AI Meets Formal: Practical Applications from Industry Leaders

Xiaolin Chen, Synopsys
Pradip Prajapati, Synopsys
Ghaith Bany Hamad, Nvidia
Bindumadhava S S, Google

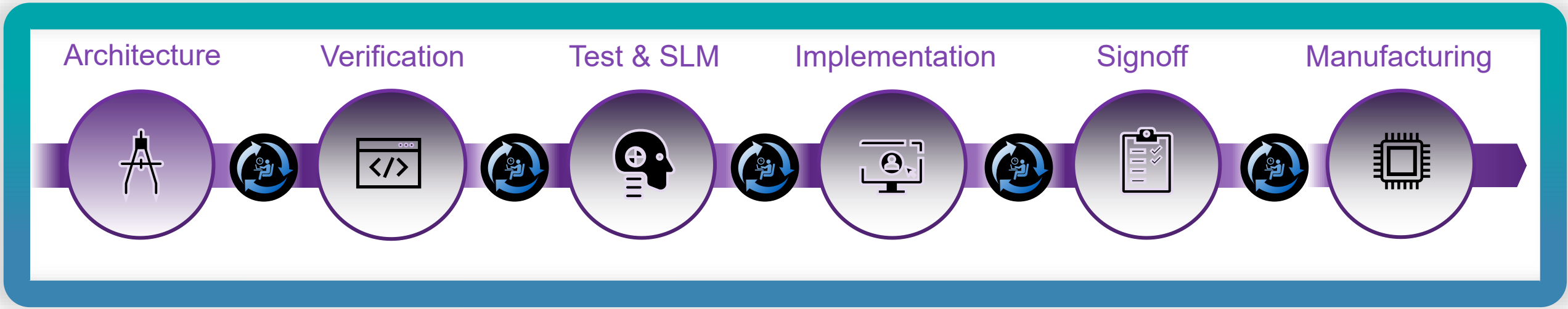
Agenda

- Synopsys AI Solutions and Formal Advisor in VC Formal (Synopsys)
- Usage Example Property Generation (Synopsys)
- Usage Example in Agentic Formal Flow (Nvidia)
- AI Deployment Best Practices (Google)
- Concluding Remarks (Synopsys)
- Q&A

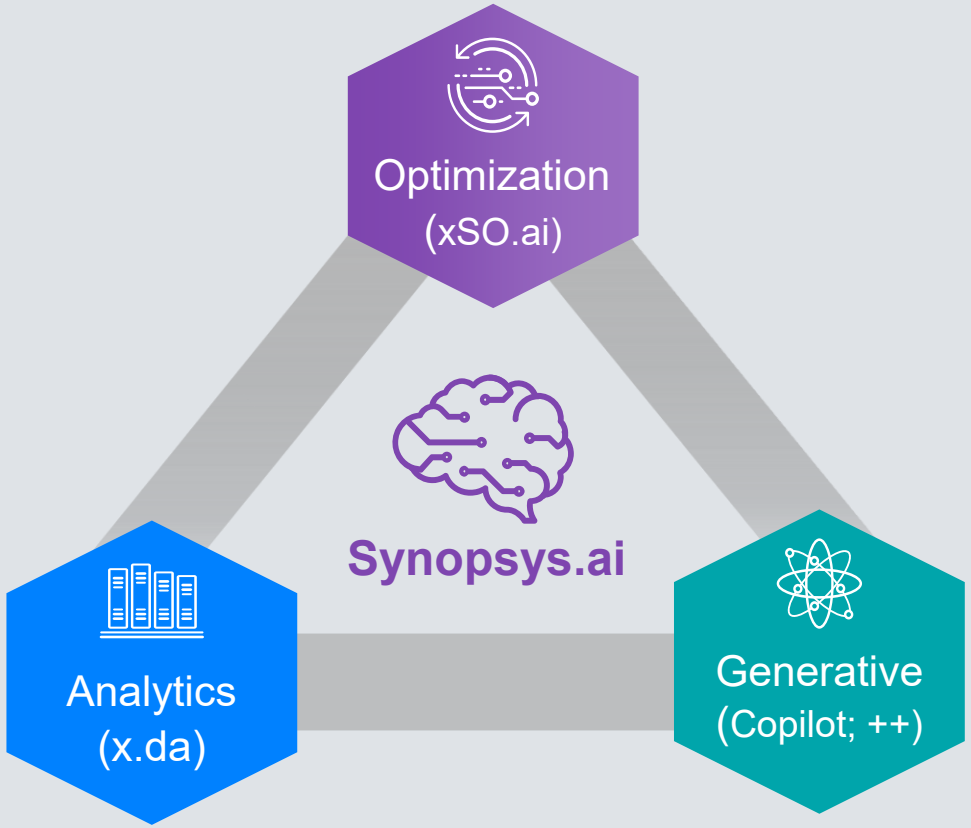
Synopsys AI Solutions and Formal Advisor in VC Formal

Xiaolin Chen, Synopsys

AI Can Accelerate Every Step of the EDA Workflow



Synopsys Pioneering AI Driven Design



AI-Driven Optimization

Better, Faster Results

DSO.ai, VSO.ai, TSO.ai, ASO.ai, 3DSO.ai

AI-Driven Analytics

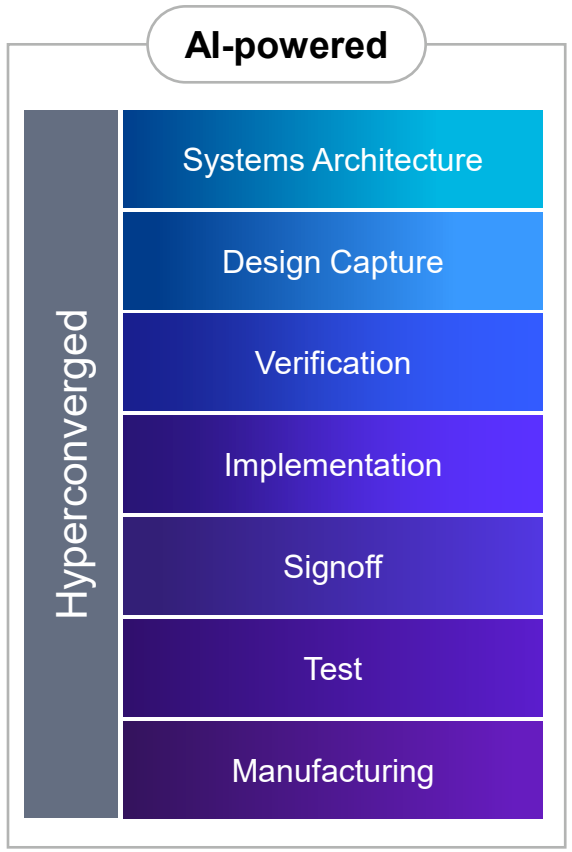
Fast Analysis of Data for Real-Time Insights

Design.da, Fab.da, Silicon.da

Generative AI

Powered by LLMs for Collaborative, Generative and Autonomous Capabilities

Synopsys.ai : Pioneering AI-Driven Design Across Full Stack



Generative AI

Collaborative	Generative	Autonomous
Expert guidance on tools, methodologies	Design collateral creation, optimization RTL, Test Bench, Formal & more	Agentic Workflow Automation

AI-driven data analytics

Design.da	Silicon.da	Fab.da
Better PPA from aggregated insights	Higher yield and quality	Higher fab efficiency

AI-driven optimization

DSO.ai™	VSO.ai™	TSO.ai™	ASO.ai™	3DSO.ai™
10% better performance per watt	2X Improvement in HW Utilization	25% average pattern count reduction	3x TAT for node migration	10% better QoR, weeks-to-days TAT

Synopsys LLM-Based GenAI Vision for Chip Design

Assistive

Assistive Features automating EDA workflow

Knowledge Assistant

Workflow Assistant

Run Assistant

Creative

Content Creation using Designer Assistant

RTL Assistant

Verification Assistant

Agentic

Automated workflows and content creation

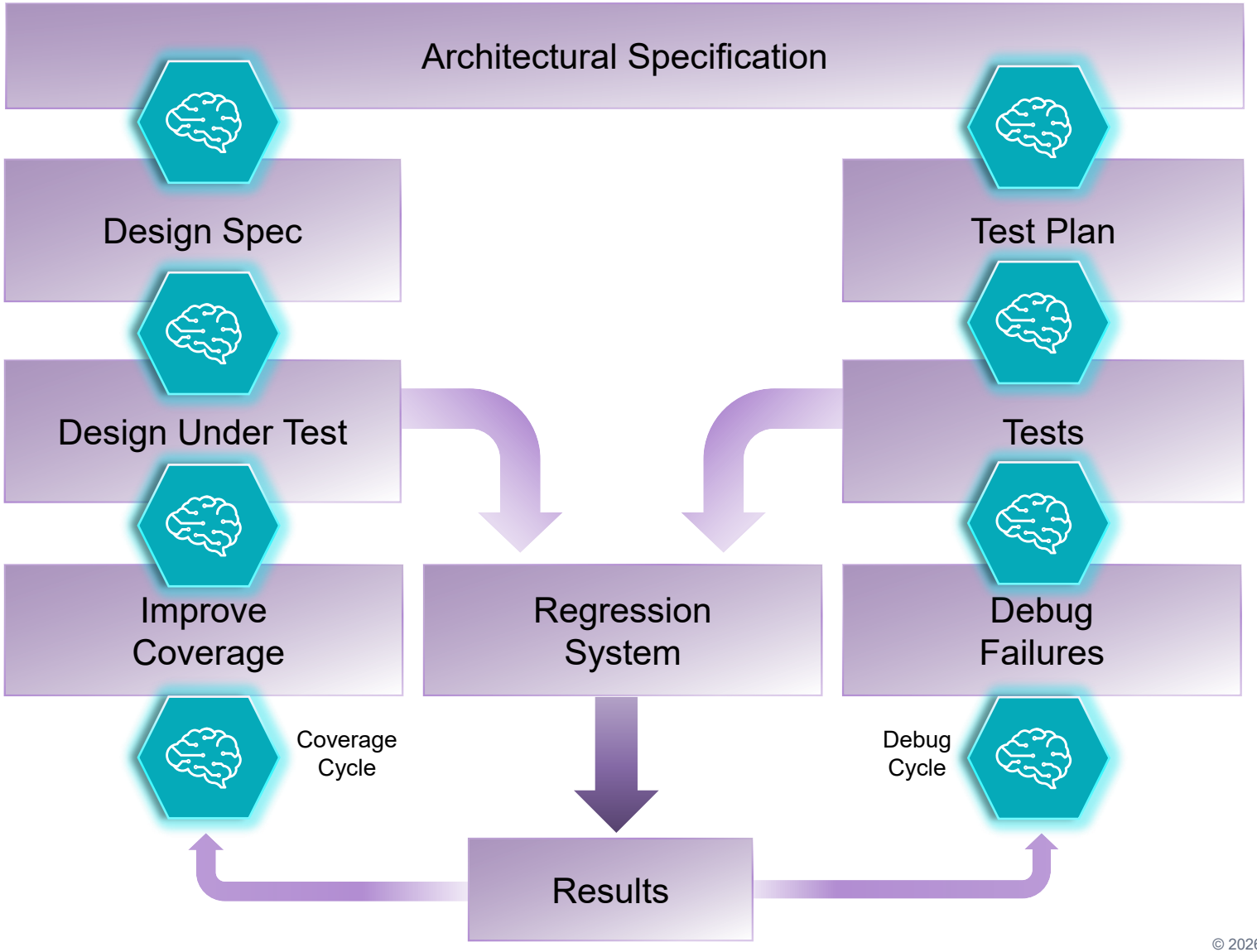
Agentic workflow automation

Design creation

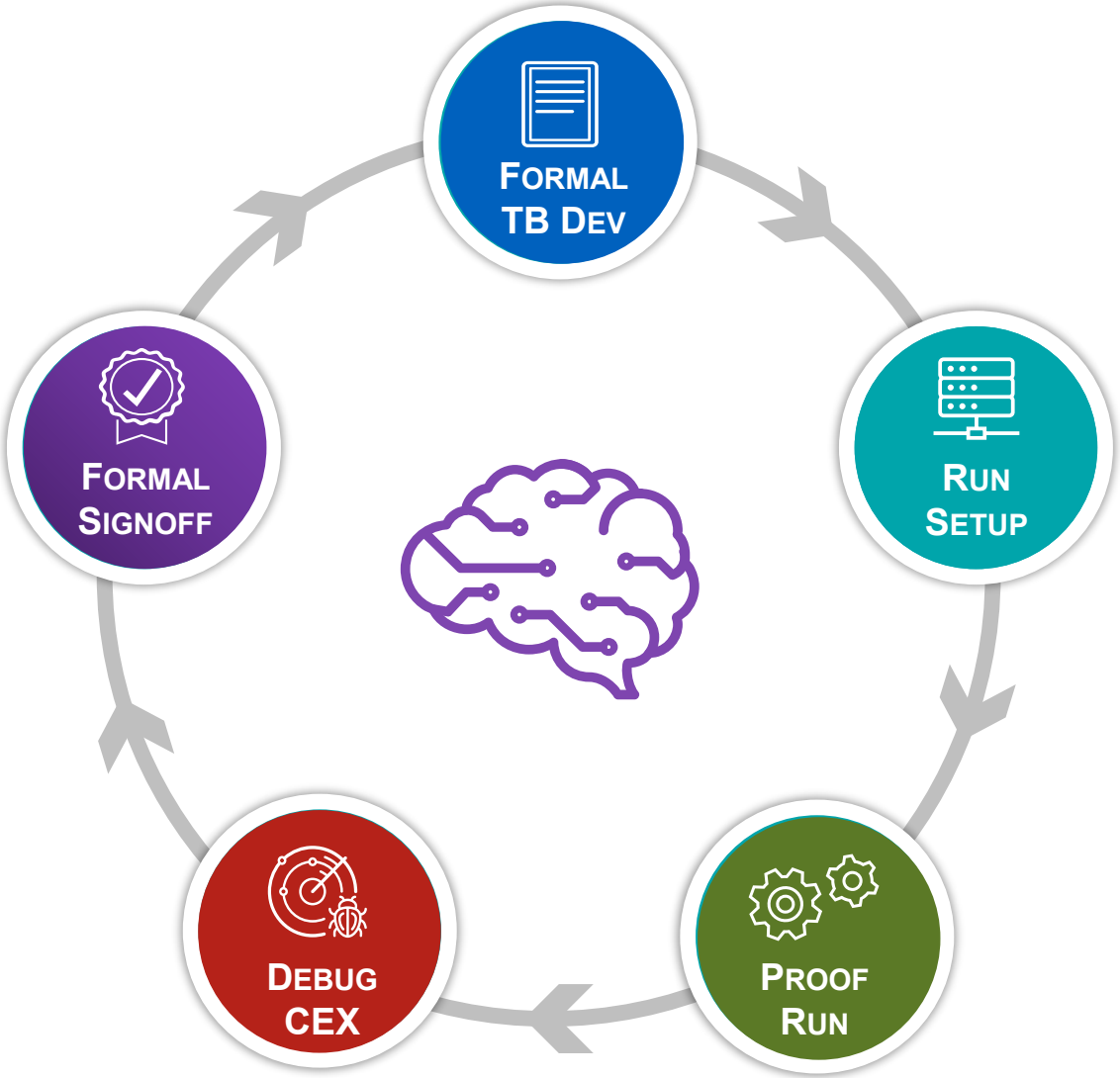
On-Prem / Cloud Enabled Optionality

Opportunities for GenAI in Verification

Verification



Synopsys VC Formal Pervasive AI Vision

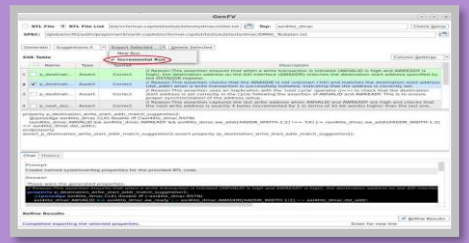


SoC / IP Design Specification

RTL File(s) or Filelist

Formal Verification Testplan

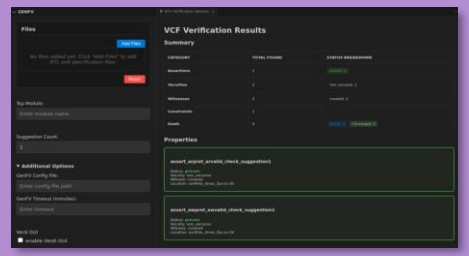
Formal Advisor



Interactive Verdi GUI

```
# User needs to source this TCL Script so that GenFV TCL commands are available in the environment.
# This will internally run
# 1_InstantiateDv and 1_1p_initialize commands which are required to initialize the GenFV environment.
source "$::env(VERDI_GENAI_HOME)/tcl/genfv_source.tcl"

# rtl_path and top_module are required arguments.
# is_rtl_list is optional, if not given, it will be set to 1
genfv::check_design -rtl_path {./rtllist.txt} -top_module axi4lite_dmac

# All the options are optional.
genfv::generate_result

# All the options are optional.
genfv::update_sva

# All the options are optional.
genfv::save_result
```

Batch Mode Automation using Tcl



VSCode Extension with RTL Assistant

FORMAL ADVISOR

Synopsys | Customer

LLM

LLM Gateway
Cloud Access Endpoint

Commercial + In-House + Open Source LLMs Using Agentic Property Generation

Azure, OpenAI, Gemini,
AWS Bedrock, Llama, TGI, etc.

