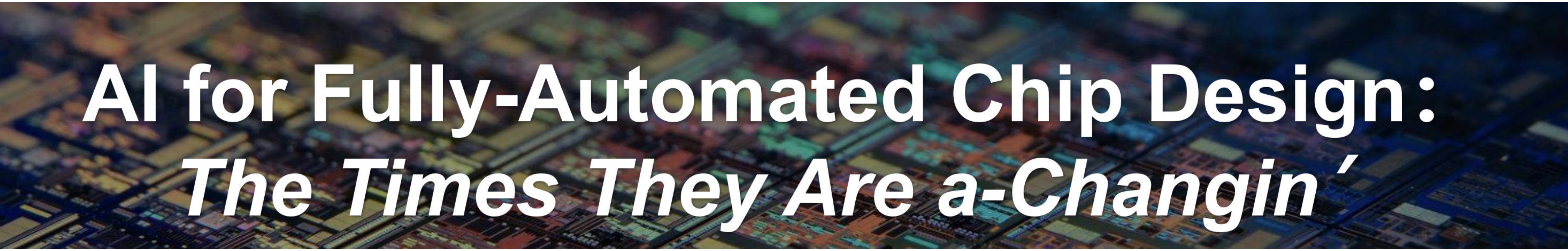


AI for Fully-Automated Chip Design：*The Times They Are a-Changin´*

Institute of Computing Technology，Chinese Academy of Sciences

Xing Hu

2024: Gimmick or Trend

AI for Automated Chip Design: Everything, Everywhere, All at Once (?)
David Pan, The University of Texas at Austin

Automatically generating robotics accelerators
Yuhao Zhu, University of Rochester

Break

A Systematic and rapid approach to design space exploration for tensor accelerators
Qijing Huang, NVIDIA

Empowering Physical Design of VLSI Circuits with Deep Learning: from Modeling to Optimization
Yibo Lin, Peking University

Machine Learning for System-Level Design: Challenges and Opportunities
Andreas Gerstlauer, The University of Texas at Austin

Chip Learning for Processor Design
Zidong Du, Institute of Computing Technology, Chinese Academy of Sciences

Scaling Up the Hardware Design Capability of LLMs: Lessons from the 1st OpenDACs Contest of Processor Design
Cangyuan Li, Institute of Computing Technology, Chinese Academy of Sciences

A High-Level Synthesis Based Framework for Design Space Exploration and Generation of Neural Network Accelerators
Kartik Prabhu, Stanford University

Machine Learning Assisted Memory and Storage System Management
Onur Mutlu, ETH Zurich

2025: The Times They Are a-Changin'

Hypothesizing (Fantasizing) Autonomous Hardware Design
Zhiru Zhang, Cornell University

The Role of AI for Next Generation SW/HW Codesign
Vincent T. Lee, Meta

QiMeng: Automated Hardware and Software Design for Processor Chip
Di Huang, Institute of Computing Technology, Chinese Academy of Sciences

Hair of the Dog: How AI Can Help Formally Verify AI-Designed Chips
Yatin Manerkar, University of Michigan

Learning with Limited Resources: Optimizing Neural Networks for Extreme Efficiency
Radu Marculescu, University of Texas at Austin

Break

hdl2v: A Code Translation Dataset for Enhanced LLM Verilog Generation

Towards an Agile and Autonomous Verification Framework for AI-Generated Chip Designs

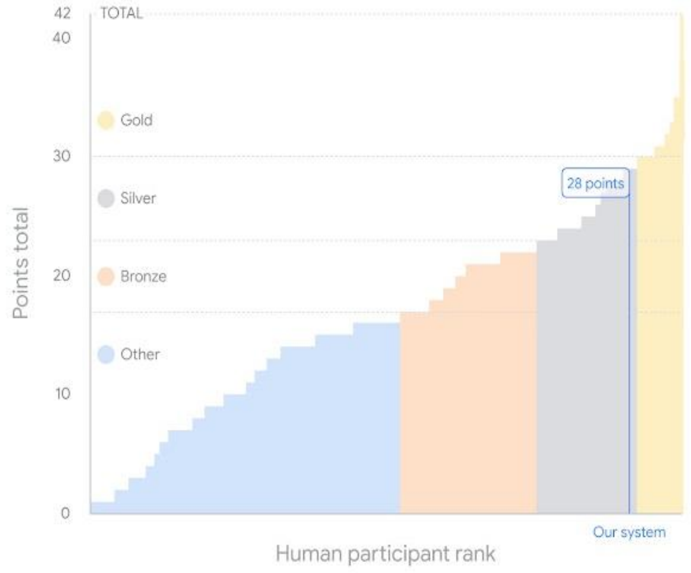
Leveraging Large Language Models for Coverage-Driven Verification of Open-Source RISC-V Cores

