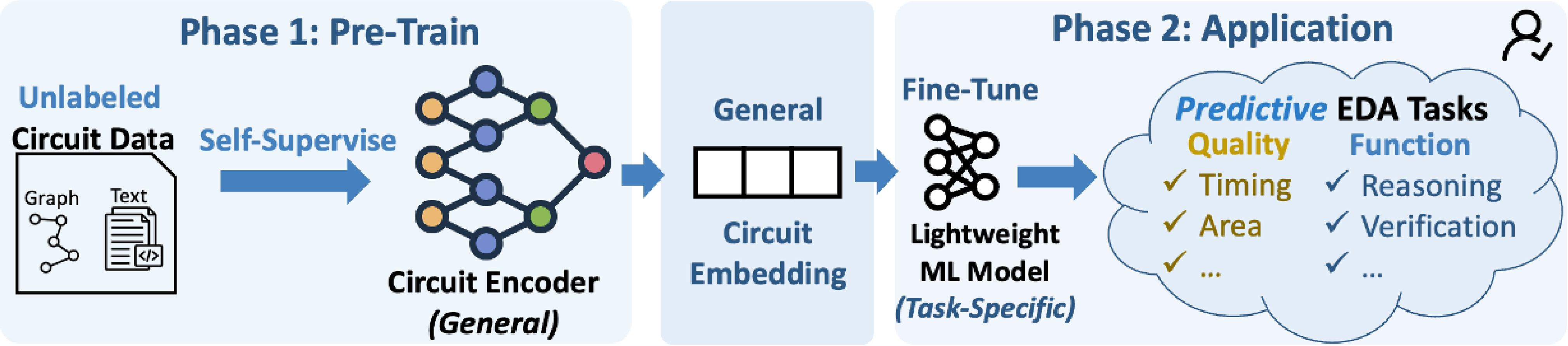
Paradigm 2: **Decoder**-based

Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

Background: Circuit Foundation Models

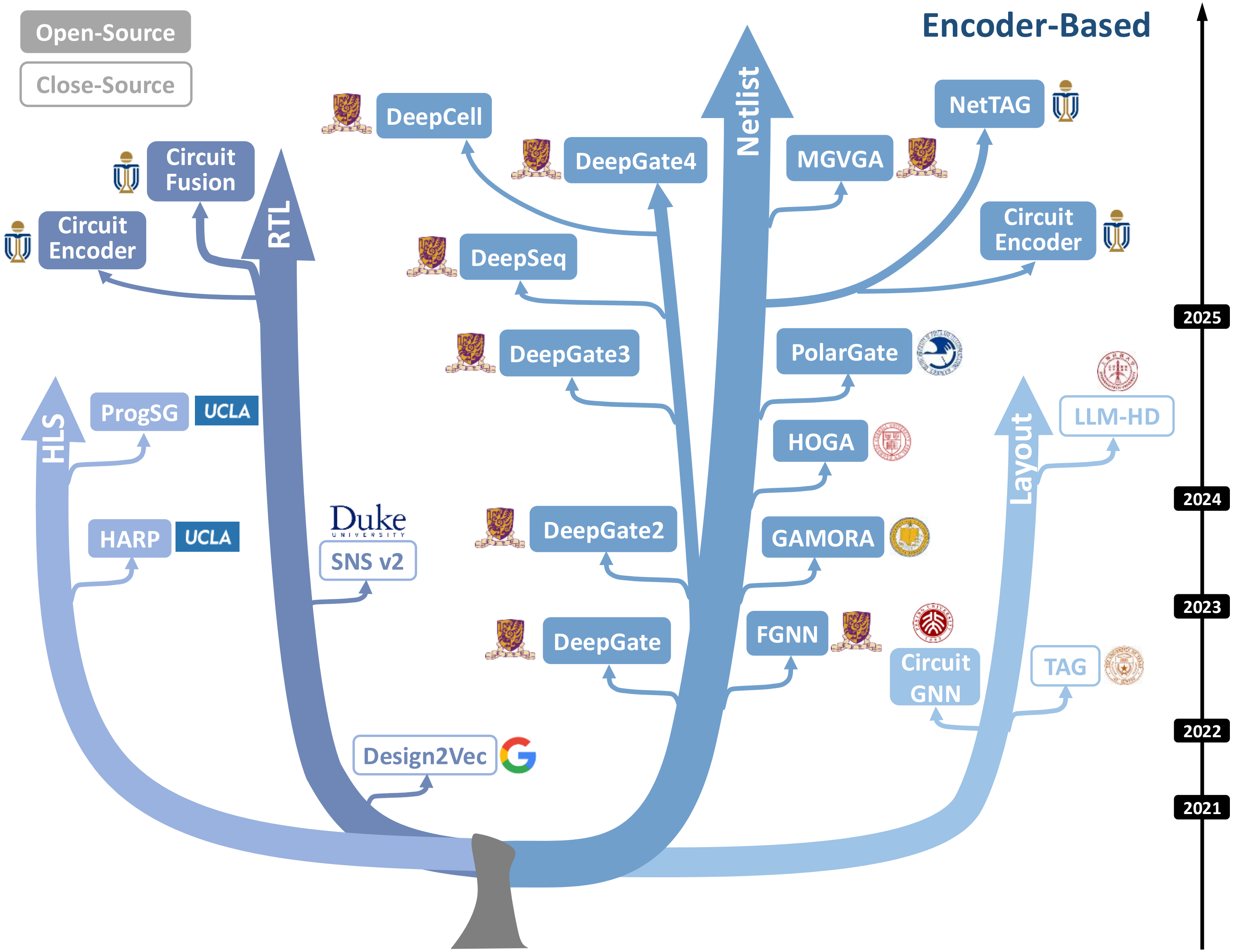
Paradigm 1: **Encoder**-based circuit foundation models