Query

Retrieve

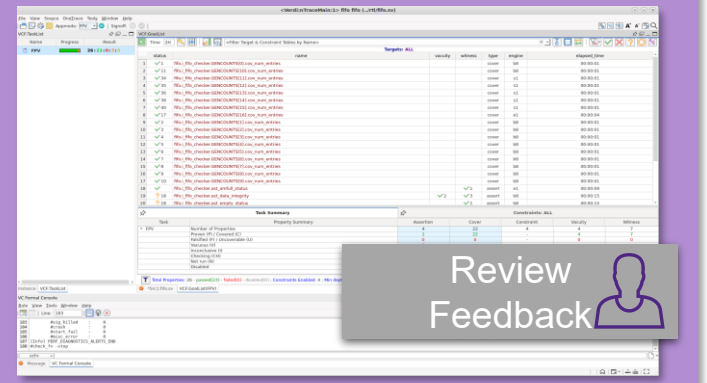
Retrieval Augmented Generation SVA Database (Recommended)

SVA
Asserts
Assumes
Covers

Reranking
Embedding
Auxiliary Code
Context and Intent

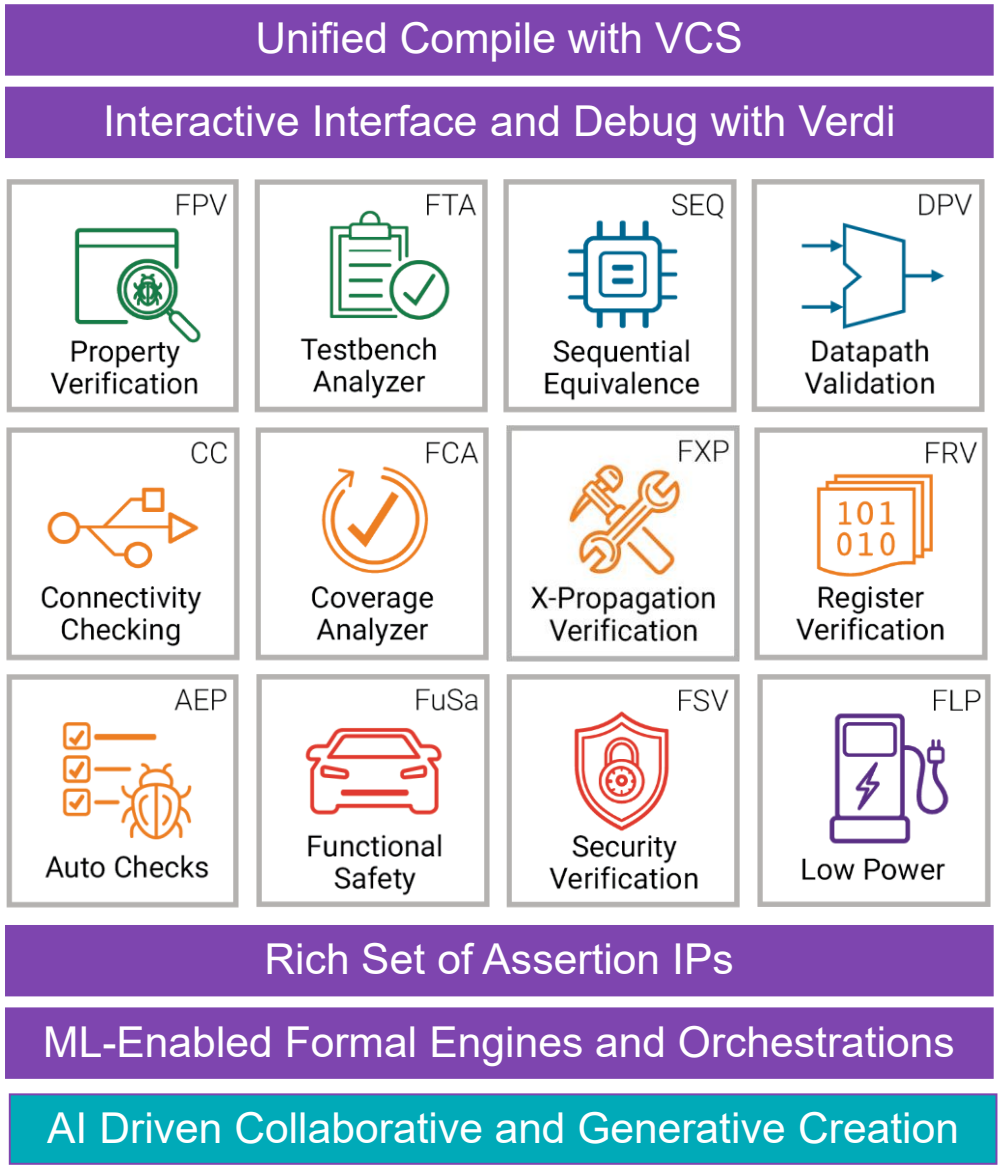
Update

VC Formal FPV



Run FPV on generated SVAs
Generate Properties from New or Existing Setup
Update RAG with high quality passing SVAs
Generate Helper Properties / Invariants

Synopsys VC Formal – Leading Formal Innovations



Industry's Fastest Growing Formal Solution!

Deliver highest performance

Innovative formal engines and ML-based orchestrations find more bugs and achieve more proofs on larger designs

Enable formal signoff

Exhaustive formal analysis catches corner-case bugs and enables formal signoff for control and datapath blocks

Ease formal adoption

Easy-to-use formal apps, native integration with VCS and Verdi, and Formal Consulting Services reduce formal adoption effort

Boost productivity with AI power

Generative AI solution Formal Advisor turbocharging design verification

SYNOPSYS®

2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Usage Example - Property Generation

Pradip Prajapati, Synopsys
Anshul Jain, Synopsys
Zaqi Momin, Synopsys

Agenda

- Problem Statement
- Formal Advisor (FA) for Formal Verification Testbenches
- Introduction to RAG
- RAG-Enabled FA Framework
- SVA-RAG Database
- Quality Scoring Metric
- Results
- Conclusion & Future Scope

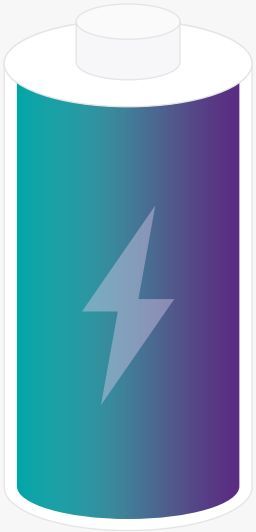
What Drains Formal Verification Resources

Testplanning

Spec Reading

Core Functions

Extract Rules

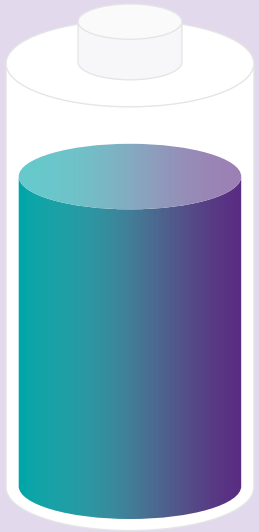


Testbench Development

Testbench Architecture

Instrumentation Logic

Property Coding

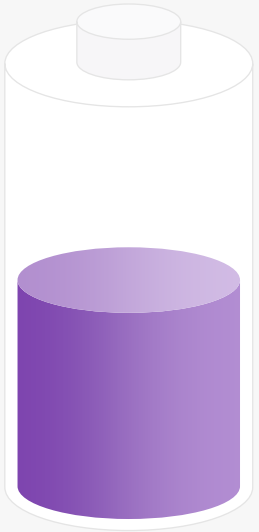


Debugging

Root-cause Analysis

Bug-fix Validation

Constraint Formulation

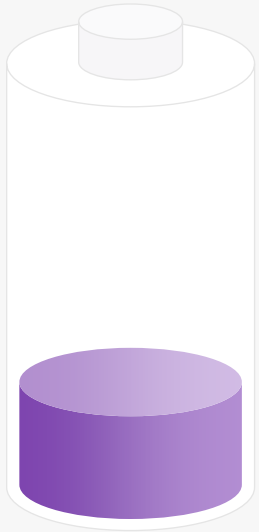


Proof Convergence

Abstractions

Invariant-mining

Proof Orchestration

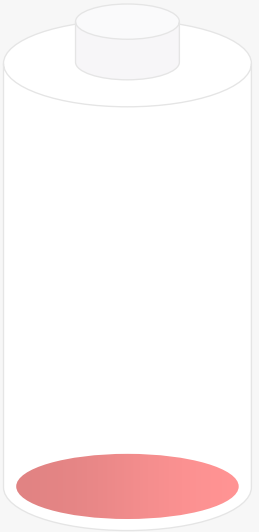


Sign-off

Coverage Closure

Cross-proofing

Report Creation



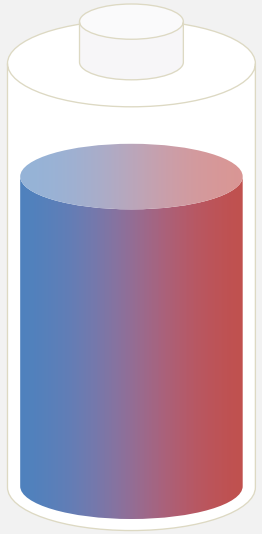
GenAI for Saving Energy (and Gaining Speed)

Testbench Development

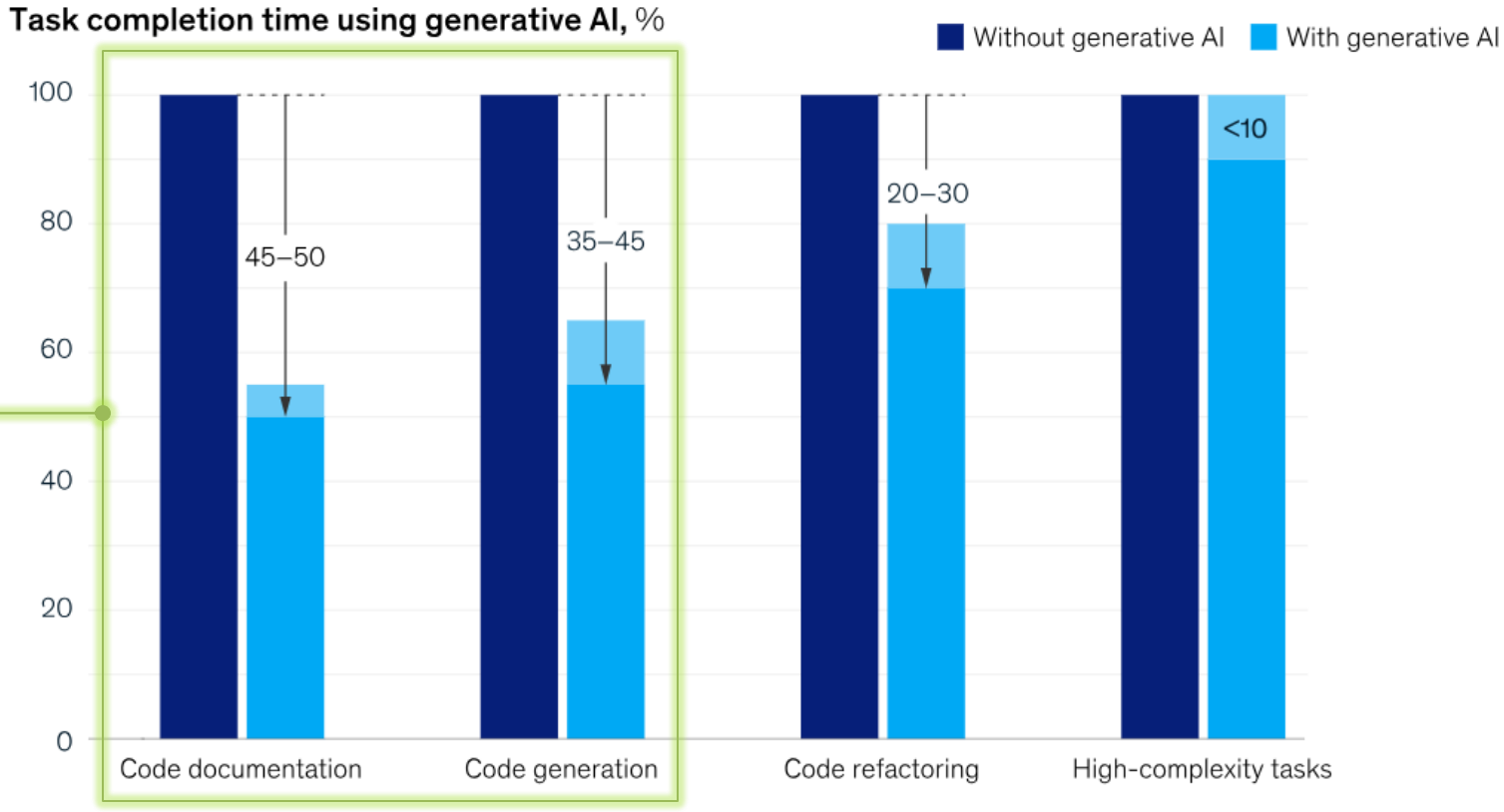
Testbench Architecture

Instrumentation Logic

Property Coding

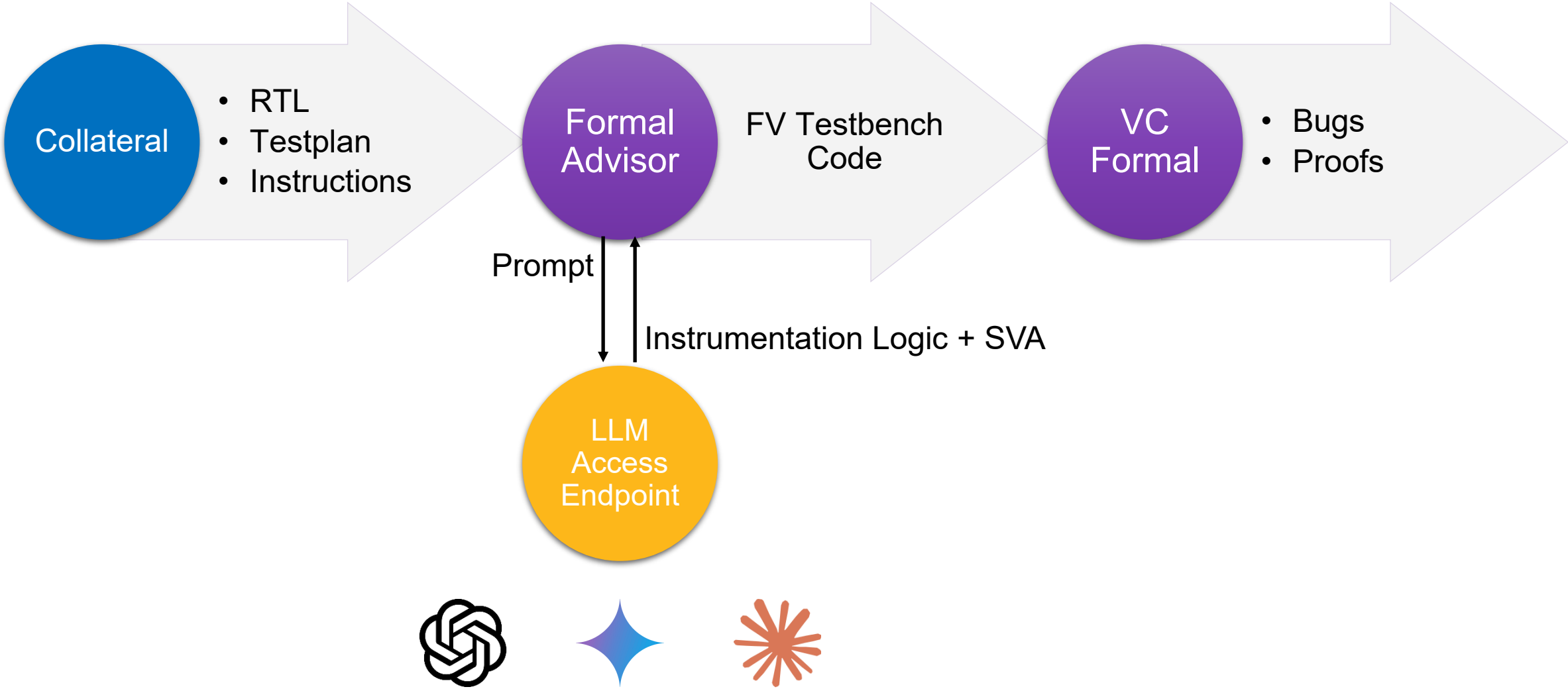


Generative AI can increase developer speed, but less so for complex tasks.