ProtocolLLM: RTL Benchmark for SystemVerilog Generation of Communication Protocols

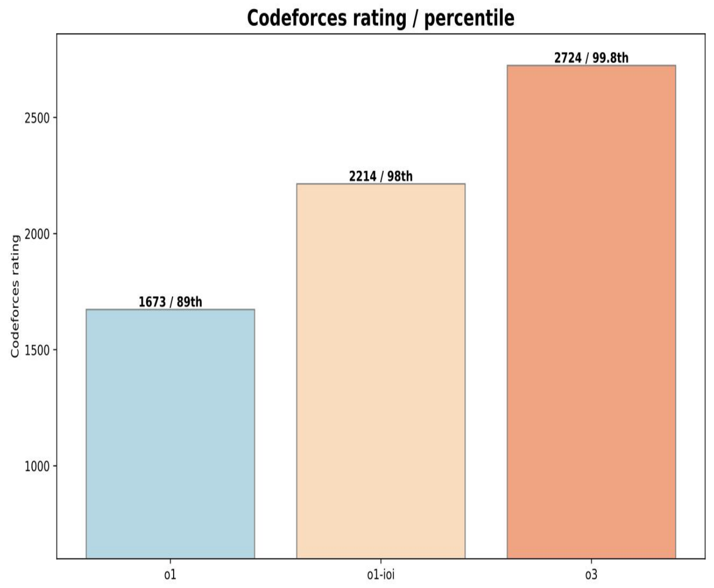
ChiseLLM: Unleashing the Power of Reasoning LLMs for Chisel Agile Hardware Development

The Breakout Year for Reasoning Models

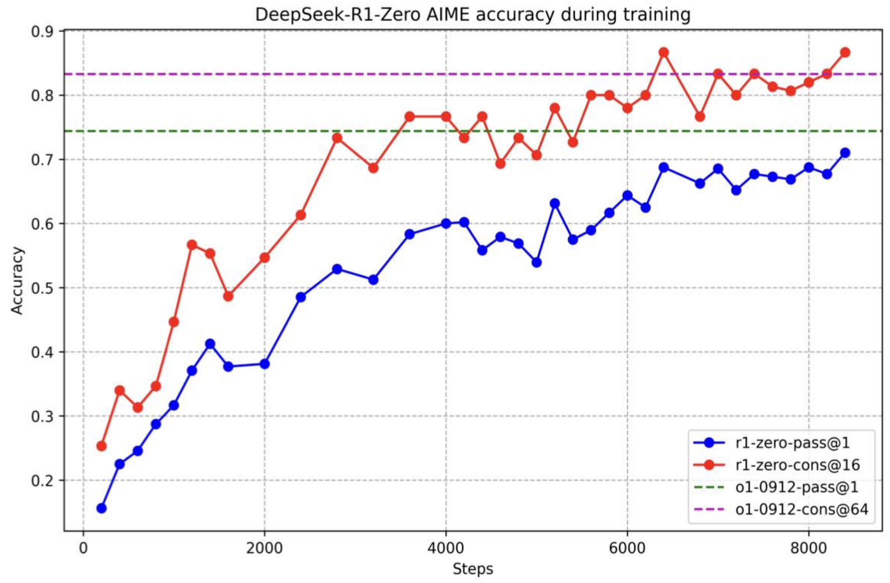
AlphaProof winning a silver medal at the IMO 2024



O3 achieved Gold; 99.8th in Codeforces



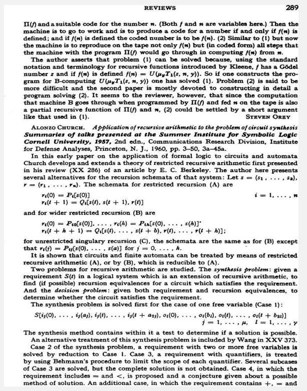
DeepSeek R1: Self-emergence of reasoning



Processor Design: Apex of Logic Reasoning

Alonzo Church

one of the founders of the computer science



REVIEWS 289

Π(*f*) and a suitable code for the number *n*. (Both *f* and *n* are variables here.) Then the machine is to go to work and is to produce a code for a number if and only if *f*(*n*) is defined; and if *f*(*n*) is defined the coded number is to be *f*(*n*). (2) Similar to (1) but now the machine is to reproduce on the tape not only *f*(*n*) but (in coded form) all steps that the machine with the program Π(*f*) would go through in computing *f*(*n*) from *n*.