Paradigm 2: **Decoder**-based circuit foundation models



Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

Encoder-based Circuit Foundation Model

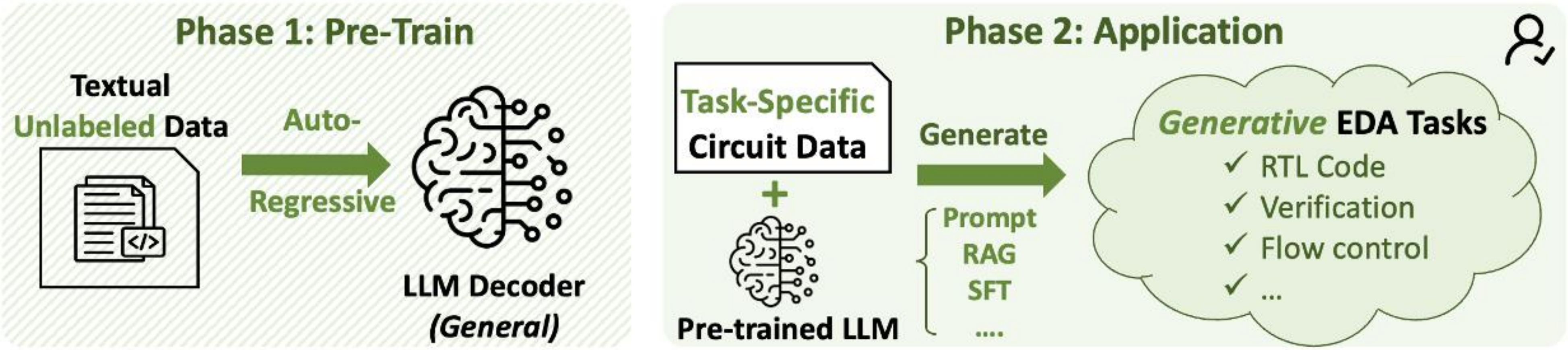


Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

How AI Assists EDA/Chip Design

Paradigm 1: Encoder-based circuit foundation models

Paradigm 2: **Decoder**-based circuit foundation models



Phase 1: Pre-Train

Textual Unlabeled Data

Auto-Regressive

LLM Decoder *(General)*

Phase 2: Application

Task-Specific Circuit Data

+

Pre-trained LLM

Generate

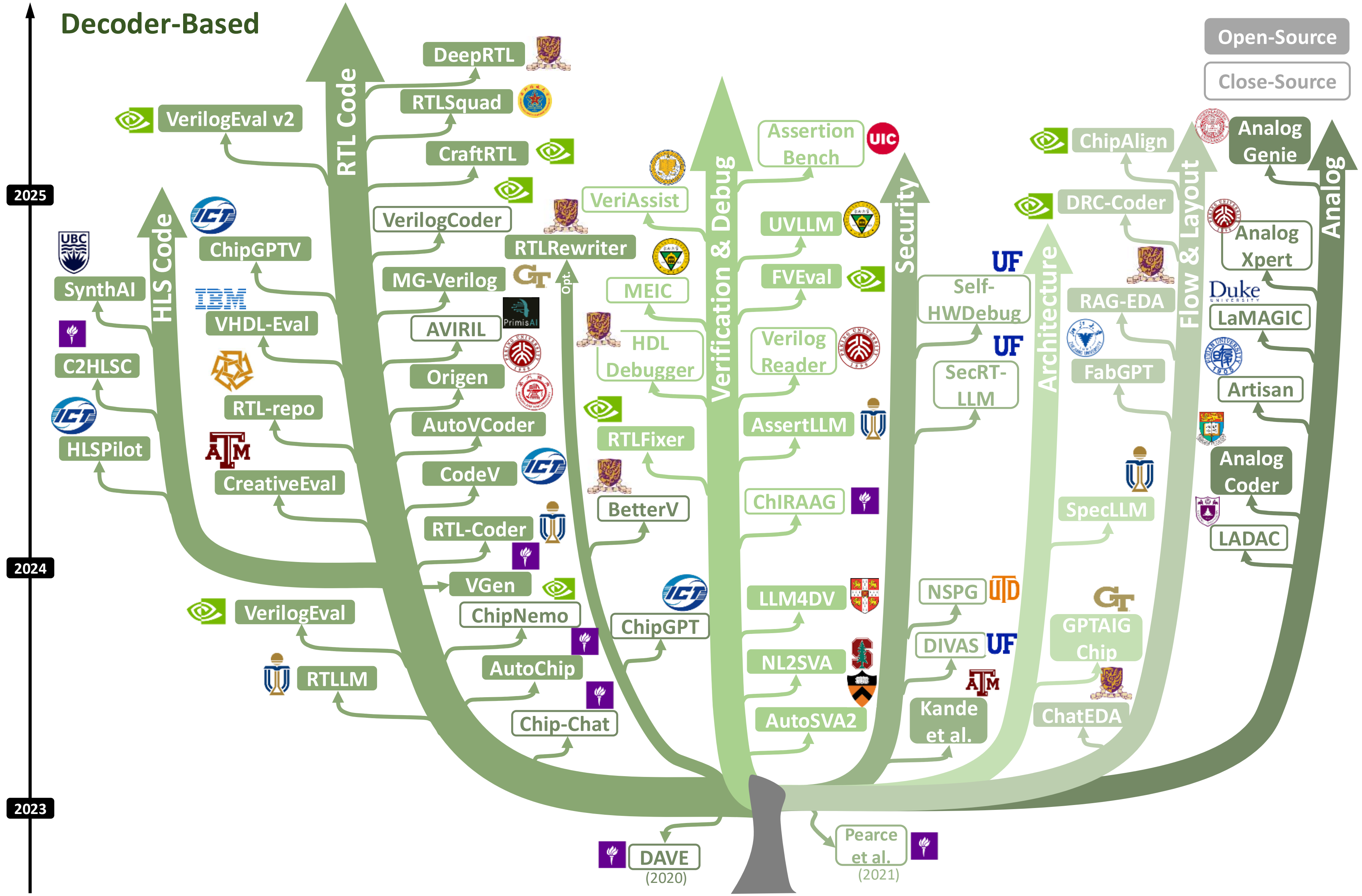
Prompt
RAG
SFT
....

***Generative* EDA Tasks**

- ✓ RTL Code
- ✓ Verification
- ✓ Flow control
- ✓ ...

Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

Decoder-based Circuit Foundation Model

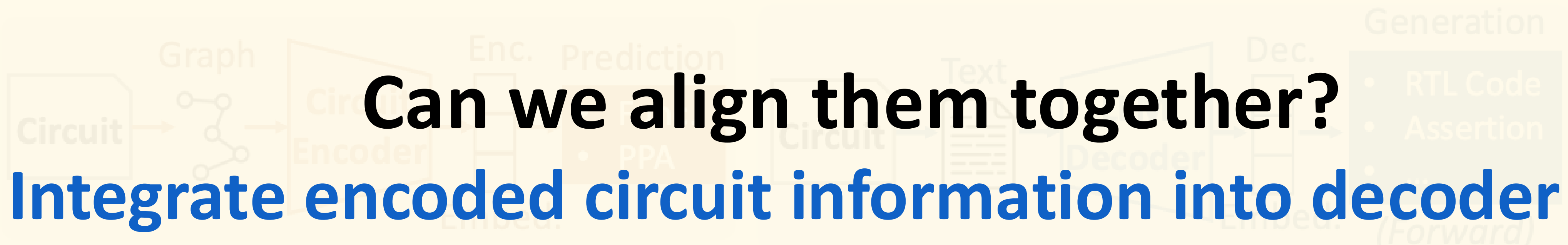


Wenji Fang et al. A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Limitation: Gap between Encoders and Decoders

- Developed independently, operated in *distinct latent space*
  - **Encoder:** graph space, circuit embeddings for prediction
  - **Decoder:** text space, text embeddings for generation



(a) Circuit encoder for prediction

(b) Circuit decoder for generation

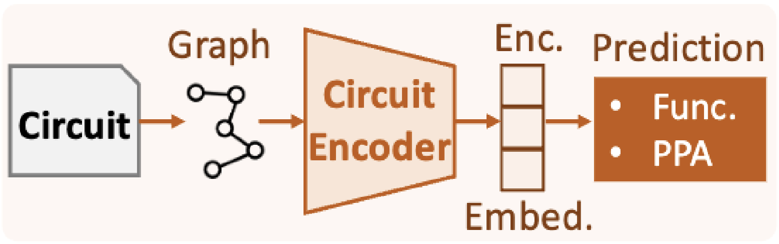
Can we align them together?

- Integrate encoded circuit information into decoder LLMs

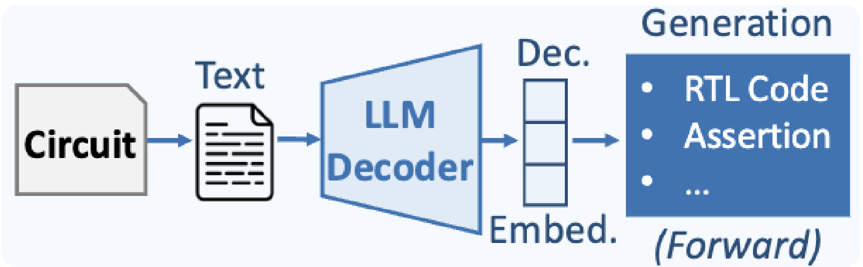
GenEDA Method and Application - Overview

GenEDA: Bridging Gap Between Encoder-Decoder

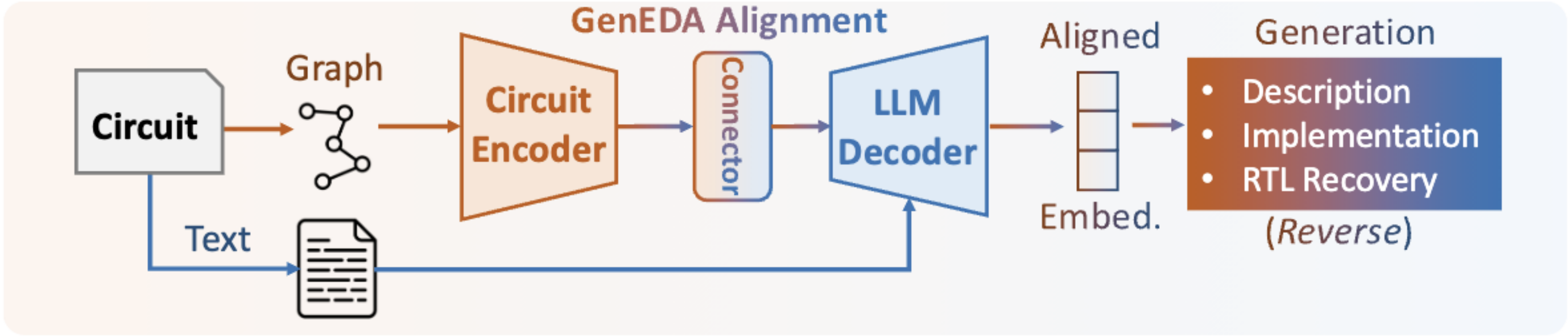
- An Encoder-Decoder alignment framework with **connectors**
  - Communicate circuit graph & text latent space