McKinsey & Company

Formal Advisor for Writing Testbench Code



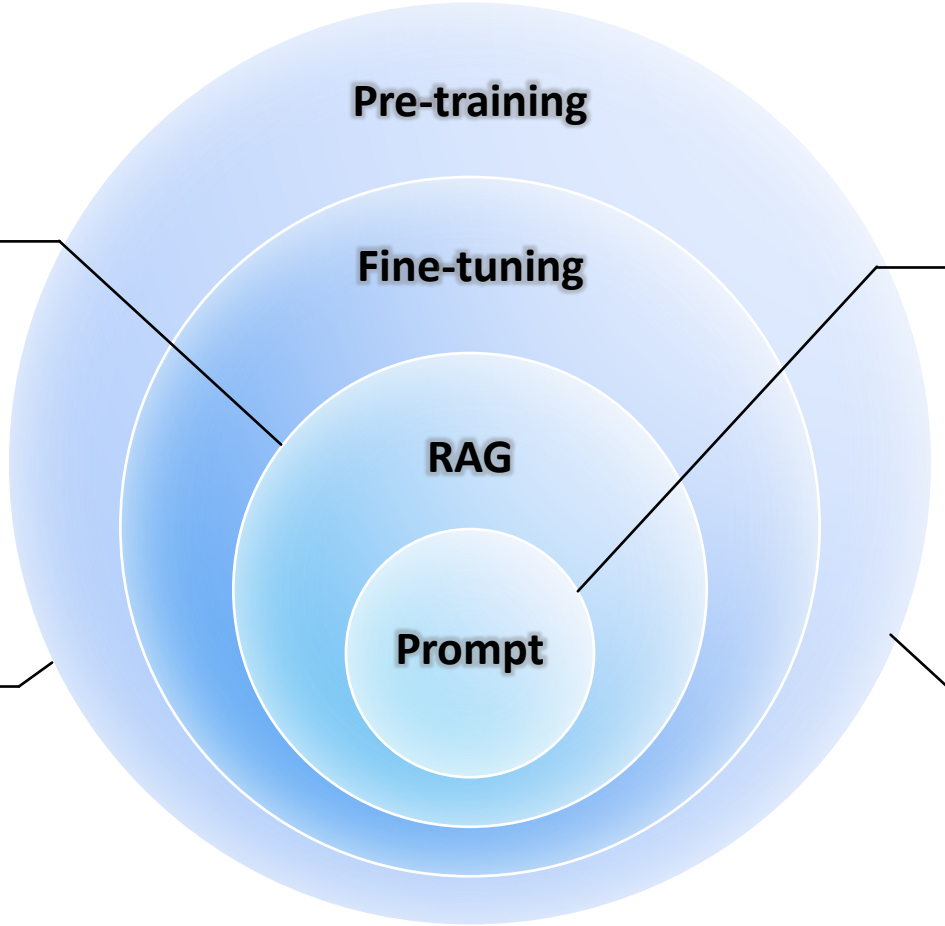
Formal Advisor is Encouraging, But… We Need Better Accuracy!

Retrieval-Augmented Generation (RAG)

Optimize responses by **referencing knowledge outside training data**; Extends model's capabilities to specific domain

Domain-Adaptive Pre-Training (DAPT)

Further train foundation models using a few billion tokens from chip design documents and code; Unsupervised training using unlabeled data



Prompt Engineering

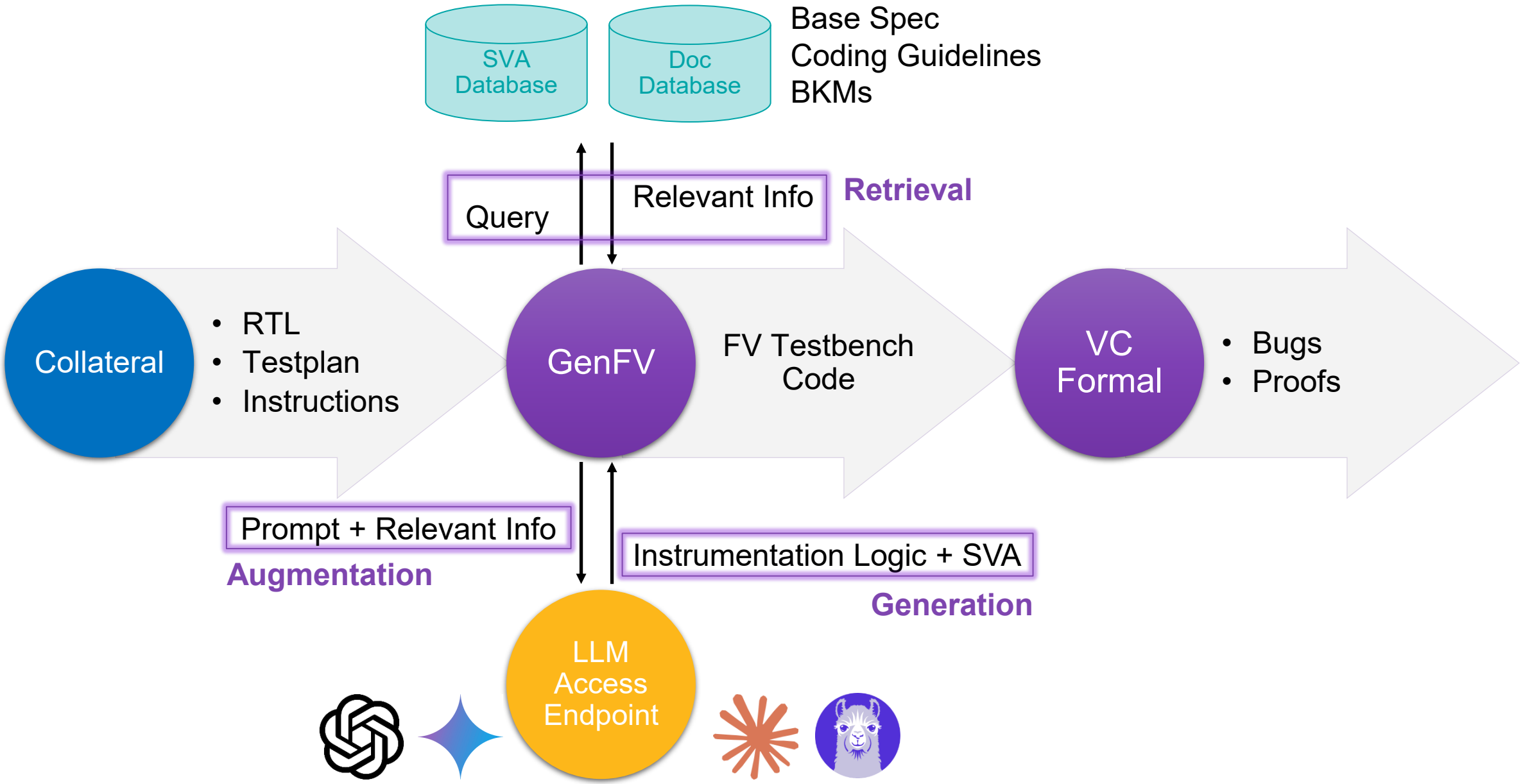
Apply **best-in-class prompt techniques** to generate high-quality responses from **foundation model.** (Explored in our previous work in DVCon India 2024)

Supervised Fine-Tuning (SFT)

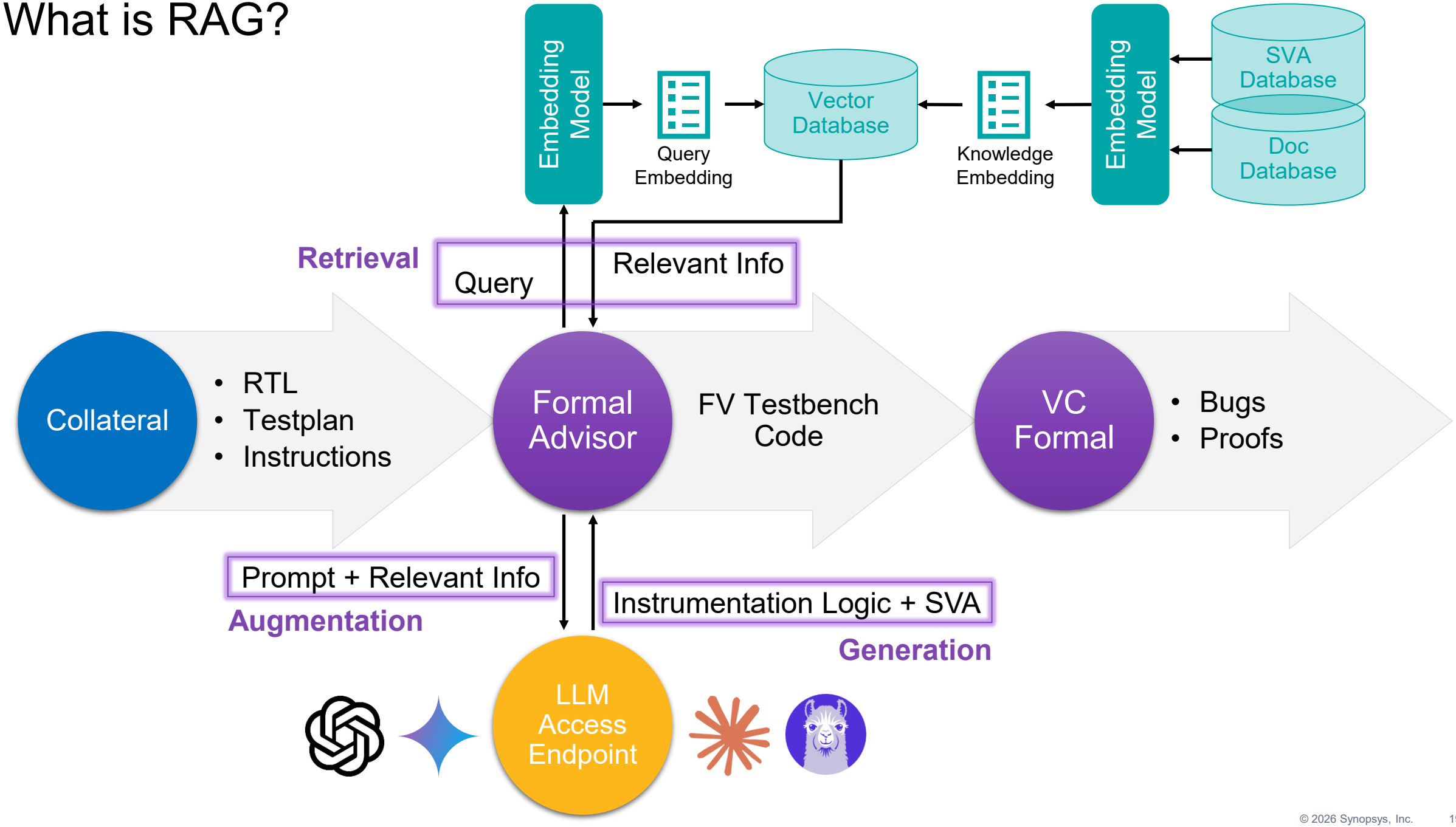
Model alignment to specific tasks using instructions using techniques such as PEFT (Parameter Efficient Fine-Tuning), LoRA (Low-rank Adaptation);
Uses labeled data in form of instructions and corresponding response.

*GenAI leap in formal verification testplanning, DVCon India 2024

RAG-powered Synopsys Formal Advisor



What is RAG?



SVA Database (JSONL Format)

Query	Code	Contributor	Time
Checks if two signals match after applying a bitwise mask	prop_mask_siged_eq: assert property (@(posedge clk_sig) disable iff (!resetn_sig) enable_check_sig |-> ((a_sig & mask_sig) == (b_sig & mask_sig)));	Zaqi	2024-10-15 15:16:04.319264

If there is at least one req then there should be at least one grant	logic pending_ack_sig;
	always@(posedge clk_sig) begin
	if(!resetn_sig)begin
	pending_ack_sig <= 0;
	end else if (ack_sig) begin
	pending_ack_sig <= 0;
	end else if (req_sig) begin pending_ack_sig <= 1;
	end
	end
	prop_req_grant: assert property (
	@(posedge clk_sig) disable iff (!resetn_sig)
	|req_sig |-> |gnt_sig);
		Pradip	2025-05-06 15:27:27.055401

…	…	…	…

Why RAG-powered Formal Advisor Works Better?