The author asserts that problem (1) can be solved because, using the standard notation and terminology for recursive functions introduced by Kleene, *f* has a Gödel number *z* and if *f*(*n*) is defined $f(n) = U(\mu_y T_1(z, n, y))$. So if one constructs the program for B-computing $U(\mu_y T_1(z, n, y))$ one has solved (1). Problem (2) is said to be more difficult and the second paper is mostly devoted to constructing in detail a program solving (2). It seems to the reviewer, however, that since the computation that machine B goes through when programmed by Π(*f*) and fed *n* on the tape is also a partial recursive function of Π(*f*) and *n*, (2) could be settled by a short argument like that used in (1).

STEVEN OREY

ALONZO CHURCH. *Application of recursive arithmetic to the problem of circuit synthesis* ***Summaries of talks presented at the Summer Institute for Symbolic Logic Cornell University, 1957,*** 2nd edn., Communications Research Division, Institute for Defense Analyses, Princeton, N. J., 1960, pp. 3–50, 3a–45a.

In this early paper on the application of formal logic to circuits and automata Church develops and extends a theory of restricted recursive arithmetic first presented in his review (XX 286) of an article by E. C. Berkeley. The author here presents several alternatives for the recursion schemata of that system: Let $s = (s_1, \ldots, s_k)$, $r = (r_1, \ldots, r_n)$. The schemata for restricted recursion (A) are

$$r_i(0) = P_i[s(0)] \qquad i = 1, \ldots, n$$
$$r_i(t+1) = Q_i[s(t), s(t+1), r(t)]$$

and for wider restricted recursion (B) are

$$r_i(0) = P_{i0}[s(0)], \ldots, r_i(h) = P_{ih}[s(0), \ldots, s(h)]^*$$
$$r_i(t+h+1) = Q_i[s(t), \ldots, s(t+b), r(t), \ldots, r(t+h)];$$

for unrestricted singulary recursion (C), the schemata are the same as for (B) except that $r_i(j) = P_{ij}[s(0), \ldots, s(a)]$ for $j = 0, \ldots, h$.

It is shown that circuits and finite automata can be treated by means of restricted recursive arithmetic (A), or by (B), which is reducible to (A).

Two problems for recursive arithmetic are studied. The *synthesis problem*: given a requirement $S(t)$ in a logical system which is an extension of recursive arithmetic, to find (if possible) recursion equivalences for a circuit which satisfies the requirement. And the *decision problem*: given both requirement and recursion equivalences, to determine whether the circuit satisfies the requirement.

The synthesis problem is solved first for the case of one free variable (Case 1):

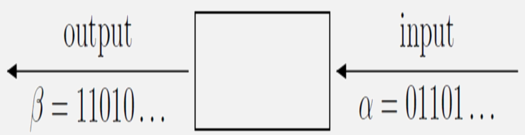
$$S[i_j(0), \ldots, i_j(a_j), i_j(t), \ldots, i_j(t+a_{2j}), o_l(0), \ldots, o_l(b_l), o_l(t), \ldots, o_l(t+b_{2l})]$$
$$j = 1, \ldots, \mu, \quad l = 1, \ldots, \gamma$$

The synthesis method contains within it a test to determine if a solution is possible.

An alternative treatment of this synthesis problem is included by Wang in XXV 373.

Case 2 of the synthesis problem, a requirement with two or more free variables is solved by reduction to Case 1. Case 3, a requirement with quantifiers, is treated by using Behmann's procedure to limit the scope of each quantifier. Several subcases of Case 3 are solved, but the complete solution is not obtained. Case 4, in which the requirement includes = and <, is proposed and a conjecture given about a possible method of solution. An additional case, in which the requirement contains +, = and

[A. Church 1957]

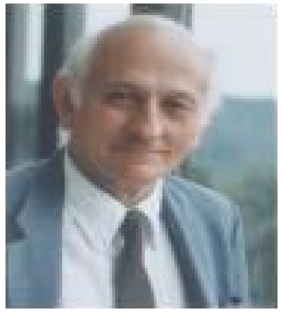


Church’ s Problem: Given input and output bitstreams α and β, automatically construct a circuit design that describes the input-output relationship.