(a) Circuit encoder for prediction



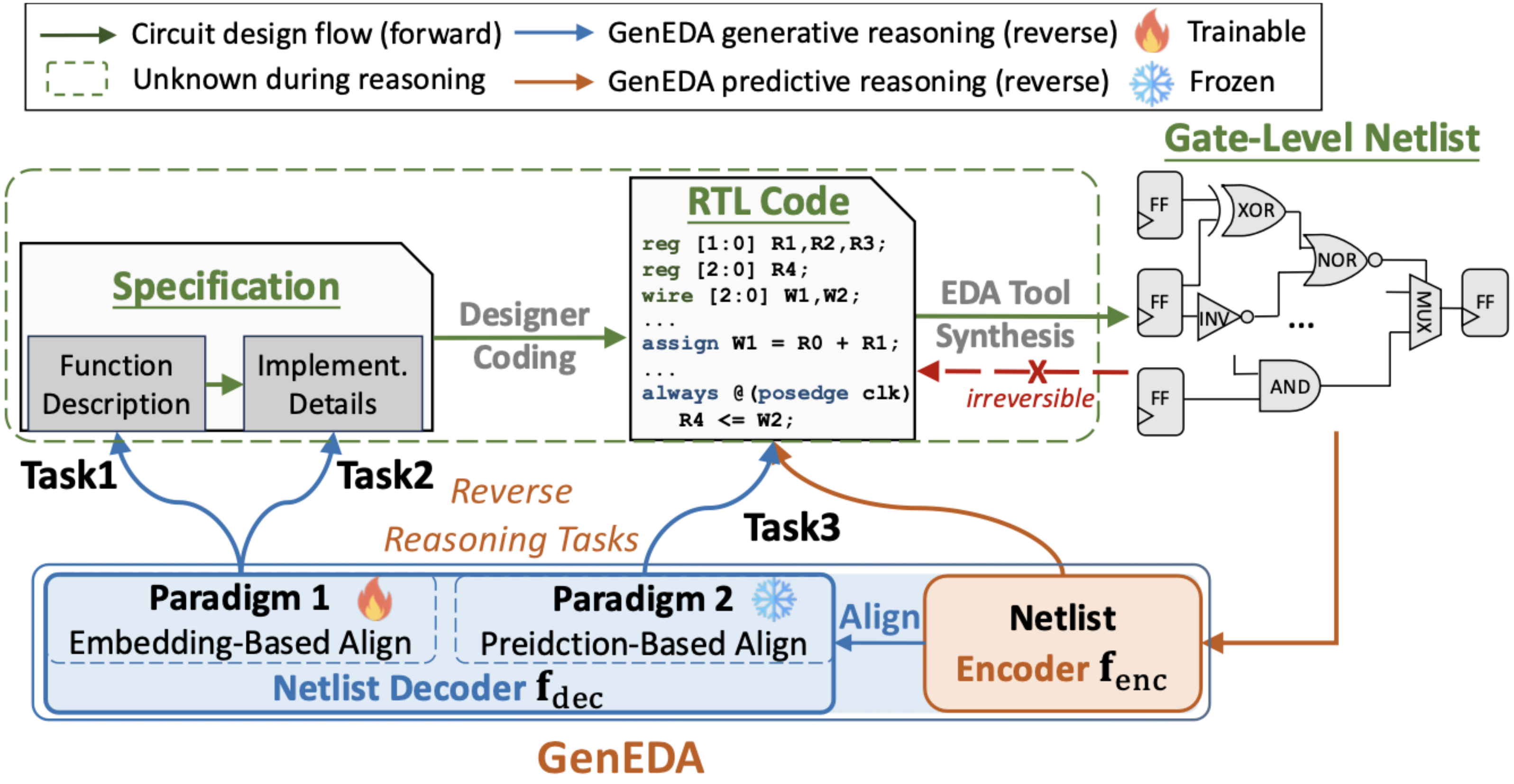
(b) Circuit decoder for generation



(c) Our proposed circuit encoder-decoder framework

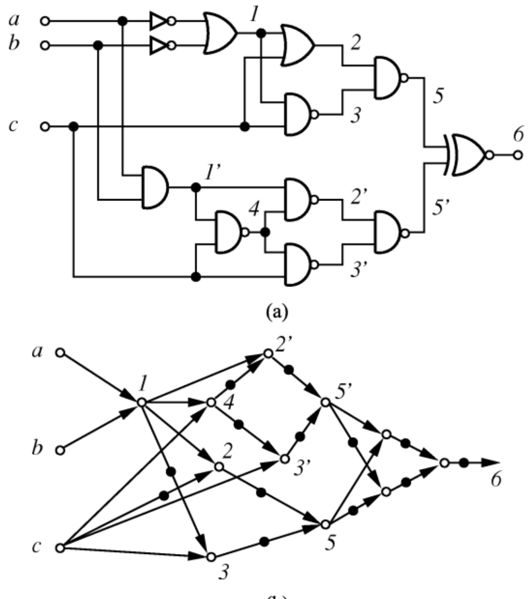
Key Idea: Encoder-Decoder Cross-Modal Alignment

- **Encoder:** pre-trained netlist encoder – capture netlist info.
- **Decoder:** commercial and open-source LLMs – generative capability



Application: Netlist Functional Reasoning

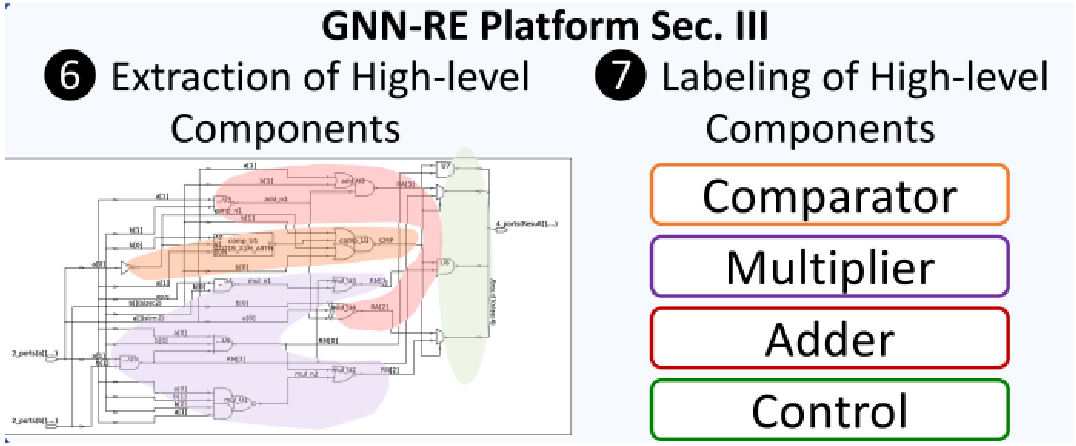
- **Existing netlist functional reasoning**
  - **1. Formal analysis**
    - Formal reasoning (SAT, BDD)
    - Limited scalability
  - **2. GNN-based *gate* function *prediction***
    - Gate *node-level* classification w. label
    - No entire circuit function
    - Cannot handle unseen functions



(a)

(b)

[2] Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. In **TCAD** 2002
[3] Template-based circuit understanding. In **FMCAD** 2014



GNN-RE Platform Sec. III

❻ Extraction of High-level Components

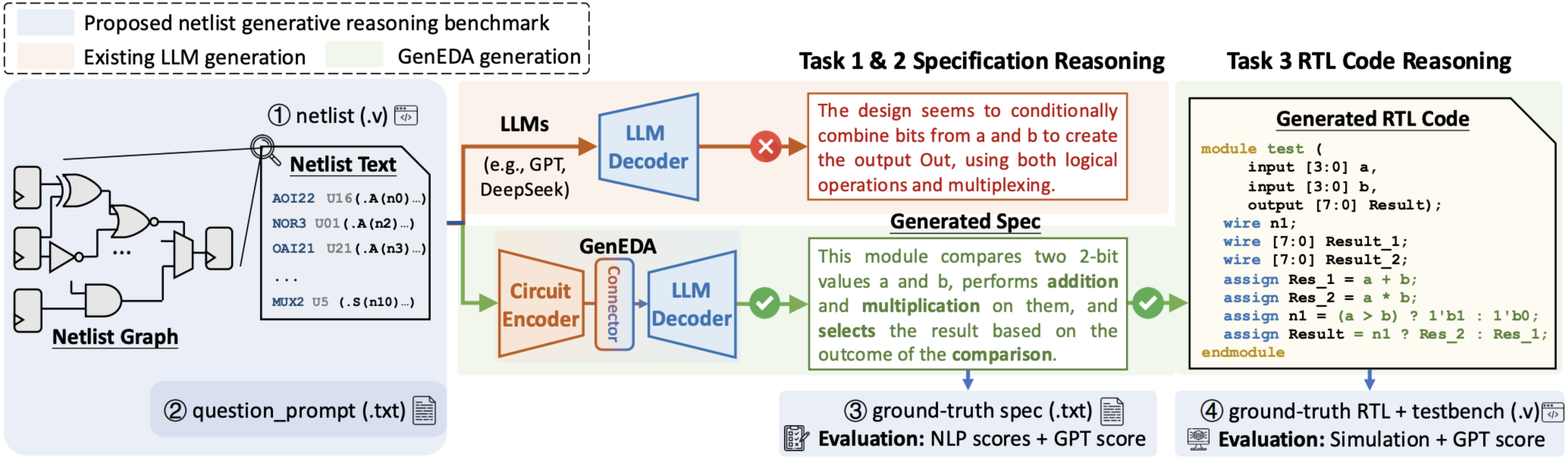
❼ Labeling of High-level Components

- Comparator
- Multiplier
- Adder
- Control

[4] GNN-RE: Graph neural networks for reverse engineering of gate-level netlists. In **TCAD** 2021
[5] Gamora: Graph learning based symbolic reasoning for large-scale Boolean networks. In **DAC** 2023

Application: Generative Functional Reasoning

- **GenEDA enables generative netlist functional reasoning**
  - **Input:** netlist (graph & text)
  - **Output:** specification and RTL code of netlist
- **Advantages:** full circuit generative reasoning, both spec and RTL code



```
AOI22  U16(.A(n0)...)
NOR3  U01(.A(n2)...)
OAI21  U21(.A(n3)...)
...
MUX2 U5  (.S(n10)...)
```

The design seems to conditionally combine bits from a and b to create the output Out, using both logical operations and multiplexing.