- LLMs are usually trained on open-source data
- Formal Verification is very closed-source for IP protection
- Good results with simple designs but struggles as it gets complex
- LLMs resolve this by adding specialized knowledge for these complex design
- RAG prevents the LLM from hallucinating by keeping the responses grounded in reality

Building Upon Our Previous Work

#	Prompt
1	*Assume the persona of a formal verification engineer, expert in verifying the functionality of digital circuits using formal property verification to ensure that all the design implementation adheres to design specification.*
2	*Please go through the specification provided as attachment and understand the overall functionality of the design. Then, execute the following steps to create a detailed formal verification testplan. Formal verification testplan comprise of list of checkers, constraints, and functional covers. Your description of each checker should be specific instruction on what needs to be implemented by a formal verification engineer. For example, instead saying "FIFO full should be asserted correctly", say "FIFO full should be high when all the entries of FIFO are in use, and FIFO full should be low when all the entries are not in use".*
3	*[Step 1] Summarize the specification in two sections, core functionality of the design and the flow of operation.*
4	*[Step 2] Write an exhaustive list of checkers required to verify each of the design functions. Checkers should be planned in such a way that they can be implemented using only inputs and outputs of the design.*
5	*[Step 3] Arrange the list of checkers, constraints, and functional covers in a tabular format. Table should have 4 columns. First column should be unique ID for each checker (e.g., CHK1, CHK2, ...), second column should be the name of checker, third column should be the detailed description of the checker.*
6	*Are these all the checkers? Please take care not to miss or repeat any checker.*
7	*[Step 4] Please suggest a staging plan in which the checkers should be implemented and debugged.*
8	*[Step 5] Arrange the staging plan in a tabular format. It should have 3 columns, Stage, Objectives and Tasks. Tasks should mention the implemented in each stage.*

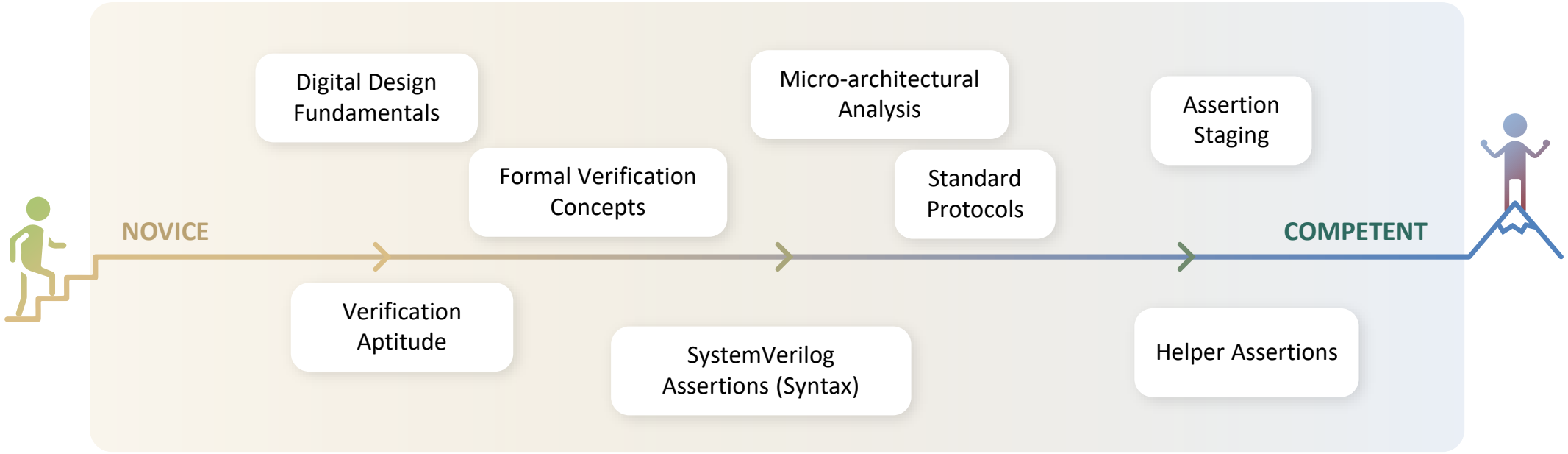
GenAI leap in formal verification testplanning, DVCon India 2024

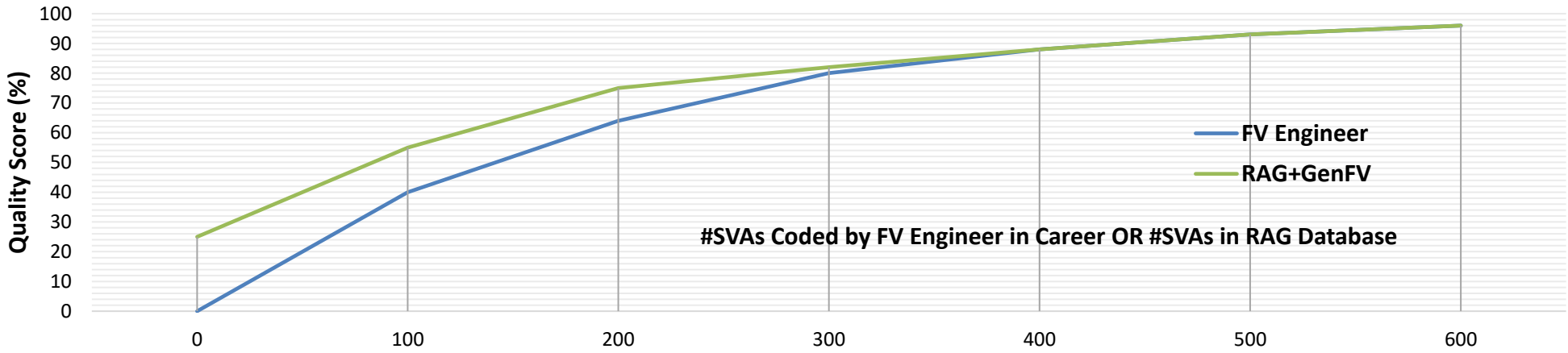
Custom Instruction for LLM

Several categories of instructions with a few examples with the highest impact on quality

Instruction Category	Example
Format Requirements	"The format of property should be in such a way that both the property definition and the directives like assert, cover and assume should be in the same line. In other words, no name to be generated for the properties."
Naming Conventions	"When generating signal or property names, do not append random or arbitrary numbers after the base name, unless a specific, meaningful index is required by the context."
Language Restrictions	"Do not use strong and weak keywords"
..	..

RAG-powered FA vs Human FV Engineer





Scoring Metric

Code Effectiveness (Ce), with a weight (We = 0.4)

Quantifies the degree to which the intended logic is verified by the property

Code Correctness (Cc), with a weight (Wc = 0.3)

Determines the presence of any false failures in the generated property

Code Quality (Cq), with a weight (Wq = 0.3)

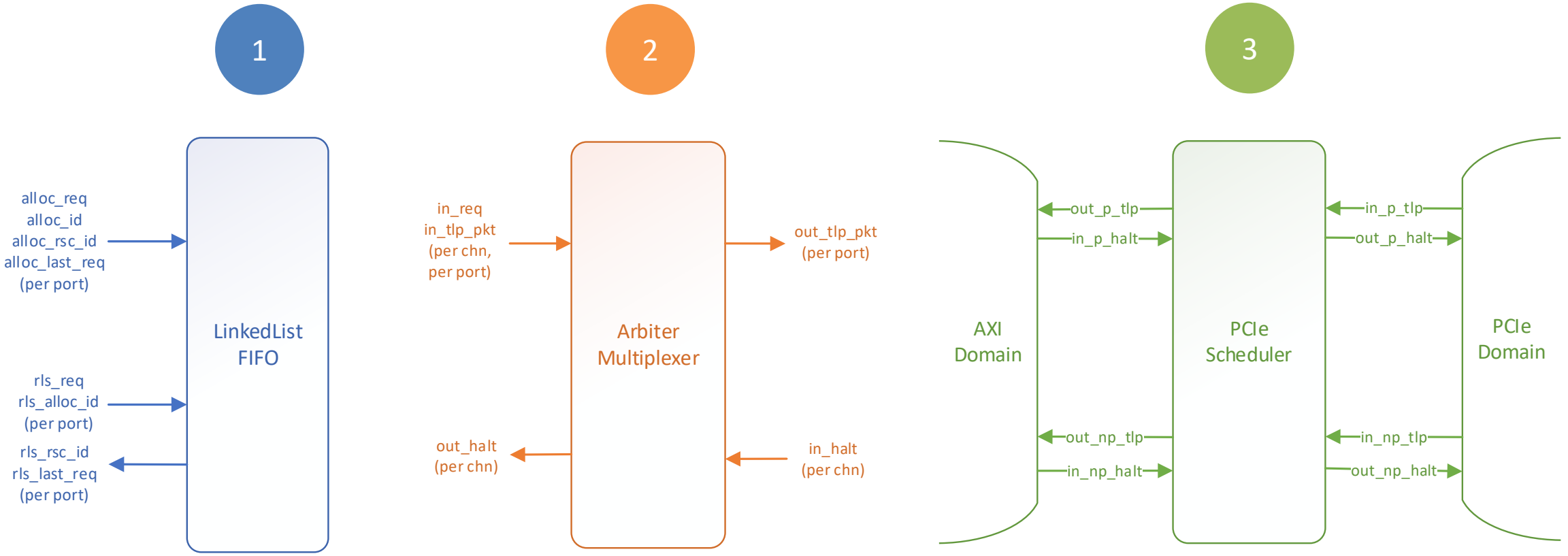
Measures adherence to coding guidelines

Overall quality (Q) can be expressed as a weighted sum

$$\boldsymbol{Q = Ce.We + Cc.Wc + Cq.Wq}$$

Assign a score of 0-100% for a complete set of assertion for a DUT for evaluation

Testing On Real Designs



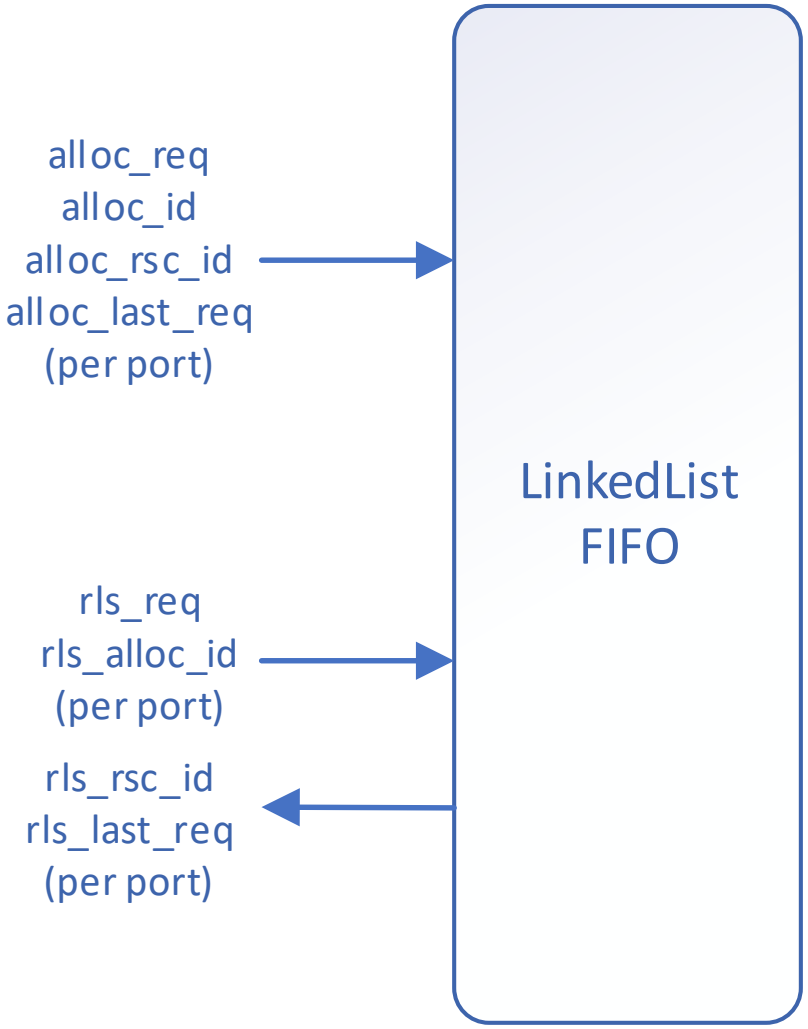
1

Allocator LUT

Allocator	Resource Head	Resource Tail
0	0	2
1	1	4
2	7	7
3	6	6

Resource LUT

Resource	Next Resource
0	3
1	4
2	2
3	5
4	4
5	2
6	6
7	7



1

Channel Inputs

Each Channel (CH0, CH1, ..., CHn) provides in_tlp_req and in_tlp_pkt

2

Request Masking & Arbitration Enable

Masks requests if TLPs are in progress; enables arbitration only if no halt and no TLP in progress.

3

Arbitration

Receives masked requests, outputs grants

4

Grant Latching

Latches grant for the duration of the TLP transfer.

5

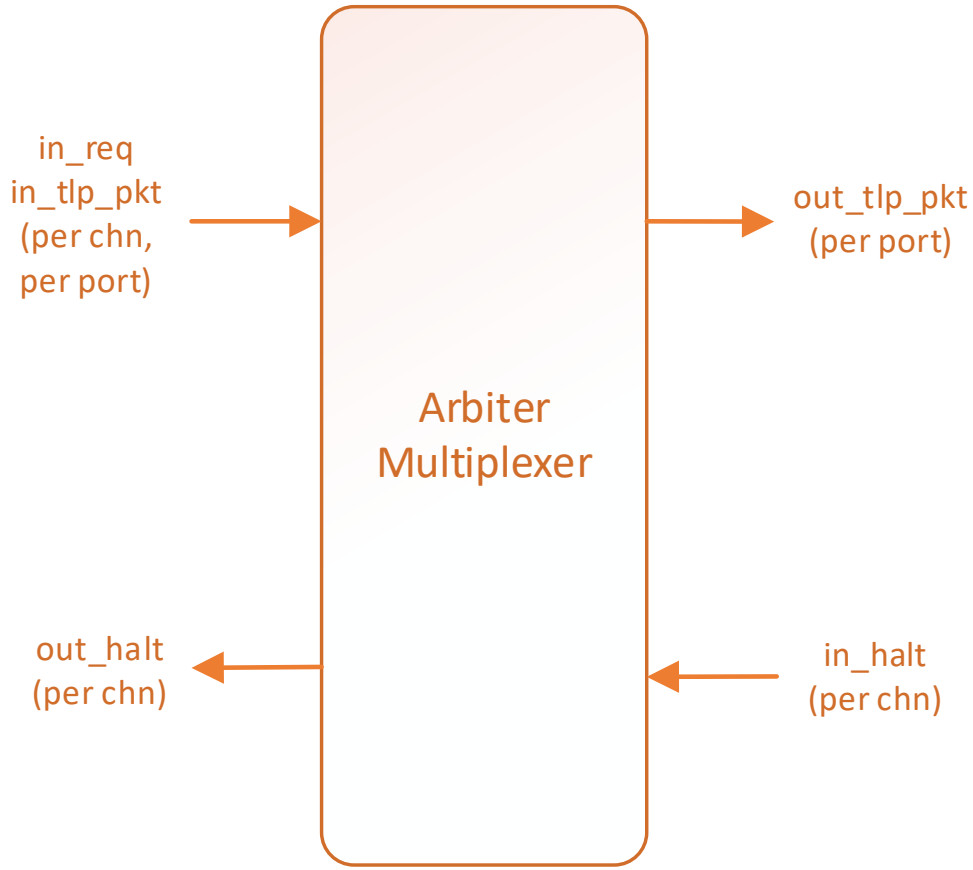
TLP Progress Tracking

Monitors tlps in progress; blocks new arbitration until EOT.

6

Multiplexing & Halt Logic

Latches grant for the duration of the TLP transfer.



2

3

Ordering Table (Table 2-40) - Must Pass / May Pass Relationships

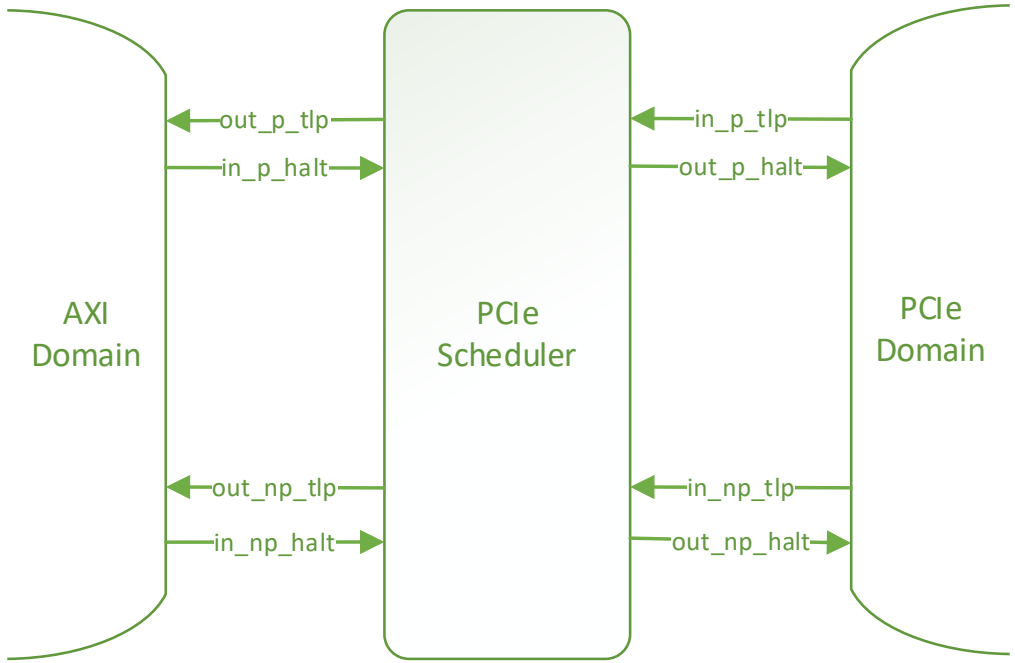
Row passes Column (Transaction B passes A)	Posted (A)	Non-Posted (A)	Completion (A)
Posted (B)	**Y/N** May Pass	**Y** **Must Pass**	**Y** **Must Pass**
Non-Posted (B)	**N** **Must NOT Pass**	**Y/N** May Pass	**Y/N** May Pass
Completion (B)	**Y/N** May Pass*	**Y** **Must Pass**	**Y/N** May Pass

Table Legend: Y = Must Pass | N = Must NOT Pass | Y/N = May Pass (implementation choice)

Key Ordering Rules:

1. Posted MUST pass NP and Cpl (prevents deadlock)
2. NP must NOT pass Posted (maintains write ordering)
3. Cpl MUST pass NP (prevents deadlock)
4. *Cpl passing P is optional (RO attribute dependent)
5. Y/N = Implementation choice (both valid)

Deadlock Prevention: Posted transactions MUST be able to pass blocked Non-Posted transactions to prevent circular dependencies in the fabric (Requester waiting for Completer waiting for Requester)



3

Table 2-42 Ordering Rules Summary

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted Request (Row A)		a) No b) Y/N	Yes	Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	a) No b) Y/N	Y/N	Y/N	Y/N
	NPR with Data (Row C)	a) No b) Y/N	Y/N	Y/N	Y/N
Completion (Row D)		a) No b) Y/N	Yes	Yes	a) Y/N b) No

Explanation of the row and column headers in § Table 2-42:

A **Posted Request** is a Memory Write Request or a Message Request.