1957: The Church’ s problem of automatic circuit design

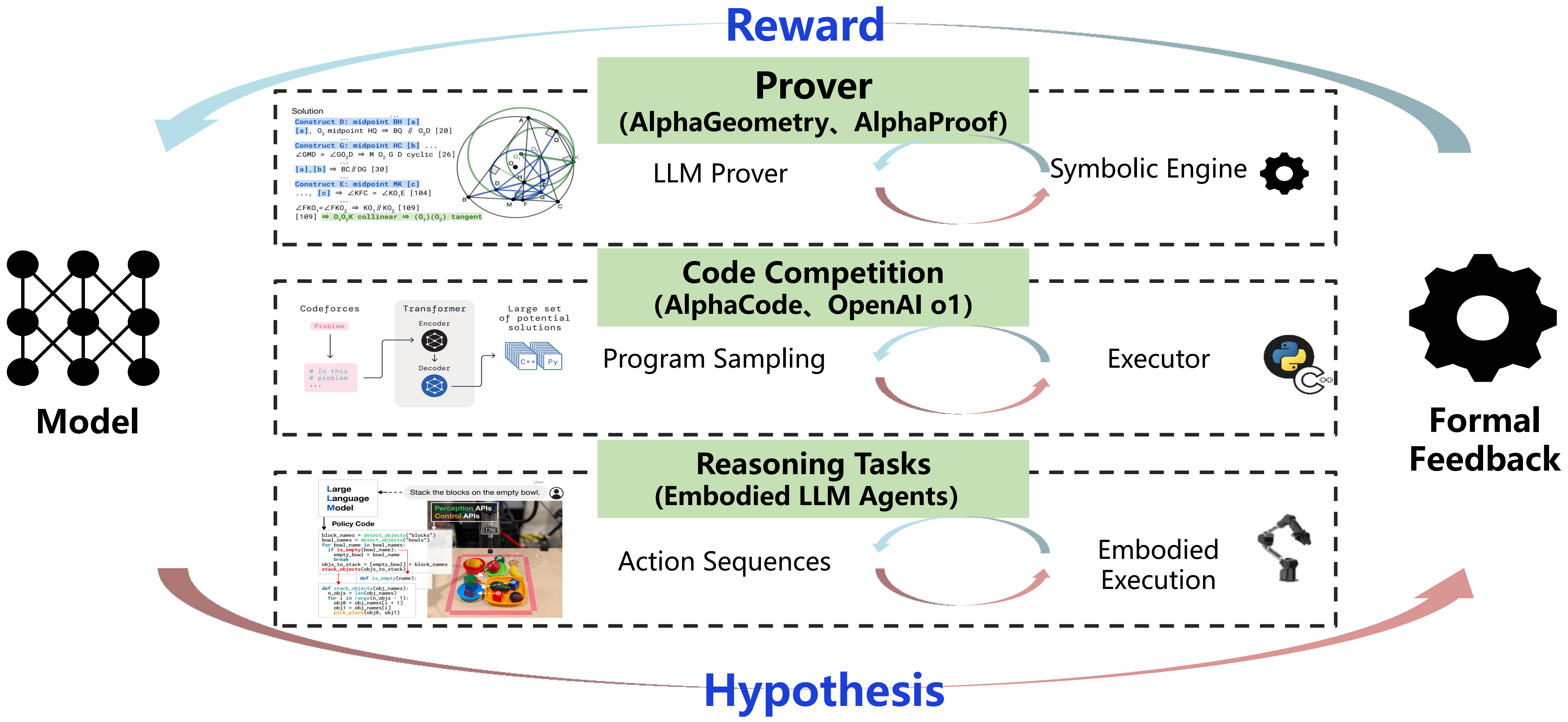
R. Floyd
Automatic program synthesis and verification