This module compares two 2-bit values a and b, performs **addition** and **multiplication** on them, and **selects** the result based on the outcome of the **comparison**.

```
module test (
      input [3:0] a,
      input [3:0] b,
      output [7:0] Result);
   wire n1;
   wire [7:0] Result_1;
   wire [7:0] Result_2;
   assign Res_1 = a + b;
   assign Res_2 = a * b;
   assign n1 = (a > b) ? 1'b1 : 1'b0;
   assign Result = n1 ? Res_2 : Res_1;
endmodule
```

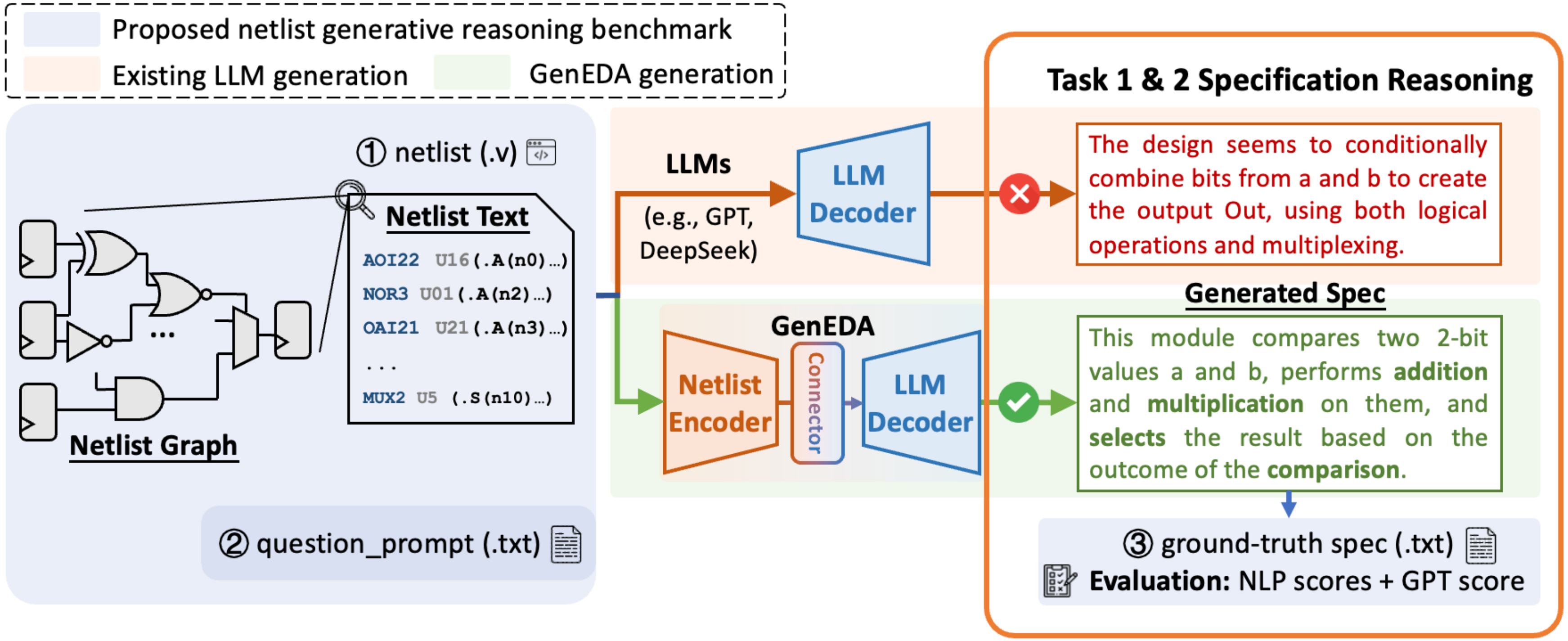
Evaluation: NLP scores + GPT score

Evaluation: Simulation + GPT score

GenEDA Method and Application
- Details

Benchmarking Generative Functional Reasoning

- **Task 1 & 2: specification reasoning from netlists**
  - **Input:** netlist (graph & text)
  - **Output:** specification text
  - **Evaluation:** NLP scores + GPT score



```
AOI22 U16 (.A(n0)…)
NOR3 U01 (.A(n2)…)
OAI21 U21 (.A(n3)…)
. . .
MUX2 U5 (.S(n10)…)
```

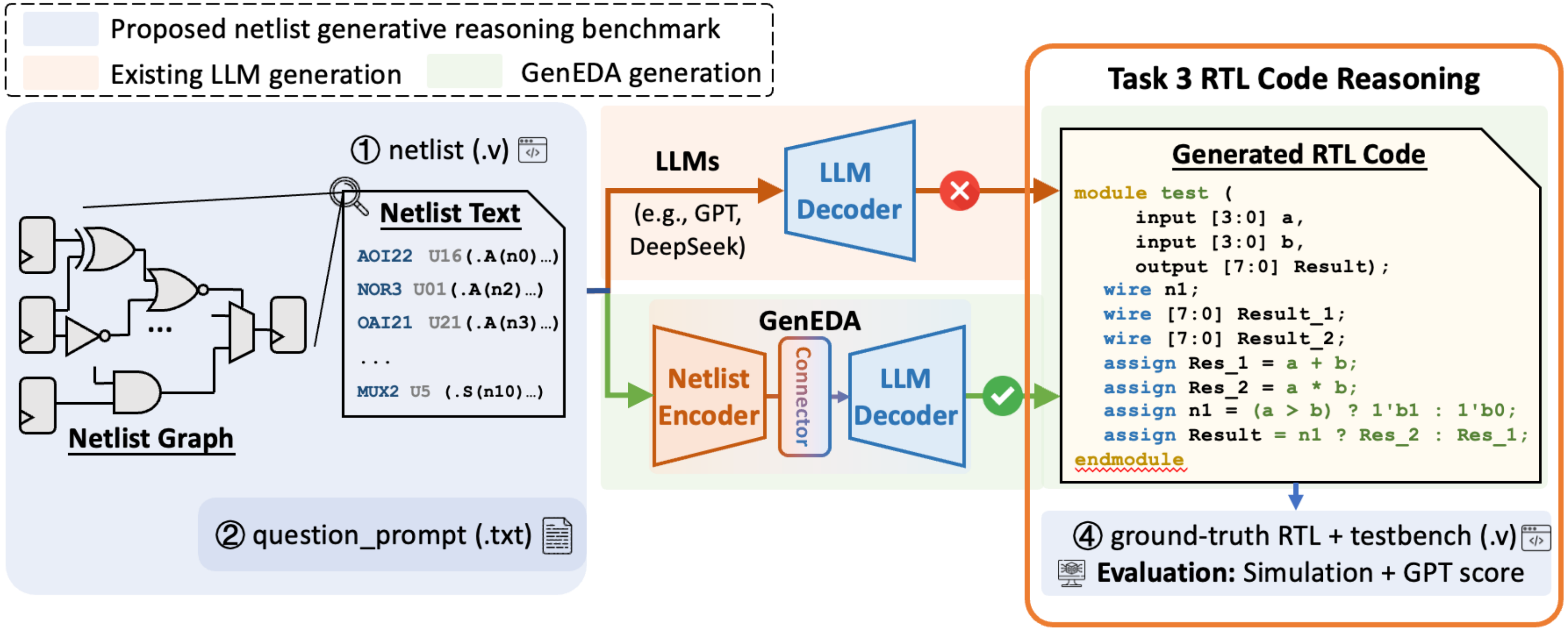
The design seems to conditionally combine bits from a and b to create the output Out, using both logical operations and multiplexing.

This module compares two 2-bit values a and b, performs **addition** and **multiplication** on them, and **selects** the result based on the outcome of the **comparison**.