A **Read Request** is a Configuration Read Request, an I/O Read Request, or a Memory Read Request.

An **NPR** (Non-Posted Request) **with Data** is a Configuration Write Request, an I/O Write Request, an AtomicOp Request, or a DMWr.

A **Non-Posted Request** is a Read Request or an NPR with Data.

Explanation of the entries in § Table 2-42:

A2a
A Posted Request must not pass another Posted Request unless A2b applies.

A2b
A Posted Request with RO [37] Set is permitted to pass another Posted Request. [38] A Posted Request with IDO Set is permitted to pass another Posted Request if the two Requester IDs (including Requester Segment when in FM) are different. Additionally, a Posted Request with IDO Set is permitted to pass another Posted Request with the same Requester ID if both Requests contain a PASID and the two PASID values are different.

A3, A4
A Posted Request must be able to pass Non-Posted Requests to avoid deadlocks.

A5a
A Posted Request is permitted to pass a Completion, but is not required to be able to pass Completions unless A5b applies.

A5b
Inside a PCI Express to PCI/PCI-X Bridge whose PCI/PCI-X bus segment is operating in conventional PCI mode, for transactions traveling in the PCI Express to PCI direction, a Posted Request must be able to pass Completions to avoid deadlock.

B2a
A Read Request must not pass a Posted Request unless B2b applies.

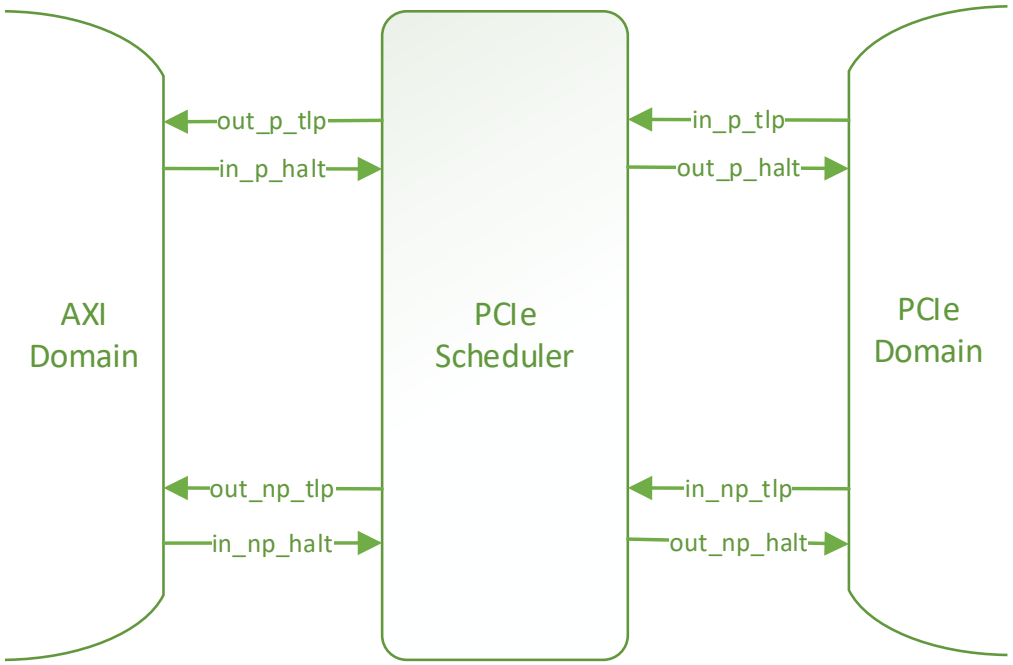
B2b
A Read Request with IDO Set is permitted to pass a Posted Request if the two Requester IDs (including Requester Segment in FM) are different. Additionally, a Read Request with IDO Set is permitted to pass a Posted Request with the same Requester ID if both Requests contain a PASID and the two PASID values are different.

C2a
An NPR with Data must not pass a Posted Request unless C2b applies.

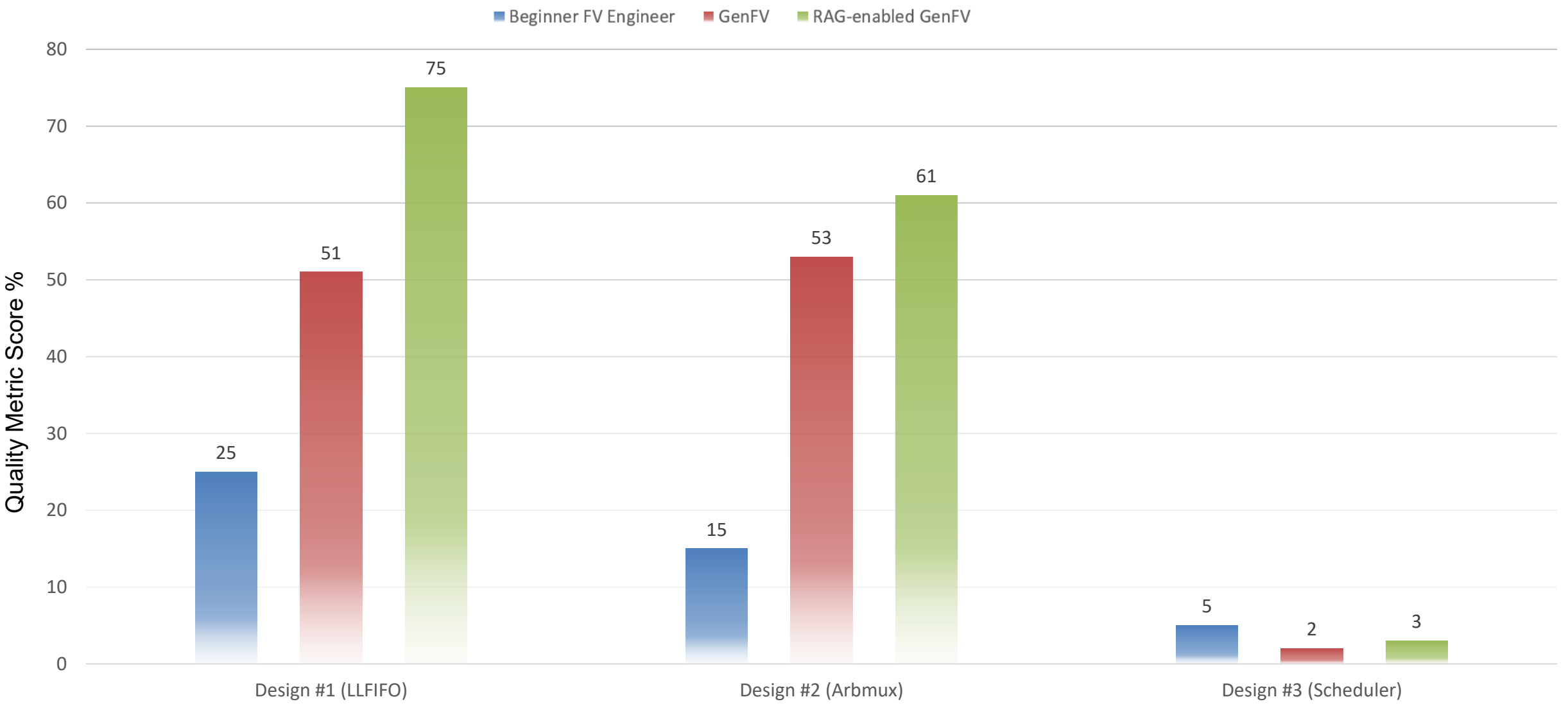
C2b
An NPR with Data and with RO Set [39] is permitted to pass Posted Requests. An NPR with Data and with IDO Set is permitted to pass a Posted Request if the two Requester IDs (including Requester Segment in FM) are different. Additionally, an NPR with Data and with IDO Set is permitted to pass a Posted Request with the same Requester ID if both Requests contain a PASID and the two PASID values are different.

B3, B4, C3, C4
A Non-Posted Request is permitted to pass another Non-Posted Request.

B5, C5
A Non-Posted Request is permitted to pass a Completion.



Results – Improvement from SVA-RAG



Conclusion & Future Scope

Conclusions

- RAG address challenges of LLMs in knowledge intensive tasks like FPV
- Demonstrated comprehensive framework to use RAG-powered GenFV
- Encouraging results with rich set of examples in RAG database
- For highly complex design, it struggles to understand verification intent

Future Scope

- Streamline process of enriching and maintaining RAG database
- Making RAG capable for more complex design
- Develop complete AI workflow from spec to testplan to testbench

2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

NVIDIA

Usage Example in Agentic Formal Flow

Ghaith Bany Hamad, Nvidia

AI Meets Formal — Scaling FV Expertise at the Speed of AI

Ghaith Bany Hamad

Agenda

- AI for FV Landscape
- Benchmarking AI Capabilities
- GenAI for FV Code
  - FSM Agent
  - VIP Agent
  - Custom Checker Agents
  - GenFV Agents
- GenAI for Debugging
- Conclusion

AI for FV Landscape

AI for FV Landscape – Planning to Sign-off

Planning

- Design / Arch Spec
- uArch & RTL Analysis
- ROI: FV vs DV vs Perf
- Simulation Data
- Legacy – RTL / FV
- Old Bug Analysis
- Test-planning

Implementation

- Testbench(es)
- Checkers
- Constraints
- Covers
- Resources
- Infrastructure

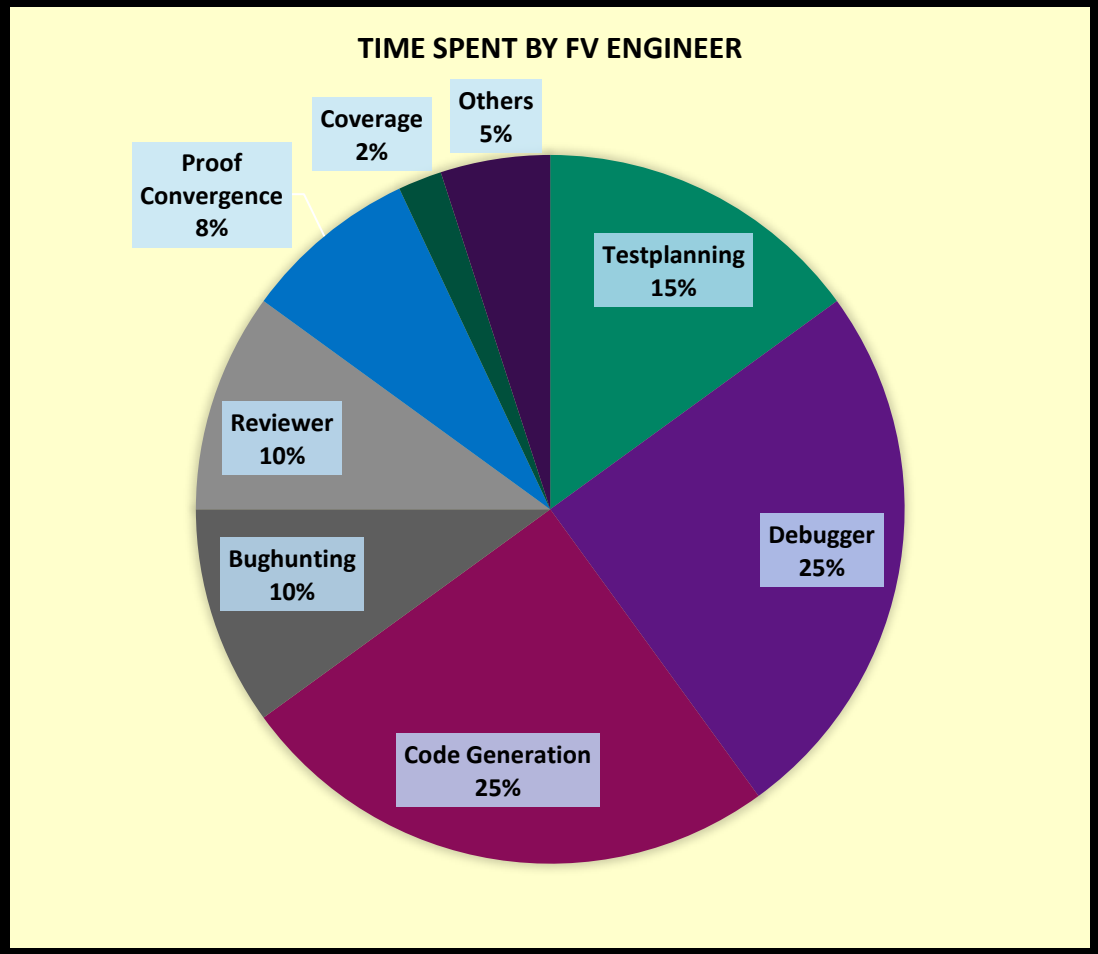
Closure

- FV Code
- Failure debugging
- Bug Reports
- Tool Settings
- Bug Hunting
- Convergence
- Abstractions

Sign-off

- Coverage Analysis
- Constraints Validation
- RPD Validation
- FV Code Review
- Clean Regressions
- Documentation