J. Hopcroft
Logic synthesis



J. Cocke
Circuit design

Pradigm of Reasoning Tasks



Solution

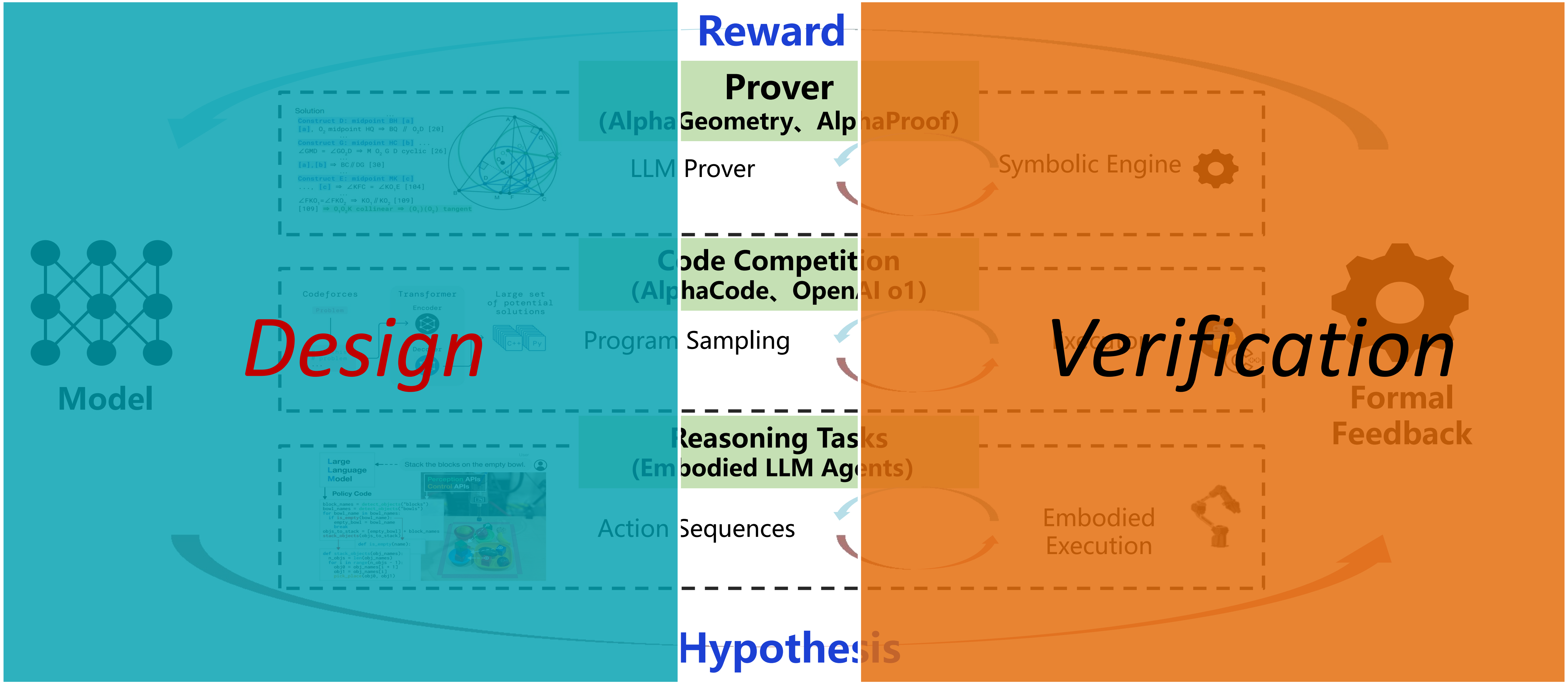
Construct D: midpoint BH [a]
[a], O_2 midpoint HQ ⇒ BQ // O_2D [20]
...
Construct G: midpoint HC [b] ...
∠GMD = ∠GO_2D ⇒ M O_2 G D cyclic [26]
...
[a],[b] ⇒ BC // DG [30]
...
Construct E: midpoint MK [c]
..., [c] ⇒ ∠KFC = ∠KO_1E [104]
...
∠FKO_1=∠FKO_2 ⇒ KO_1 // KO_2 [109]
[109] ⇒ O_1O_2K collinear ⇒ (O_1)(O_2) tangent

```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```

Pradigm of Reasoning Tasks



Talk Overview

SW/HW Co-design

The Role of AI for Next Generation SW/HW Codesign

Design

ProtocolLLM: RTL Benchmark for SystemVerilog Generation of Communication Protocols

ChiseLLM: Unleashing the Power of Reasoning LLMs for Chisel Agile Hardware Development

Feedback-driven Loop

Hypothesizing (Fantasizing) Autonomous Hardware Design

QiMeng: Automated Hardware and Software Design for Processor Chip

Verification

Hair of the Dog: How AI Can Help Formally Verify AI-Designed Chips

Leveraging Large Language Models for Coverage-Driven Verification of Open-Source RISC-V Cores

Boosting

Boosting

Data, Infrastructure, Tools

hdl2v: A Code Translation Dataset for Enhanced LLM Verilog Generation

Learning with Limited Resources: Optimizing Neural Networks for Extreme Efficiency

Towards an Agile and Autonomous Verification Framework for AI-Generated Chip Designs