Evaluation: NLP scores + GPT score

Benchmarking Generative Functional Reasoning

- **Task 3: arithmetic RTL code reasoning from netlists**
  - **Input:** netlist (graph & text)
  - **Output:** RTL code
  - **Evaluation:** simulation for syntax and function, GPT score

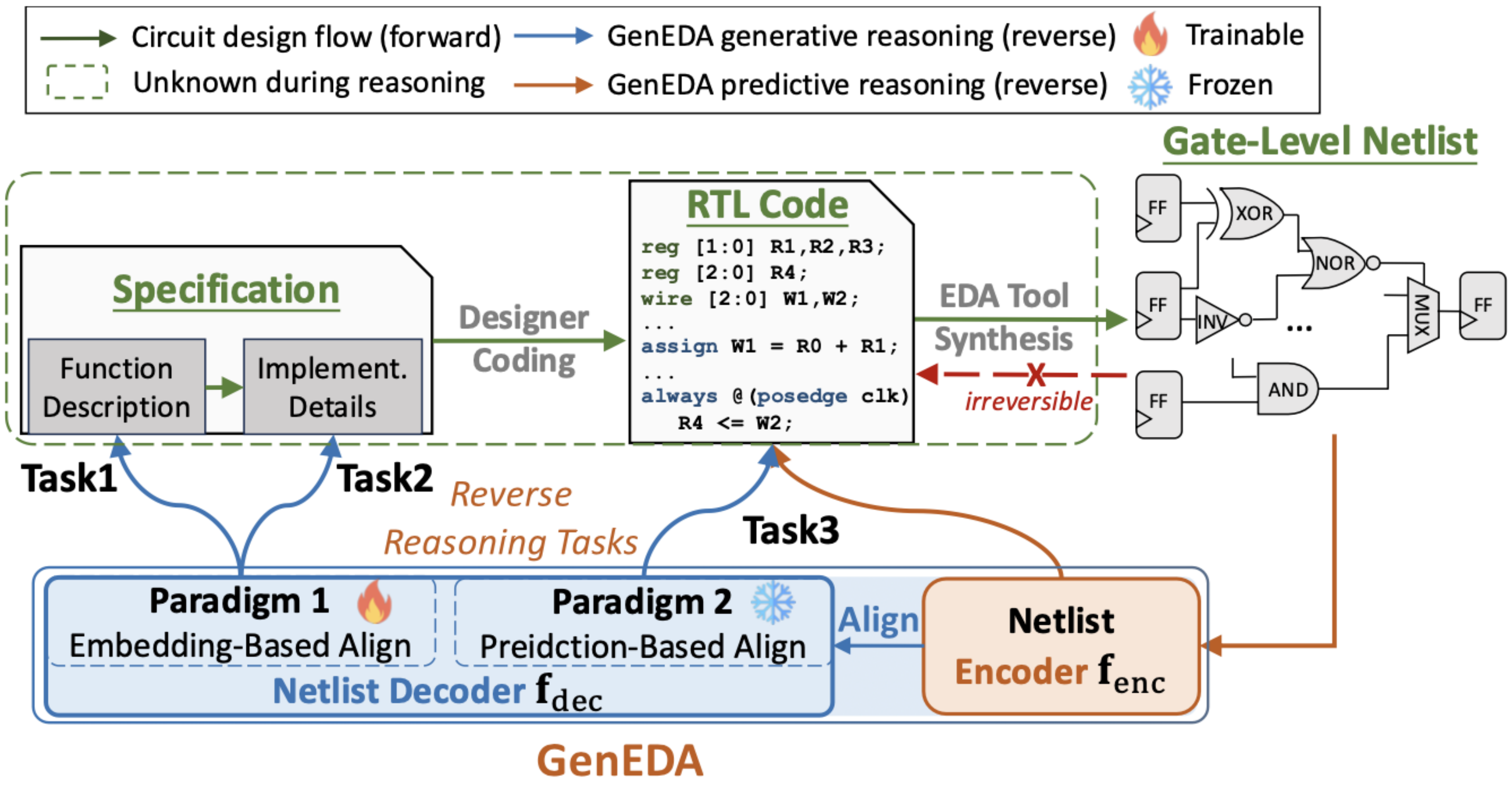


```
AOI22  U16(.A(n0)...)
NOR3  U01(.A(n2)...)
OAI21  U21(.A(n3)...)
...
MUX2 U5  (.S(n10)...)
```

```
module test (
     input [3:0] a,
     input [3:0] b,
     output [7:0] Result);
  wire n1;
  wire [7:0] Result_1;
  wire [7:0] Result_2;
  assign Res_1 = a + b;
  assign Res_2 = a * b;
  assign n1 = (a > b) ? 1'b1 : 1'b0;
  assign Result = n1 ? Res_2 : Res_1;
endmodule
```

GenEDA Encoder-Decoder Alignment

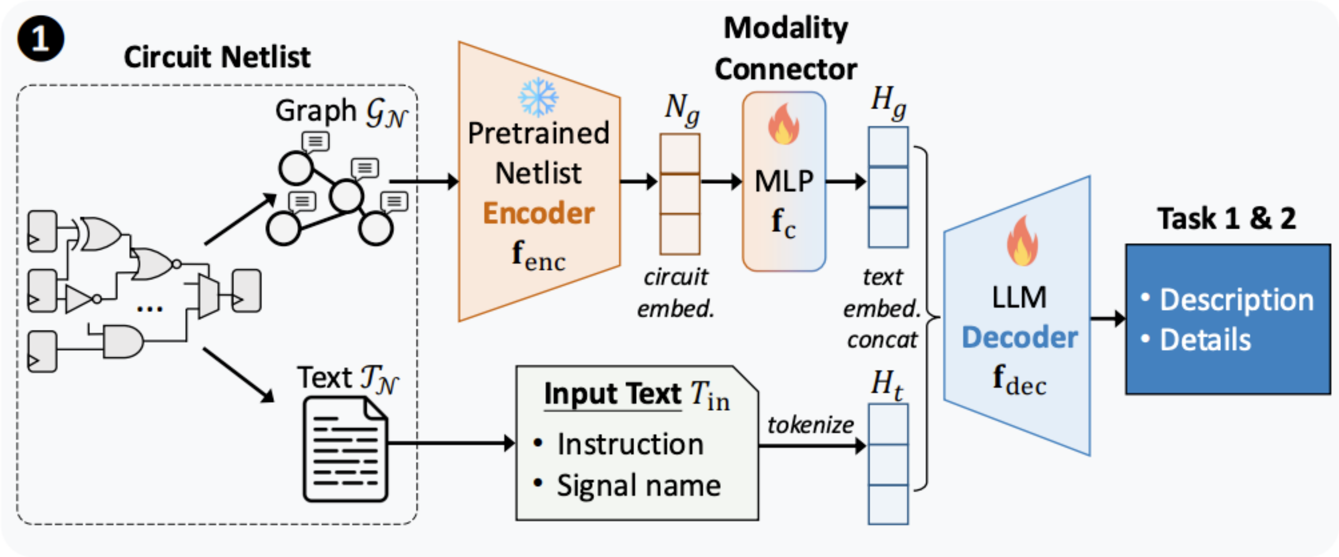
- **Two alignment paradigms**
  - 1. **Embedding**-based for **tunable open-source** LLMs → **Task 1 & 2**
  - 2. **Prediction**-based for **frozen commercial** LLMs → **Task 3**



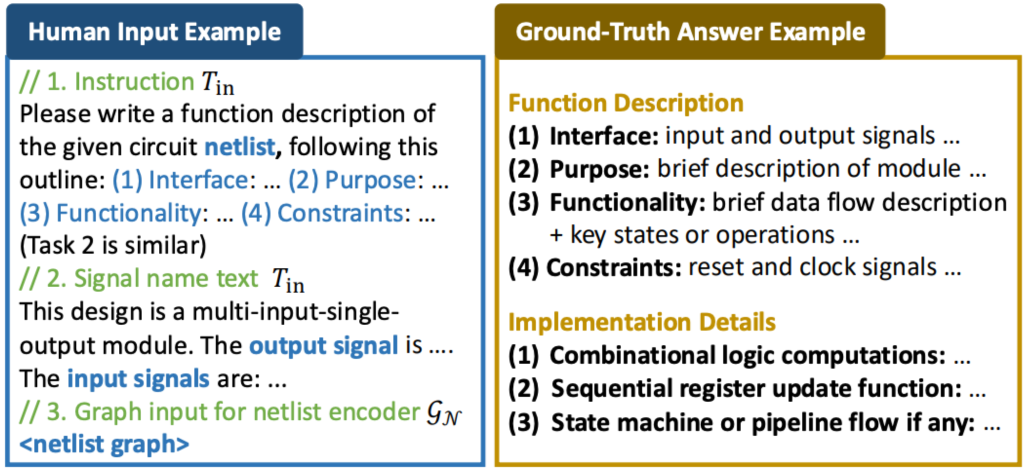
香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

GenEDA Encoder-Decoder Alignment

- **Paradigm 1: embedding-based alignment**
  - Introduce modality connector
  - Fine-tune connector & LLM with instruction tuning


①
Circuit Netlist
Graph $\mathcal{G}_\mathcal{N}$
...
Text $\mathcal{T}_\mathcal{N}$
Pretrained Netlist Encoder $\mathbf{f}_{\mathrm{enc}}$
N_g
circuit embed.
Modality Connector
MLP $\mathbf{f}_{\mathrm{c}}$
H_g
text embed. concat
H_t
tokenize
Input Text T_{in}
• Instruction
• Signal name
LLM Decoder $\mathbf{f}_{\mathrm{dec}}$
Task 1 & 2
• Description
• Details

(a) Paradigm 1: Embedding-based alignment



Human Input Example

// 1. Instruction T_{in}
Please write a function description of the given circuit **netlist**, following this outline: (1) Interface: … (2) Purpose: … (3) Functionality: … (4) Constraints: …
(Task 2 is similar)
// 2. Signal name text T_{in}
This design is a multi-input-single-output module. The **output signal** is …. The **input signals** are: …
// 3. Graph input for netlist encoder $\mathcal{G}_\mathcal{N}$
<netlist graph>