AI for FV Landscape – Agents ROI



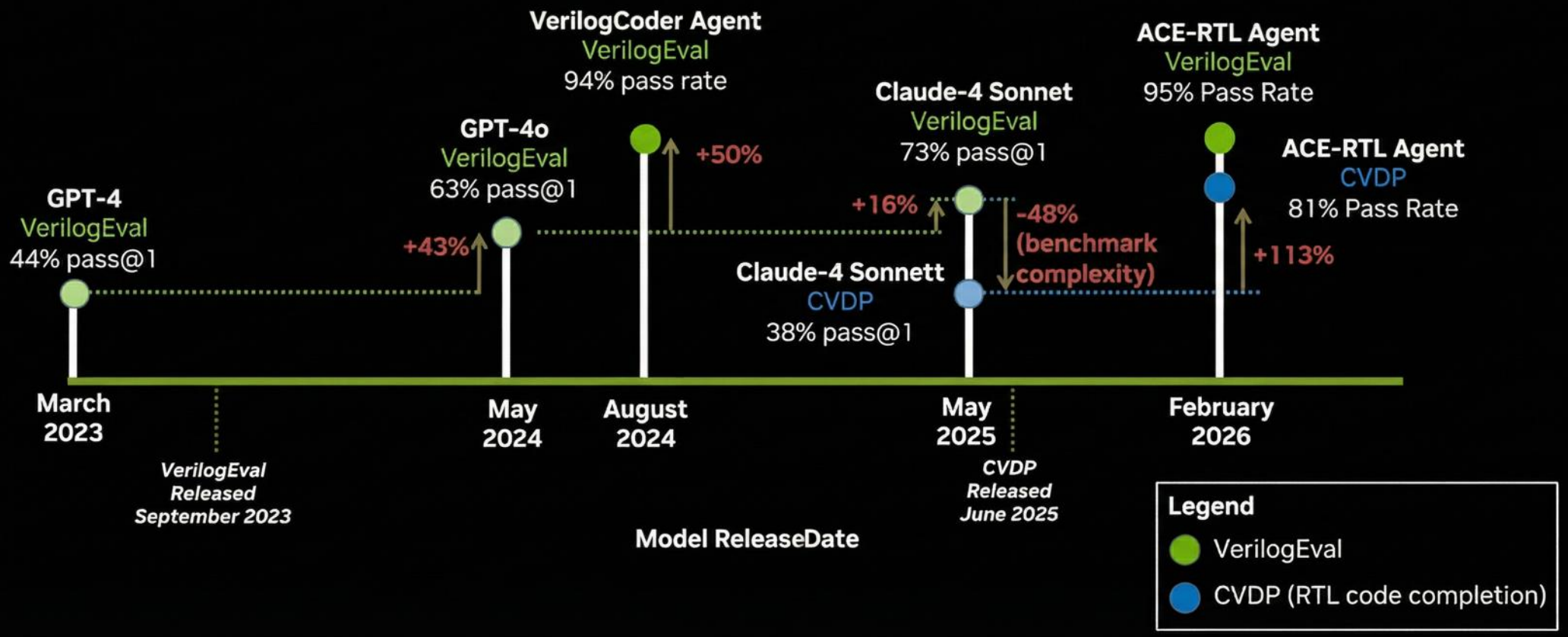
Tasks	FV Expertise	Profit (ROI w/ Agents)	Investment (Cost Engi.)
Testplanning	Med	Med	Med
Debugger	Low – Med	Very High	High
Code Generation	Med – High	High	Med
Bughunting	High	Very High	High
Reviewer	Low	High	Med
Proof Convergence	High	Very High	High
Coverage	Low	High	Med
Others	Low	Med	Med

Benchmarking AI Capabilities

Models and Agent Performance on RTL Code Generation

Agents are as important as the models

VerilogEval - Released September 2023 – 156 problems across 1 category
Comprehensive Verilog Design Problems (CVDP) - Release June 2025 – 783 problems across 13 categories



VerilogEval: https://doi.org/10.1109/ICCAD57390.2023.10323812
CVDP: https://arxiv.org/abs/2506.14074

VerilogCoder: https://doi.org/10.1609/aaai.v39i1.32007
ACE-RTL: https://www.arxiv.org/abs/2602.10218

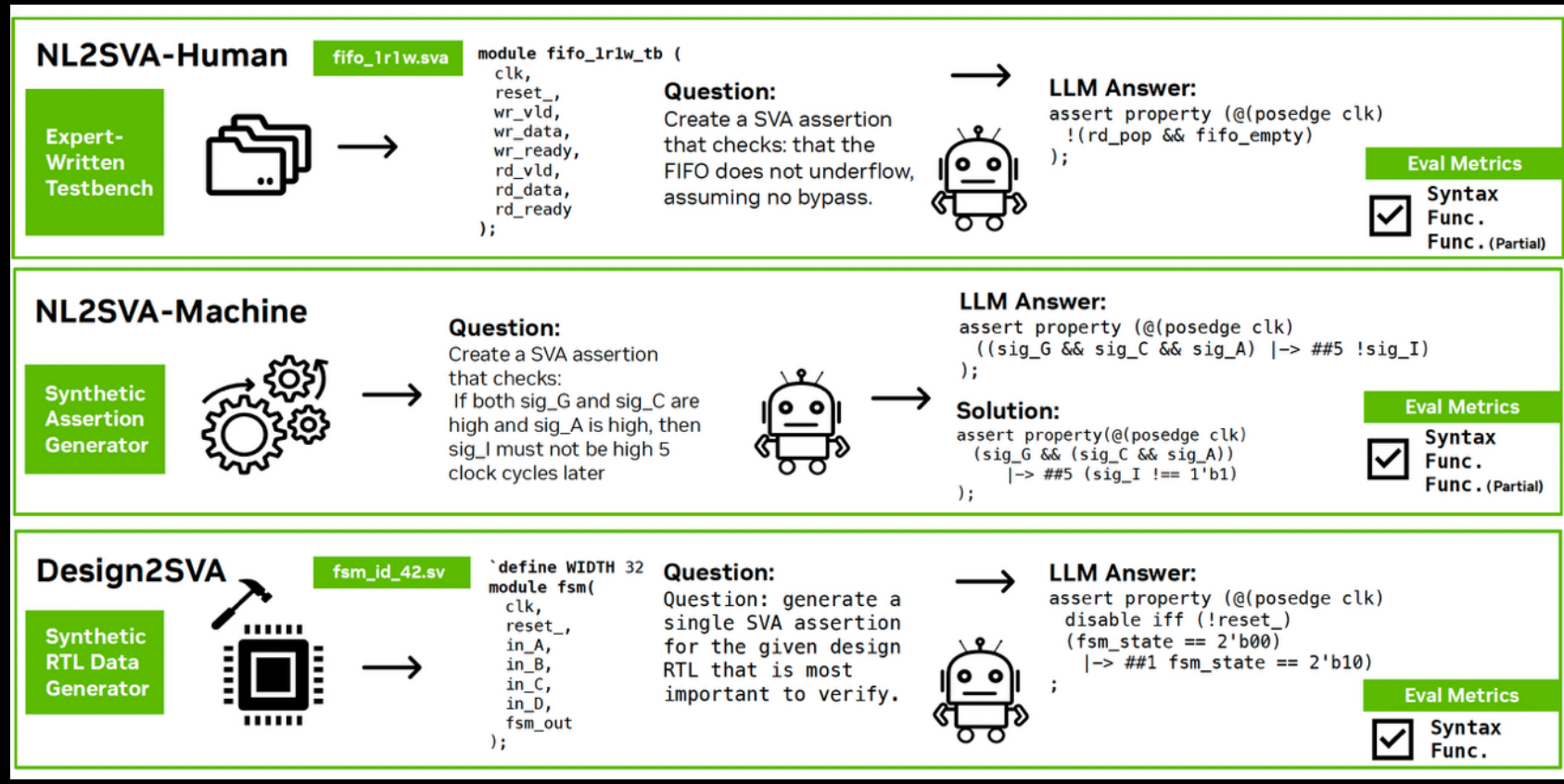
Evaluating LLM's Current Capabilities for FV code

Datasets:

- Three benchmarking tasks:
  - NL2SVA-Human
  - NL2SVA-Machine
  - Design2SVA
- https://github.com/NVlabs/FVEval

Evaluation Flow

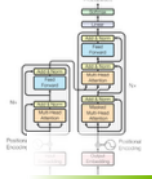
- Integrates FV tools for end-to-end automatic evaluation
- Holistic evaluation of LLM's generated assertion using property equivalence checking



FVEval: Understanding Language Model Capabilities in Formal Verification of Digital Hardware
Minwoo Kang, Mingjie Liu, Ghaith Bany Hamad, Syed Suhaib, Haoxing Ren
Design, Automation & Test in Europe Conference (DATE) 2025

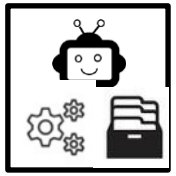
FV AI Capabilities

Build LLM Model based on Transformer model
Attention is all you need

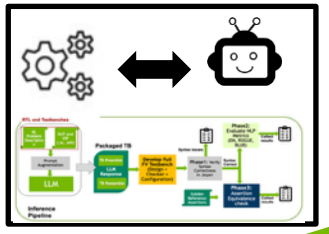


Initial Experiments with GPT 3 and GPT 3.5

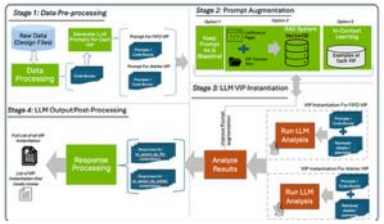
Evaluation of LLM Capabilities and Benchmarking
NvBugs, Generate basic assertion



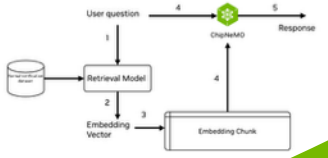
Built an Evaluation Flow to benchmark AI generated code "FVEval benchmark"



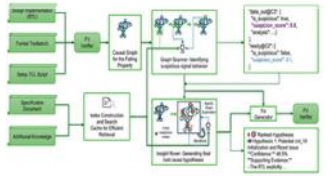
FV code generation:
VIP instantiation flow



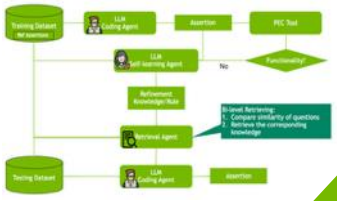
FV RAG
Started working for RAGs for VIP Designs, FV Infra



Debugger Agent

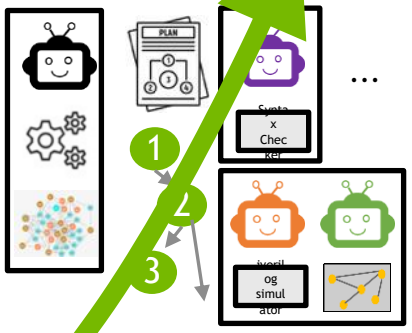


Assertion Generation from Natural Language Specifications via Multi-Agent Language Models



Utilize LLM for Testplan Creation

Multi-Agent for FV Testbench code generation
Multi-Agent & Multi-LLMs + Tools Reasoning + FV code Generation



FV ASSISTANT

2021 INFORMATION ERA

2023 GENERATIVE AI Era

2024

2025 REASONING Era

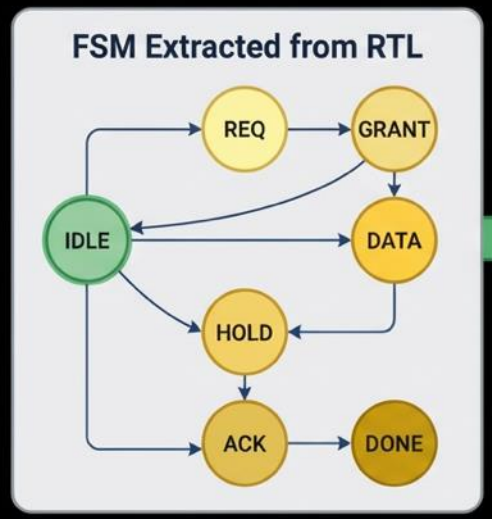
Jan 2026 AGENTIC AI Era

GenAI for FV Code

GenAI for FV Code - FSM

Forward Progress

How AI Can I
FSM Forwarc



Liveness Assertion

```
FSM_state != IDLE |->
  eventually(FSM_state == IDLE)
```

State Transition Covers

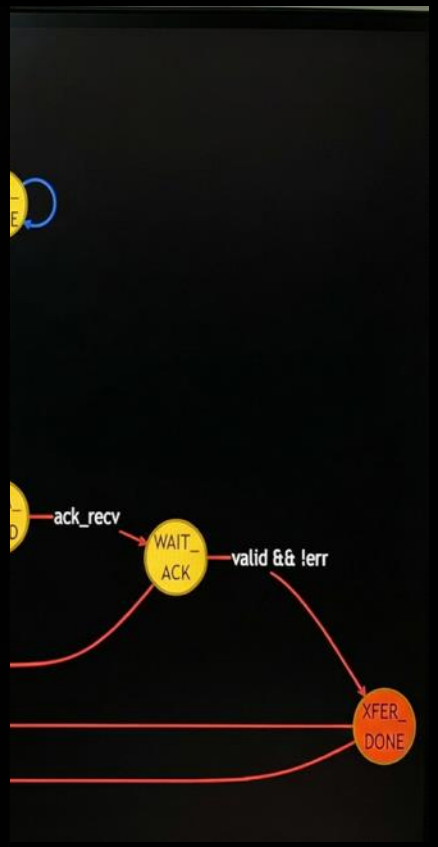
```
cover(FSM_state == REQ → GRANT)
```

Decomposed Properties

```
FSM_state == x |->
eventually(FSM_state == IDLE)
```

Proven → Assumptions

Convert proven assertions into assumptions for compositional reasoning

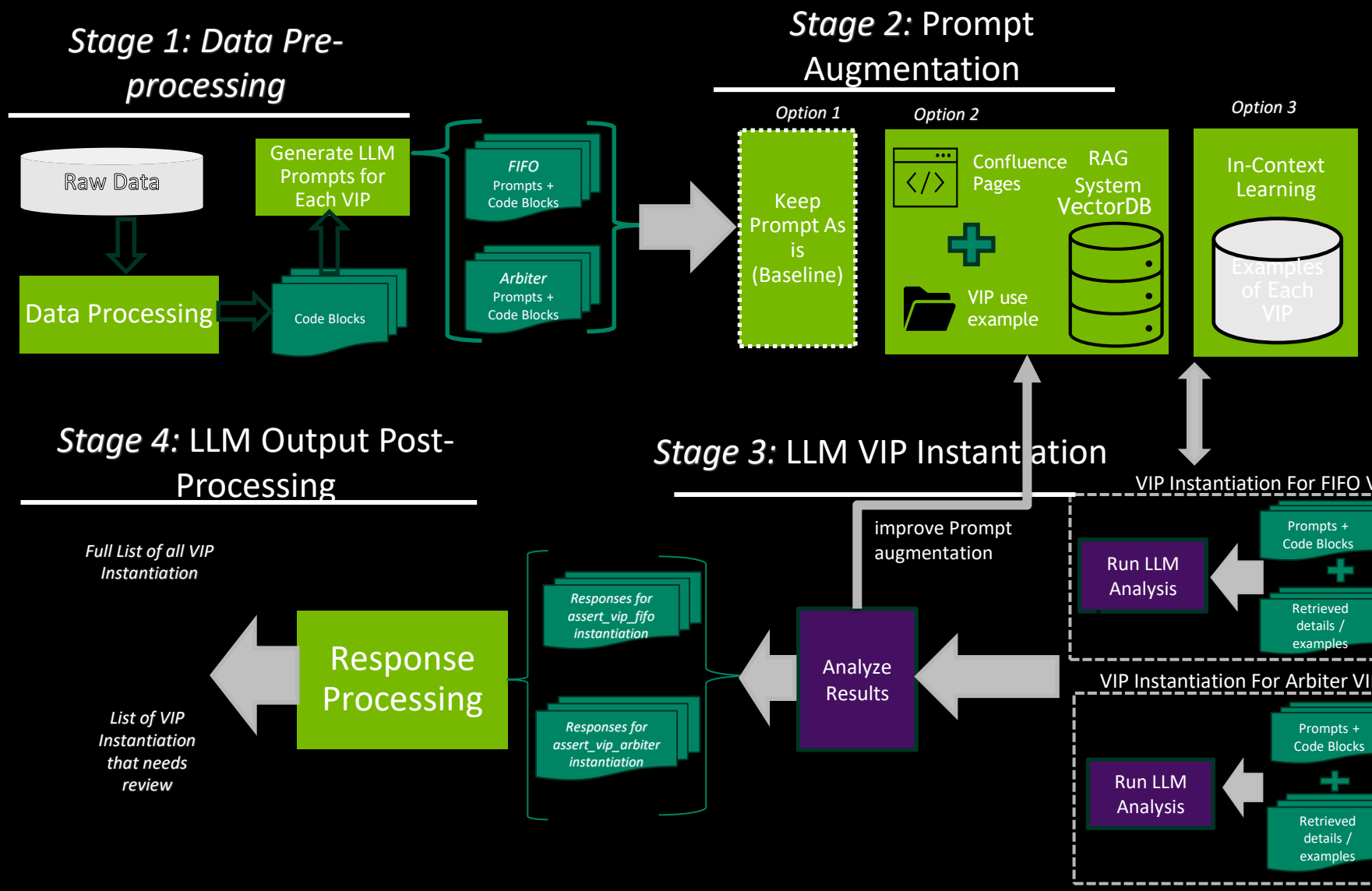


GenAI for FV Code - VIP

Towards Automated Formal Verification IP Instantiation via LLMs

Yunsheng Bai, Ghaith Bany Hamad, Syed Suhaib, Haoxing Ren

https://dvcon-proceedings.org/wp-content/uploads/1016-2.pdf



GenAI for FV Code - **VIP**

Results

FV VIP	LLM Model	Prompt Augmentation	Correct	Incorrect	Missing	Total # of instances
nv_assert_vip_fifo	chipmixtral_8x7b_chat_tp4_trt_h100	Baseline	0	0	7	7
	chipmixtral_8x7b_chat_tp4_trt_h100	ICL	4	1	2	7
	Llama-3-70b	RAG	7	0	0	7
nv_assert_vip_arb	chipmixtral_8x7b_chat_tp4_trt_h100	Baseline	0	0	12	12
	chipmixtral_8x7b_chat_tp4_trt_h100	ICL	1	4	7	12
	Llama-3-70b	RAG	11	1	0	12