Ground-Truth Answer Example

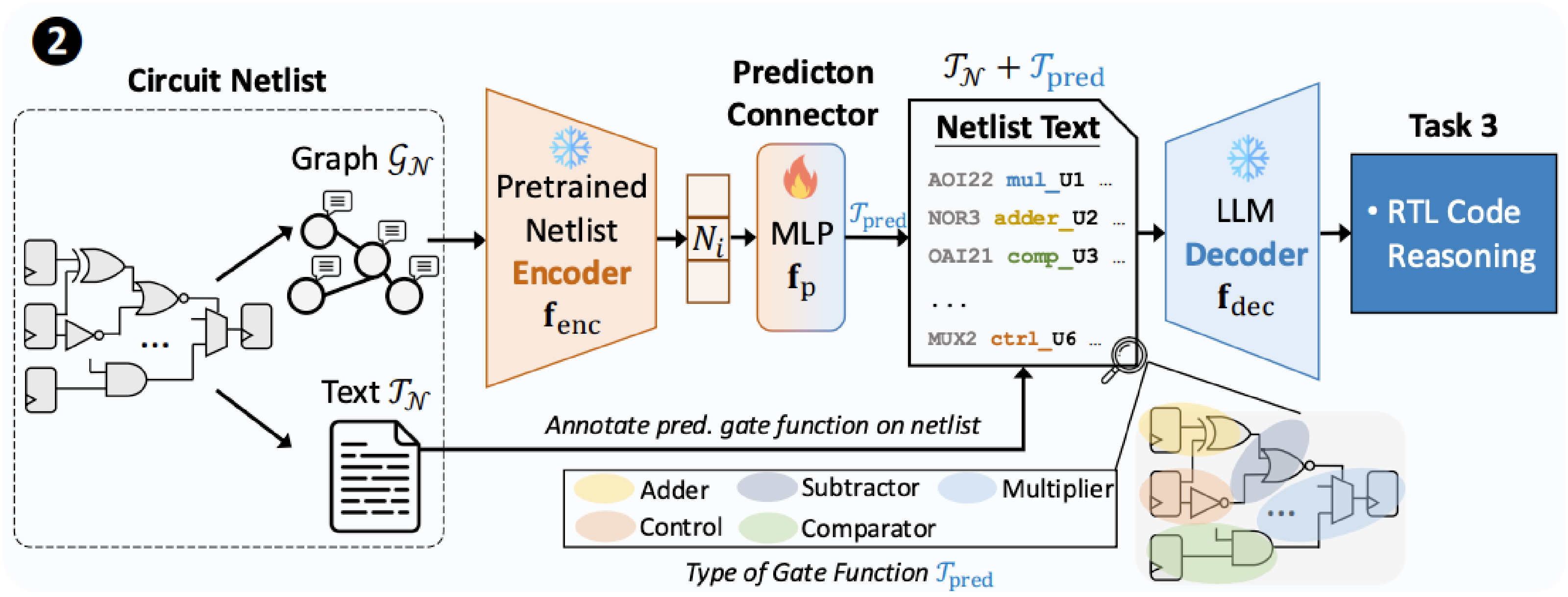
Function Description
(1) Interface: input and output signals …
(2) Purpose: brief description of module …
(3) Functionality: brief data flow description + key states or operations …
(4) Constraints: reset and clock signals …

Implementation Details
(1) Combinational logic computations: …
(2) Sequential register update function: …
(3) State machine or pipeline flow if any: …

Fig. 5: Instruction tuning data pair of alignment paradigm 1.

GenEDA Encoder-Decoder Alignment

- **Paradigm 2: prediction-based alignment**
  - Predict gate function w. encoder
  - Annotate gate predictions on netlist text
  - RTL code reverse generation w. LLM decoder


2
Circuit Netlist
Graph $\mathcal{G}_\mathcal{N}$
Text $\mathcal{T}_\mathcal{N}$
...
Pretrained Netlist Encoder $\mathbf{f}_{\text{enc}}$
N_i
Predicton Connector
MLP $\mathbf{f}_{\text{p}}$
$\mathcal{T}_{\text{pred}}$
$\mathcal{T}_\mathcal{N} + \mathcal{T}_{\text{pred}}$
Netlist Text
AOI22 mul_U1 ...
NOR3 adder_U2 ...
OAI21 comp_U3 ...
...
MUX2 ctrl_U6 ...
LLM Decoder $\mathbf{f}_{\text{dec}}$
Task 3
• RTL Code Reasoning
Annotate pred. gate function on netlist
Adder
Subtractor
Multiplier
Control
Comparator
Type of Gate Function $\mathcal{T}_{\text{pred}}$

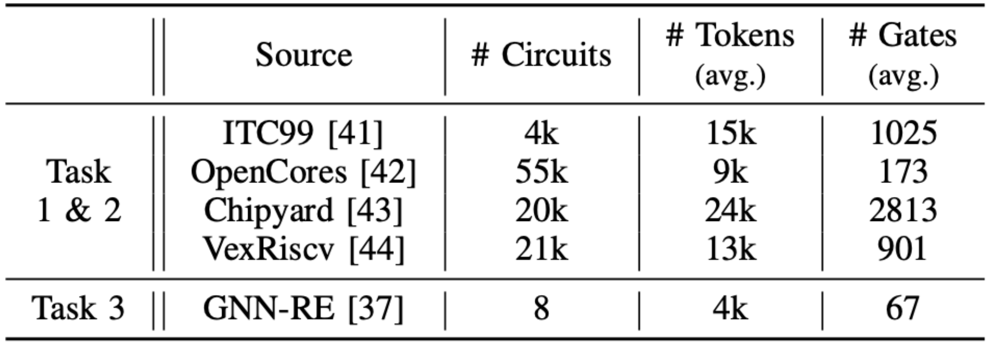
(b) Paradigm 2: Prediction-based alignment

Experimental Results

Experimental Setup

- **Circuit dataset**

TABLE II: Statistics of the netlist dataset.



	Source	# Circuits	# Tokens (avg.)	# Gates (avg.)
Task 1 & 2	ITC99 [41]	4k	15k	1025
	OpenCores [42]	55k	9k	173
	Chipyard [43]	20k	24k	2813
	VexRiscv [44]	21k	13k	901
Task 3	GNN-RE [37]	8	4k	67

- **Model and training**
  - Encoder: NetTAG[6]
  - Decoder:
    - Open-source: DeepSeek-Coder (1/7B)
    - Commercial: GPT-4o, DS-V3
  - Training:
    - DeepSpeed ZeRO and LoRA

[6] *Wenji Fang* et al. NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph, **DAC'25.**

Experimental Results: Task 1 & 2

- **Task 1 & 2: specification reasoning from netlist**
  - Evaluation metrics
    - NLP scores: including BLUE, ROUGE, and embedding similarity
    - GPT scores to evaluate spec similarity via LLMs
  - GenEDA embedding-based alignment improves open-source LLMs

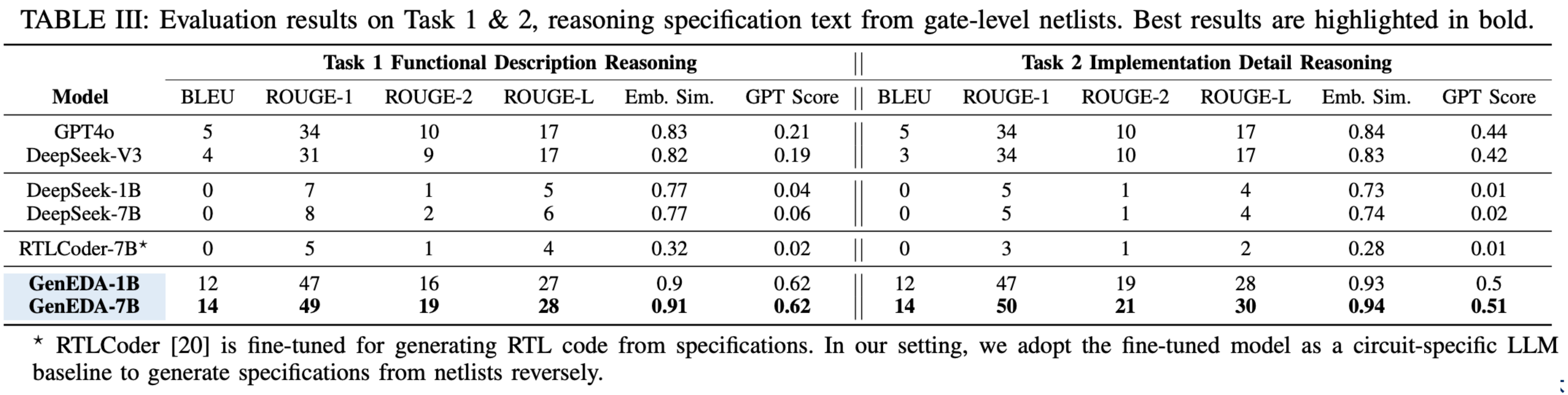


TABLE III: Evaluation results on Task 1 & 2, reasoning specification text from gate-level netlists. Best results are highlighted in bold.

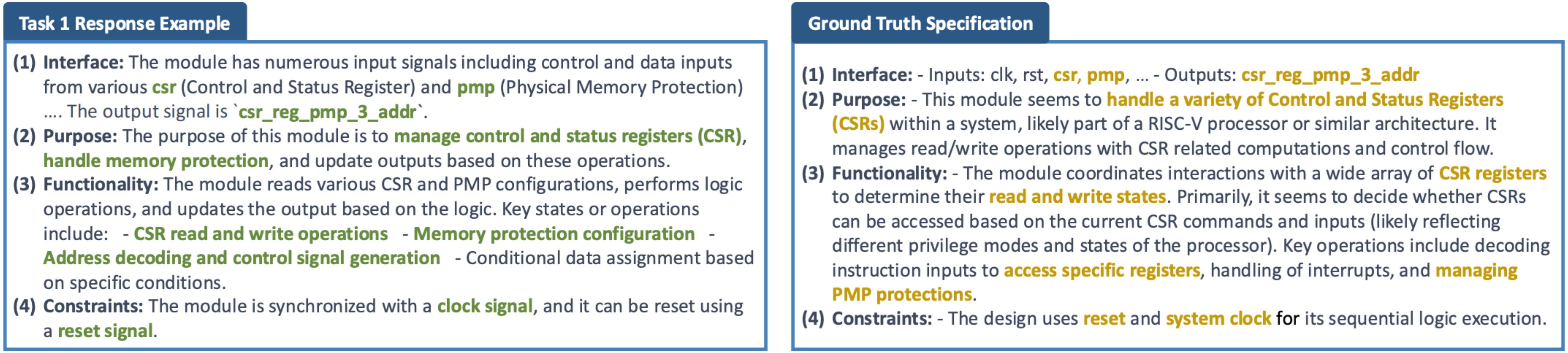
	Task 1 Functional Description Reasoning						**Task 2 Implementation Detail Reasoning**					
Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score
GPT4o	5	34	10	17	0.83	0.21	5	34	10	17	0.84	0.44
DeepSeek-V3	4	31	9	17	0.82	0.19	3	34	10	17	0.83	0.42
DeepSeek-1B	0	7	1	5	0.77	0.04	0	5	1	4	0.73	0.01
DeepSeek-7B	0	8	2	6	0.77	0.06	0	5	1	4	0.74	0.02
RTLCoder-7B*	0	5	1	4	0.32	0.02	0	3	1	2	0.28	0.01
GenEDA-1B	12	47	16	27	0.9	0.62	12	47	19	28	0.93	0.5
GenEDA-7B	**14**	**49**	**19**	**28**	**0.91**	**0.62**	**14**	**50**	**21**	**30**	**0.94**	**0.51**

* RTLCoder [20] is fine-tuned for generating RTL code from specifications. In our setting, we adopt the fine-tuned model as a circuit-specific LLM baseline to generate specifications from netlists reversely.

THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Experimental Results: Task 1 & 2

- Case study for Task 1



Task 1 Response Example

(1) **Interface:** The module has numerous input signals including control and data inputs from various **csr** (Control and Status Register) and **pmp** (Physical Memory Protection) …. The output signal is `csr_reg_pmp_3_addr`.
(2) **Purpose:** The purpose of this module is to **manage control and status registers (CSR)**, **handle memory protection**, and update outputs based on these operations.
(3) **Functionality:** The module reads various CSR and PMP configurations, performs logic operations, and updates the output based on the logic. Key states or operations include: - **CSR read and write operations** - **Memory protection configuration** - **Address decoding and control signal generation** - Conditional data assignment based on specific conditions.
(4) **Constraints:** The module is synchronized with a **clock signal**, and it can be reset using a **reset signal**.

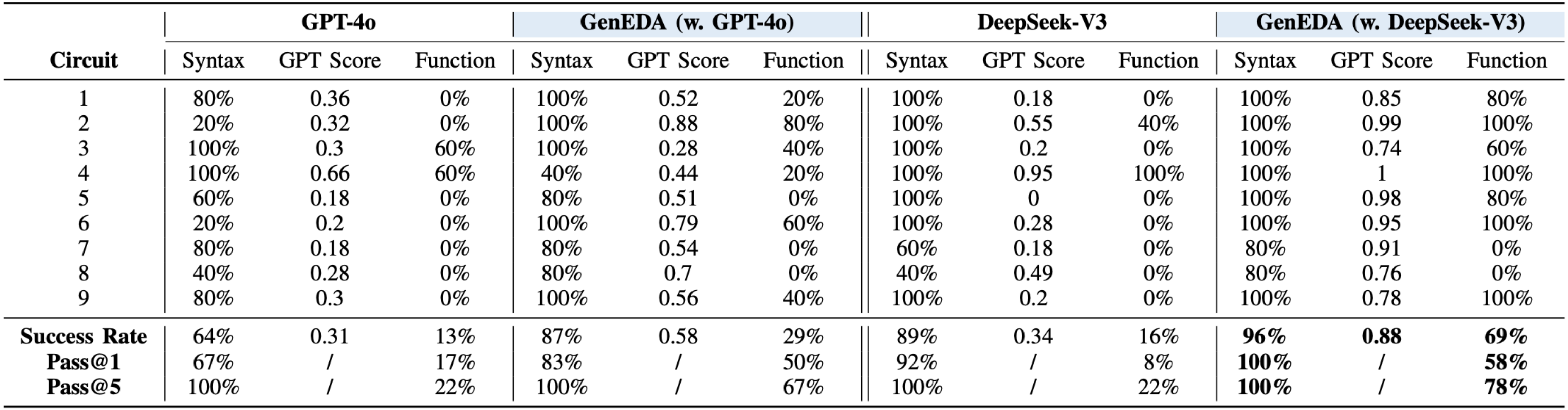
Ground Truth Specification

(1) **Interface:** - Inputs: clk, rst, **csr, pmp**, … - Outputs: **csr_reg_pmp_3_addr**
(2) **Purpose:** - This module seems to **handle a variety of Control and Status Registers (CSRs)** within a system, likely part of a RISC-V processor or similar architecture. It manages read/write operations with CSR related computations and control flow.
(3) **Functionality:** - The module coordinates interactions with a wide array of **CSR registers** to determine their **read and write states**. Primarily, it seems to decide whether CSRs can be accessed based on the current CSR commands and inputs (likely reflecting different privilege modes and states of the processor). Key operations include decoding instruction inputs to **access specific registers**, handling of interrupts, and **managing PMP protections**.
(4) **Constraints:** - The design uses **reset** and **system clock** for its sequential logic execution.

Fig. 6: Case study for Task 1. Comparison between the response from GenEDA and the ground truth specification for a circuit module.

Experimental Results: Task 3

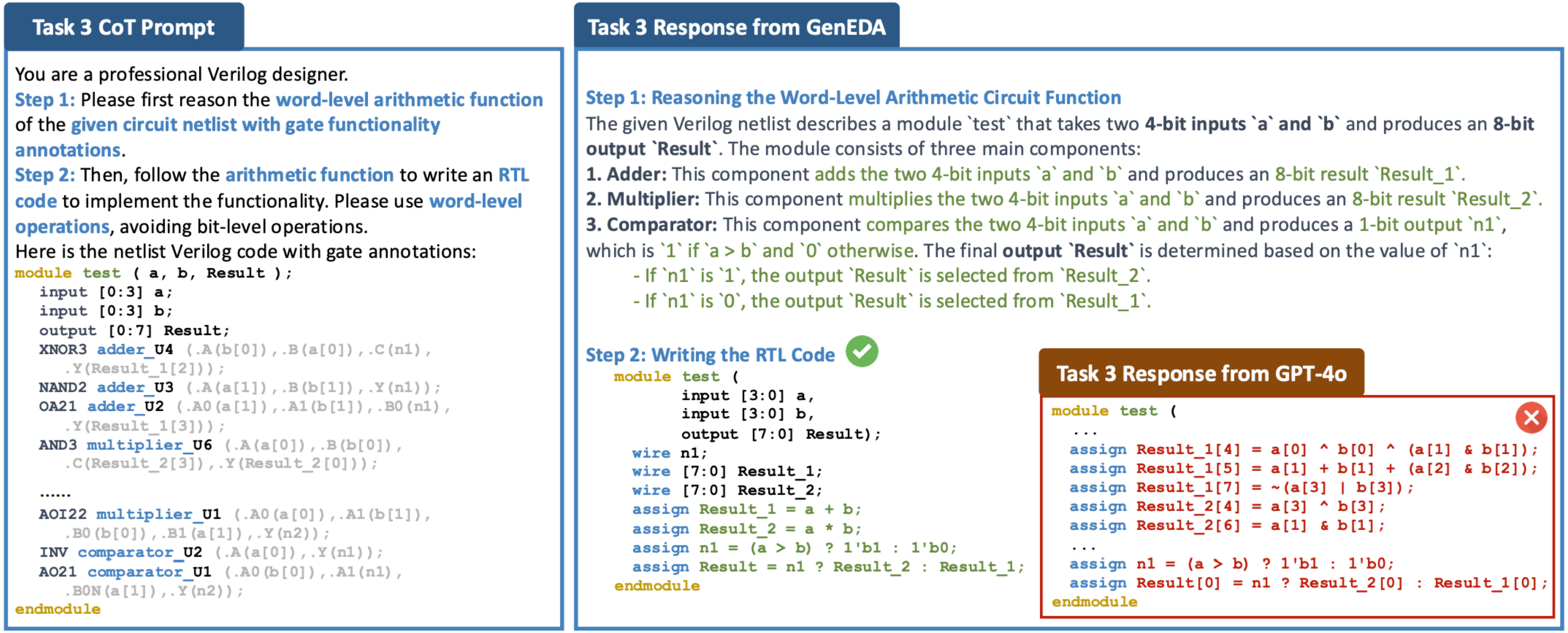
- **Task 3: arithmetic RTL code reasoning from netlist**
  - Evaluation metrics
    - Testbench + VCS simulation: syntax and function correctness
    - GPT score to evaluate code similarity
  - GenEDA prediction-based alignment improves commercial LLMs