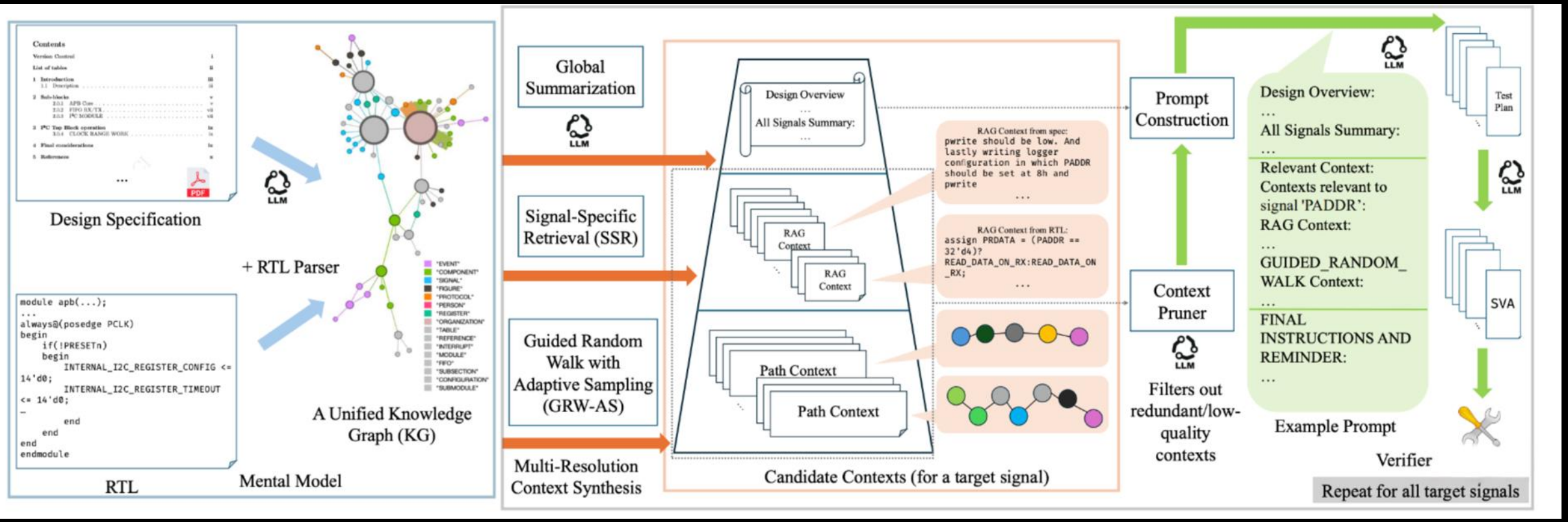
Correct: This column means completely correct instantiation, all signals correct

Incorrect: This column means VIP instantiation for modules present with either incomplete or wrong signals

Missing: This column means missing VIP instantiation for the module

- RAG method provides the most accurate VIP Instantiation compared to ICL and baseline comparisons for both VIPs
- RAG method utilized Llama-3-70b as the driving LLM, providing additional value by demonstrating that expensive fine-tuning may not be always needed

GenAI for FV Code – AssertionForge Framework



AssertionForge: Enhancing Formal Verification Assertion Generation with Structured Representation of Specifications and RTL
Yunsheng Bai, Ghaith Bany Hamad, Syed Suhaib, Haoxing Ren

https://arxiv.org/pdf/2503.19174

GenAI for FV Code - AssertionForge

Results

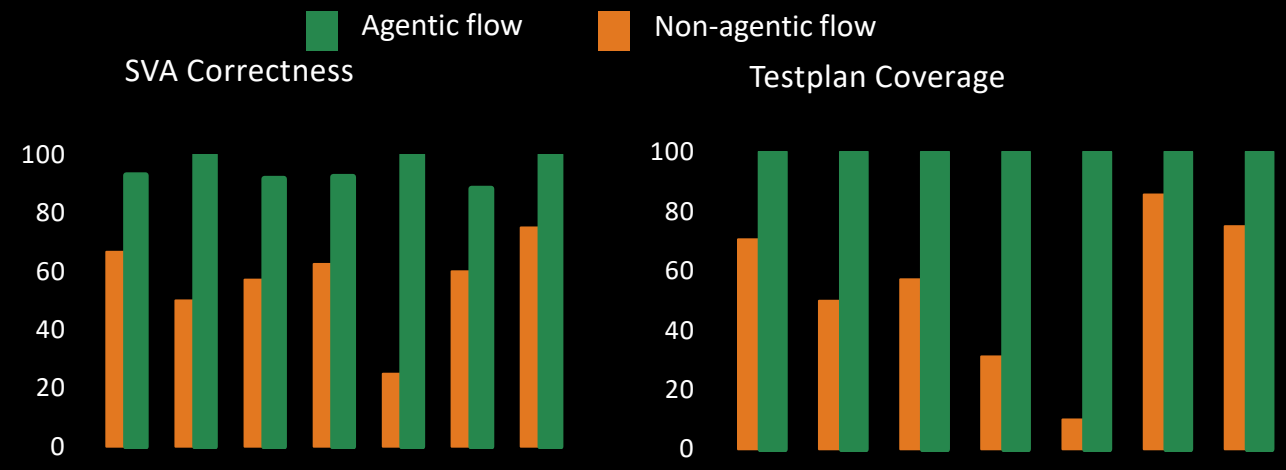
EVALUATION RESULTS

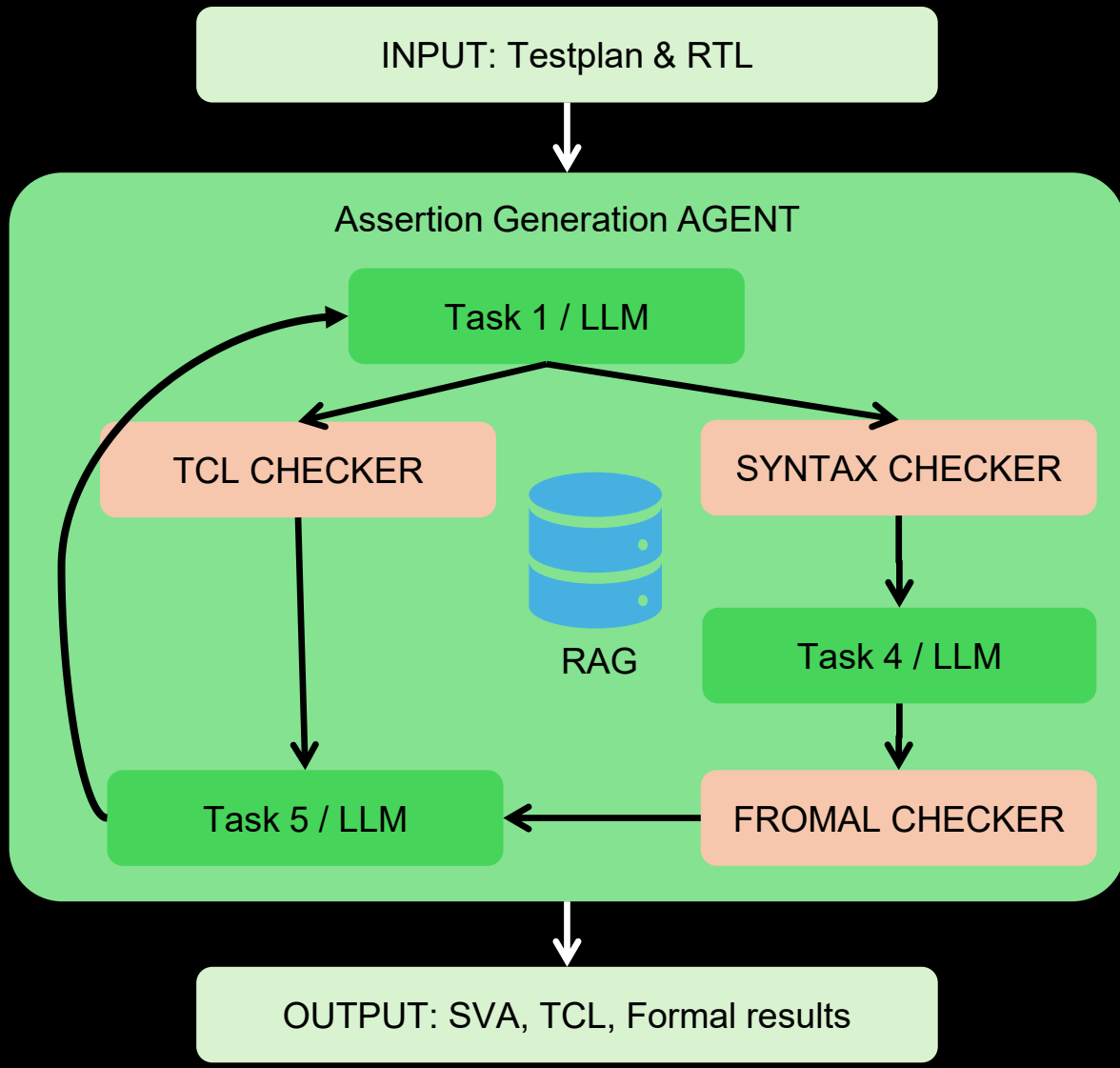
#SVA: total assertions generated | #SynC: syntactically correct | #Proven: proven or passing
COI Statement / Branch / Functional / Toggle: coverage metrics for each coverage model under COI

Model	#SVA	#SynC	#Proven	COI Coverage (%)			
				Statement	Branch	Functional	Toggle
APB							
AssertLLM	221	170	56	100.00	86.67	84.79	84.16
AssertionForge w/o RTL	188	128	22	100.00	86.67	70.05	68.32
AssertionForge	**615**	**549**	**220**	**100.00**	**100.00**	**100.00**	**100.00**
ETHMAC							
AssertLLM	520	106	15	45.10	44.83	49.01	50.29
AssertionForge w/o RTL	1345	331	42	50.98	51.72	71.21	77.46
AssertionForge	**1673**	**960**	**208**	**100.00**	**100.00**	**99.12**	**98.84**
OpenMSP430							
AssertLLM	409	128	55	94.70	95.05	87.32	85.81
AssertionForge w/o RTL	1123	216	106	99.08	98.91	88.94	86.95
AssertionForge	**1600**	**698**	**327**	**100.00**	**99.76**	**90.09**	**88.16**
SockIT							
AssertLLM	118	49	13	85.71	82.35	88.54	90.46
AssertionForge w/o RTL	393	131	41	100.00	**100.00**	**99.33**	99.08
AssertionForge	**448**	**136**	**49**	**100.00**	**100.00**	**99.78**	**99.69**
UART							
AssertLLM	249	83	29	74.36	78.26	66.78	63.01
AssertionForge w/o RTL	233	93	28	74.36	78.26	70.72	68.49
AssertionForge	**253**	**132**	**27**	**84.62**	**84.78**	**88.82**	**90.41**

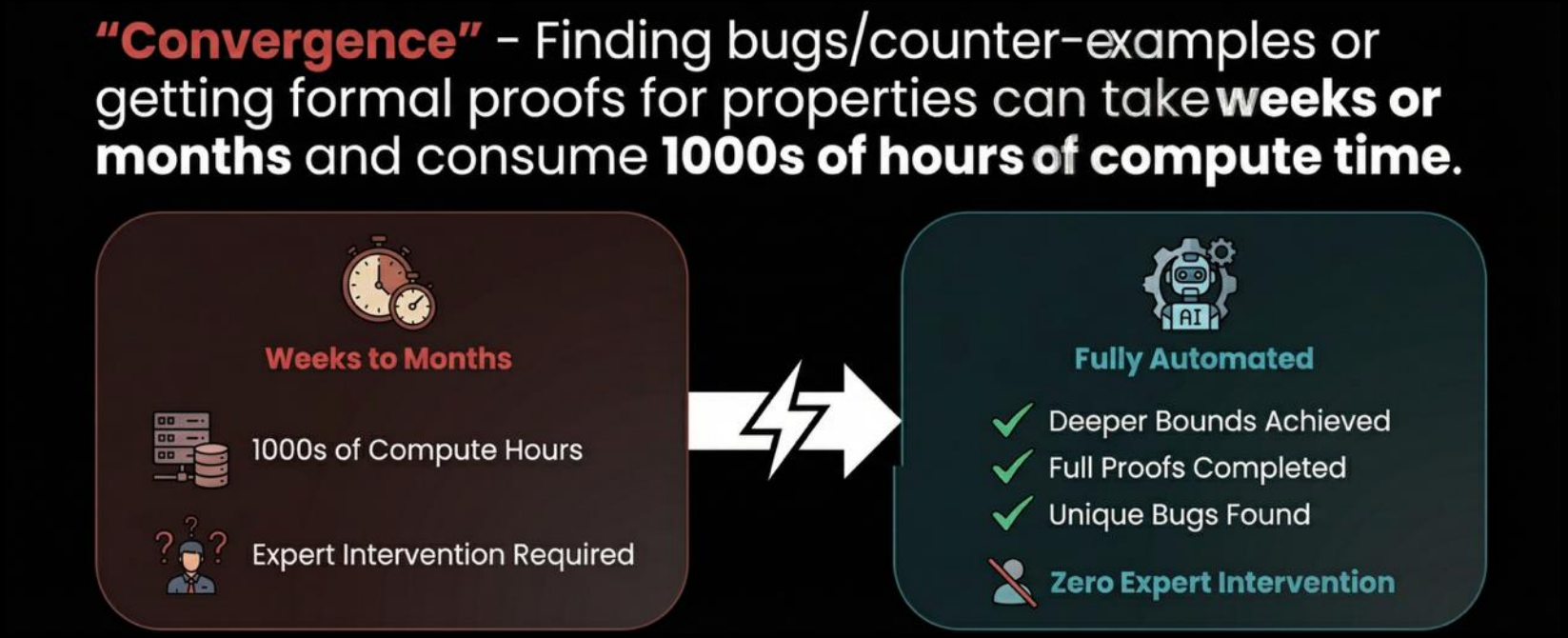
GenAI for FV Code - **GenFV - Agentic Assertion Generation**

- Agentic Flow with multiple LLMs interacting together
- Collaborating with Core VC Formal components
- Leveraging RAG for guidance
- Multipass to give the best results
- Plug & play with other agents