	GPT-4o			**GenEDA (w. GPT-4o)**			**DeepSeek-V3**			**GenEDA (w. DeepSeek-V3)**		
Circuit	Syntax	GPT Score	Function	Syntax	GPT Score	Function	Syntax	GPT Score	Function	Syntax	GPT Score	Function
1	80%	0.36	0%	100%	0.52	20%	100%	0.18	0%	100%	0.85	80%
2	20%	0.32	0%	100%	0.88	80%	100%	0.55	40%	100%	0.99	100%
3	100%	0.3	60%	100%	0.28	40%	100%	0.2	0%	100%	0.74	60%
4	100%	0.66	60%	40%	0.44	20%	100%	0.95	100%	100%	1	100%
5	60%	0.18	0%	80%	0.51	0%	100%	0	0%	100%	0.98	80%
6	20%	0.2	0%	100%	0.79	60%	100%	0.28	0%	100%	0.95	100%
7	80%	0.18	0%	80%	0.54	0%	60%	0.18	0%	80%	0.91	0%
8	40%	0.28	0%	80%	0.7	0%	40%	0.49	0%	80%	0.76	0%
9	80%	0.3	0%	100%	0.56	40%	100%	0.2	0%	100%	0.78	100%
Success Rate	64%	0.31	13%	87%	0.58	29%	89%	0.34	16%	**96%**	**0.88**	**69%**
Pass@1	67%	/	17%	83%	/	50%	92%	/	8%	**100%**	/	**58%**
Pass@5	100%	/	22%	100%	/	67%	100%	/	22%	**100%**	/	**78%**

Experimental Results: Task 3

- Case study for Task 3



Task 3 CoT Prompt

You are a professional Verilog designer.
Step 1: Please first reason the **word-level arithmetic function** of the **given circuit netlist with gate functionality annotations**.
Step 2: Then, follow the **arithmetic function** to write an **RTL code** to implement the functionality. Please use **word-level operations**, avoiding bit-level operations.
Here is the netlist Verilog code with gate annotations:

```
module test ( a, b, Result );
   input [0:3] a;
   input [0:3] b;
   output [0:7] Result;
   XNOR3 adder_U4 (.A(b[0]),.B(a[0]),.C(n1),
      .Y(Result_1[2]));
   NAND2 adder_U3 (.A(a[1]),.B(b[1]),.Y(n1));
   OA21 adder_U2 (.A0(a[1]),.A1(b[1]),.B0(n1),
      .Y(Result_1[3]));
   AND3 multiplier_U6 (.A(a[0]),.B(b[0]),
      .C(Result_2[3]),.Y(Result_2[0]));

   ......
   AOI22 multiplier_U1 (.A0(a[0]),.A1(b[1]),
      .B0(b[0]),.B1(a[1]),.Y(n2));
   INV comparator_U2 (.A(a[0]),.Y(n1));
   AO21 comparator_U1 (.A0(b[0]),.A1(n1),
      .B0N(a[1]),.Y(n2));
endmodule
```

Task 3 Response from GenEDA

Step 1: Reasoning the Word-Level Arithmetic Circuit Function

The given Verilog netlist describes a module `test` that takes two **4-bit inputs `a` and `b`** and produces an **8-bit output `Result`**. The module consists of three main components:
1. Adder: This component adds the two 4-bit inputs `a` and `b` and produces an 8-bit result `Result_1`.
2. Multiplier: This component multiplies the two 4-bit inputs `a` and `b` and produces an 8-bit result `Result_2`.
3. Comparator: This component compares the two 4-bit inputs `a` and `b` and produces a 1-bit output `n1`, which is `1` if `a > b` and `0` otherwise. The final **output `Result`** is determined based on the value of `n1`:
- If `n1` is `1`, the output `Result` is selected from `Result_2`.
- If `n1` is `0`, the output `Result` is selected from `Result_1`.

Step 2: Writing the RTL Code ✓

```
module test (
       input [3:0] a,
       input [3:0] b,
       output [7:0] Result);
  wire n1;
  wire [7:0] Result_1;
  wire [7:0] Result_2;
  assign Result_1 = a + b;
  assign Result_2 = a * b;
  assign n1 = (a > b) ? 1'b1 : 1'b0;
  assign Result = n1 ? Result_2 : Result_1;
endmodule
```

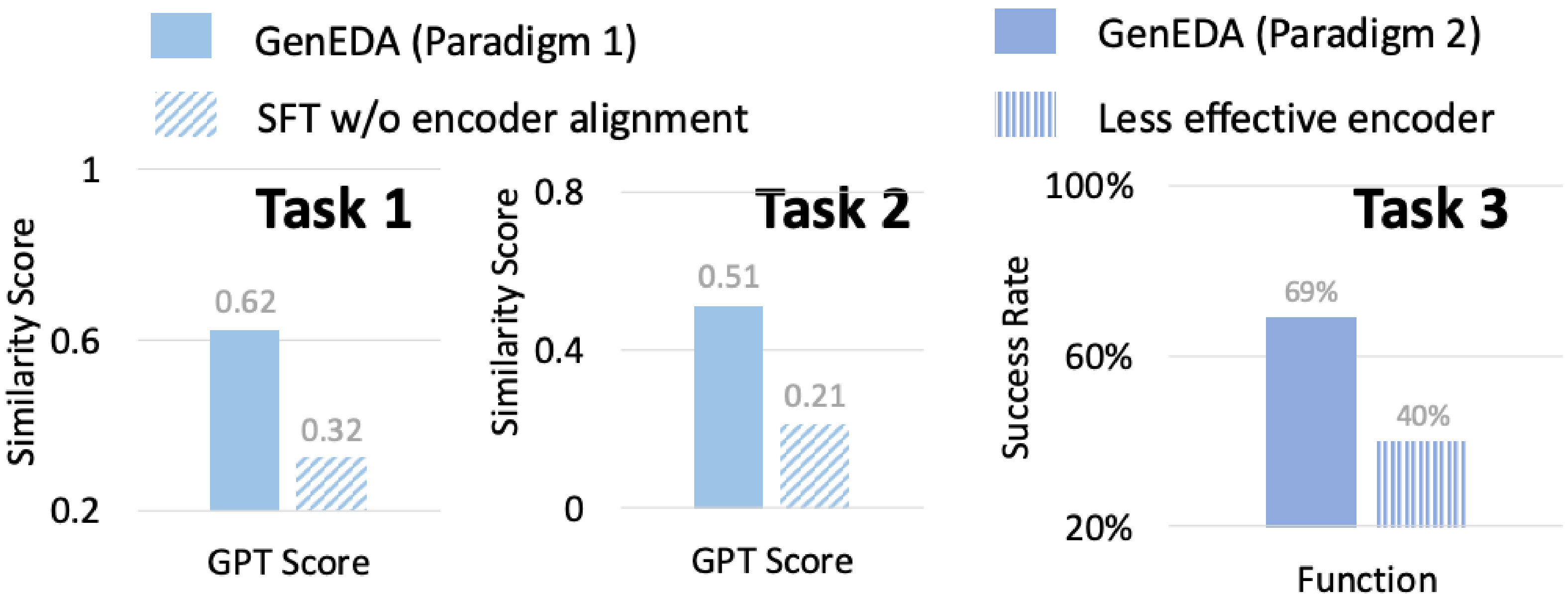
Task 3 Response from GPT-4o ✗

```
module test (
   ...
  assign Result_1[4] = a[0] ^ b[0] ^ (a[1] & b[1]);
  assign Result_1[5] = a[1] + b[1] + (a[2] & b[2]);
  assign Result_1[7] = ~(a[3] | b[3]);
  assign Result_2[4] = a[3] ^ b[3];
  assign Result_2[6] = a[1] & b[1];
   ...
  assign n1 = (a > b) ? 1'b1 : 1'b0;
  assign Result[0] = n1 ? Result_2[0] : Result_1[0];
endmodule
```

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Ablation Study

- Effectiveness of alignment for Task 1 & 2
- Impact of encoder quality for Task 3



(a) Effectiveness of alignment (b) Impact of encoder quality

Conclusion

- **GenEDA: Encoder-Decoder Alignment**
  - Cross-modal alignment for two types of Circuit Foundation Models
  - Enable generative netlist functional reasoning

Paper

Code

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Thanks!
Questions?

Contact: wfang838@connect.ust.hk

About Me