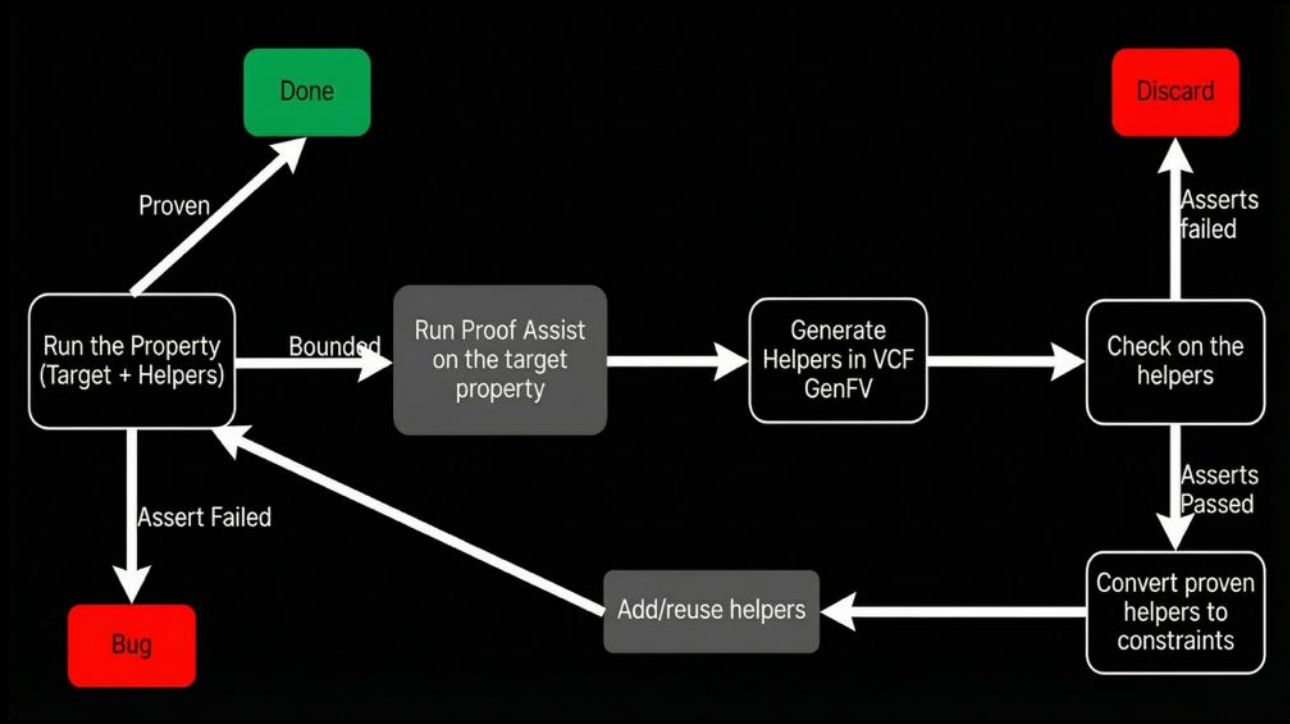
GenAI for FV Code - **GenFV – Helper Assertion Agent**



- Worked with VCF team and rolled out a solution in VC Formal GenFV using AI agents, LLMs, and formal engines to automate the task of finding "helper assertions" to converge on hard properties

GenAI for FV Code

GenFV – Helper Assertion Agent



Iterative method:

- genfv_generate_helpers -iterative -property {Target_Property_Name} -count 10 -helper_time 15m -max_iter 10 -target 25

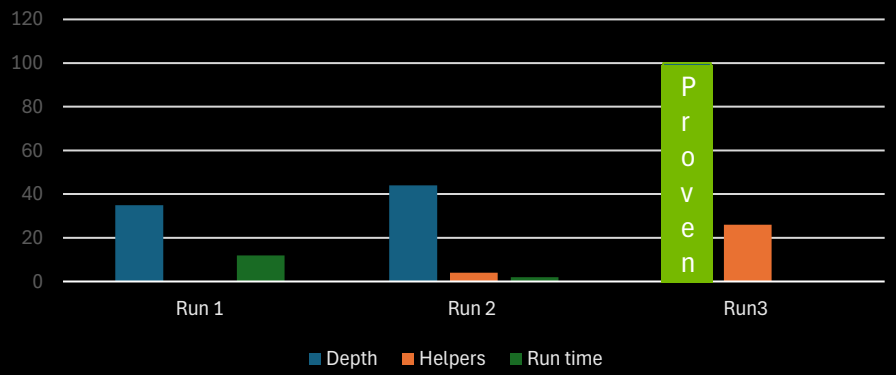
To run the helpers in assume guarantee fashion :

- ***genfv_check_helpers*** *-property {Target_Property_Name } -helper_time 2m -prop_time 5*

GenAI for FV Code - GenFV – Helper Assertion Agent

Property 01:

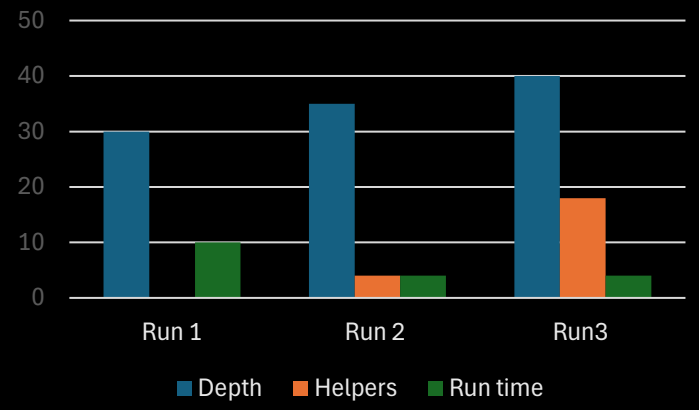
- Inconclusive after 12 Hrs
- Proof Assist Depth after 30m: 30+
- The target assertion verifies correct linked-list construction and validates the sequencing across multiple link stages (three to four stages).



	Run1	Run2	Run3
Depth(Cycles)	35	44	Proven
Helpers(Count)	0	4* (10)	26* (50)
Run Time(Hrs)	12	2	~2mins

Property 02:

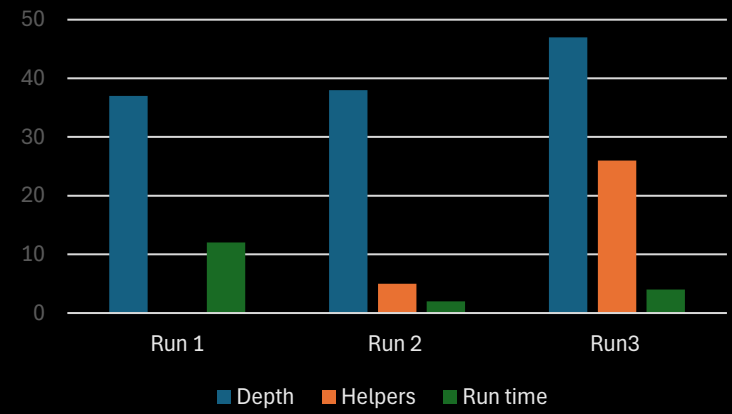
- Inconclusive after 10 Hrs
- Depth after 10Hr Run: 30+ Proof Assist Depth after 30m: 30



	Run1	Run2	Run3
Depth(Cycles)	30	35	40
Helpers(Count)	0	4*(10)	18*(30)
Run Time(Hrs)	10	4	4

Property 03:

- Inconclusive after 12 Hrs
- Depth after 4Hr Run: 37



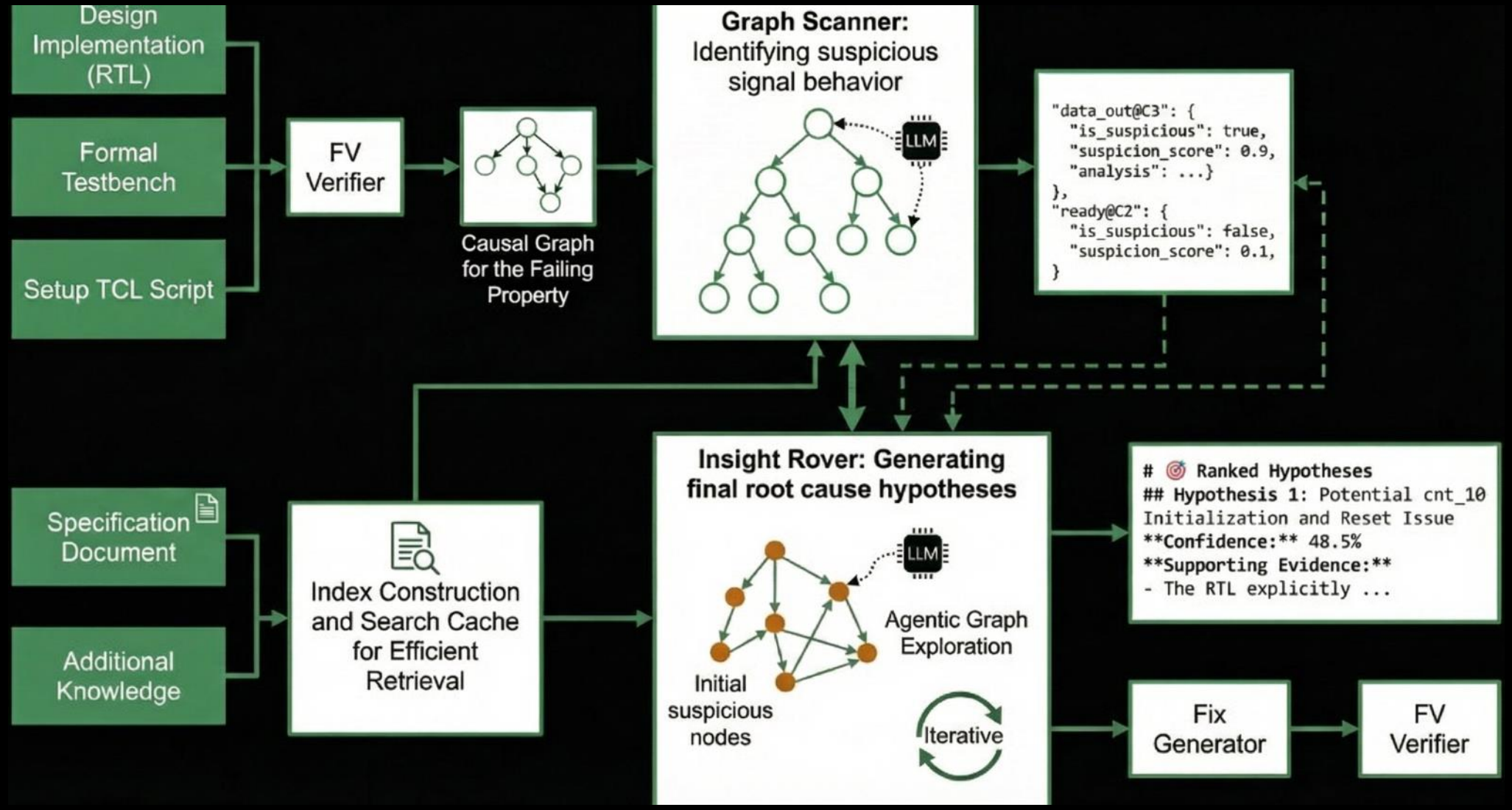
	Run1	Run2	Run3
Depth(Cycles)	37	38	47
Helpers(Count)	0	5*(10)	26*(50)
Run Time(Hrs)	12	4	4

*** Syntactically Correct/Formally Correct**

GenAI for FV Debugging

GenAI for FV Debugging - FVDebug

FVDebug: An LLM-Driven Debugging Assistant for Automated Root Cause Analysis of Formal Verification Failures **Accepted in DVCON 26**



Conclusion

- Automated VIP Instantiation: LLMs streamline and automate the integration of Verification IPs (VIPs), reducing manual effort.
- Retrieval-Augmented Generation (RAG): Achieves superior accuracy in VIP instantiation compared to baseline and In-Context Learning (ICL).
- Efficiency Gains: Significant reduction in verification time by automating assertions generation and VIP deployments.
- Scalability: Modular framework adaptable for various VIPs without additional model fine-tuning.
- Improved Verification Coverage: More reliable VIP instantiations reduce missed bugs and enhance verification integrity.
- Helper assertion flow has great potential, but still needs to improve scalability and try to get full proof
- AI is starting to show value for other FV tasks such as testplanning and failure debugging.

2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Google

AI Deployment Best Practices

Bindumadhava S S, Google

Google

GenFV: GenAI Driven Formal Verification

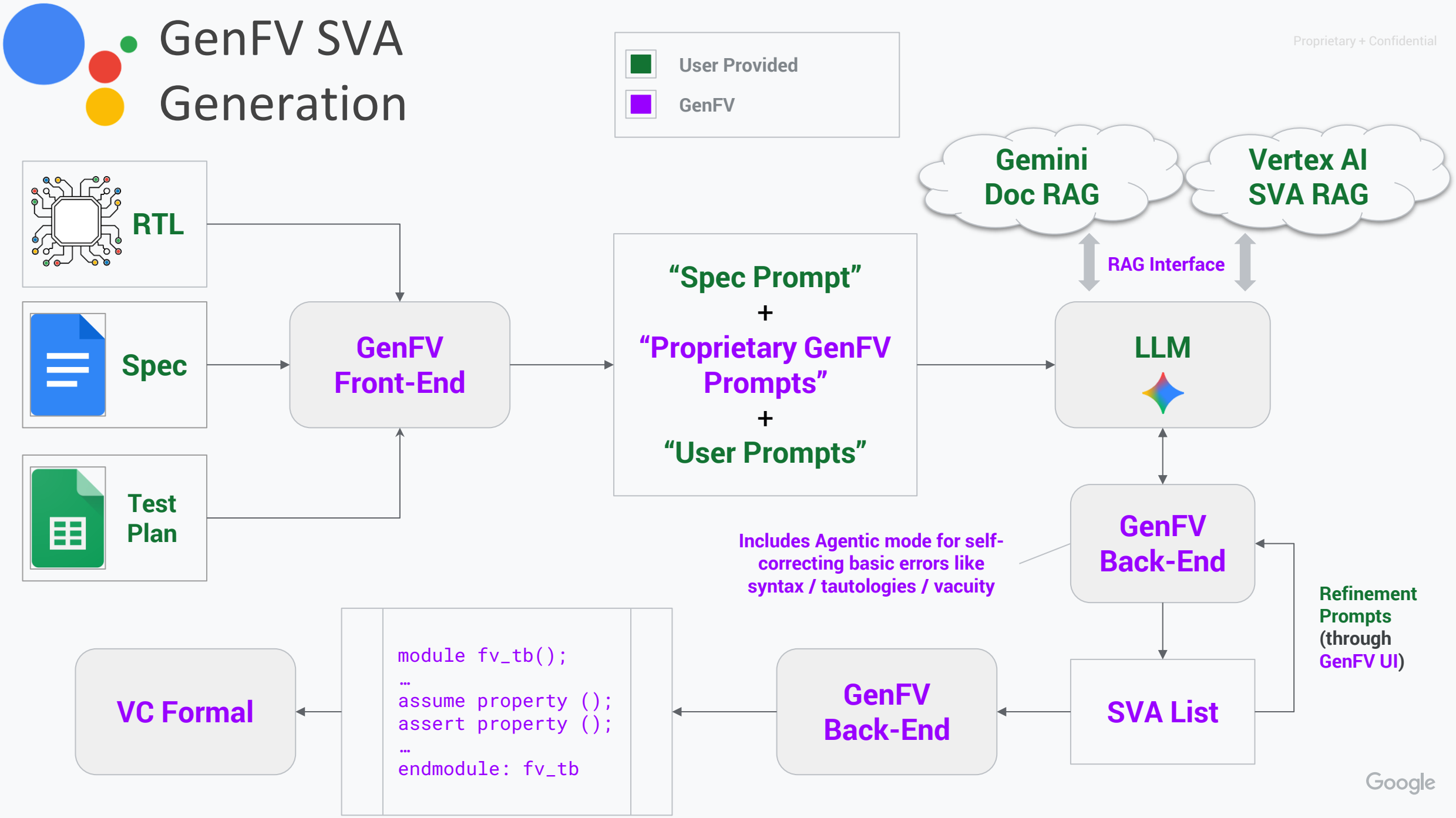
GenFV

GenFV Overview

LLM-enabled Assertion and Formal Verification (FV) accelerator tool developed by Synopsys

1. Specification / Test-Plan to Formal Verification Testbench generation
2. Annotation of existing SVA
3. Helper SVA generation for FV run-time improvement
4. Agentic mode for self-correction of errors
5. Built-in support for Gemini and Google Vertex AI RAG engine

GenFV SVA Generation



GenFV Vertex AI SVA RAG

What is a RAG?

RAG: A database of example queries and correct responses.

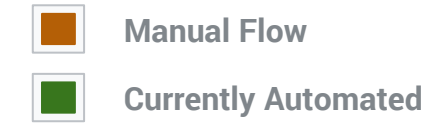
Improves the SVA quality by referring to known examples (**Retrieval**) and injecting these examples into the LLM prompt (**Augmented Generation**).

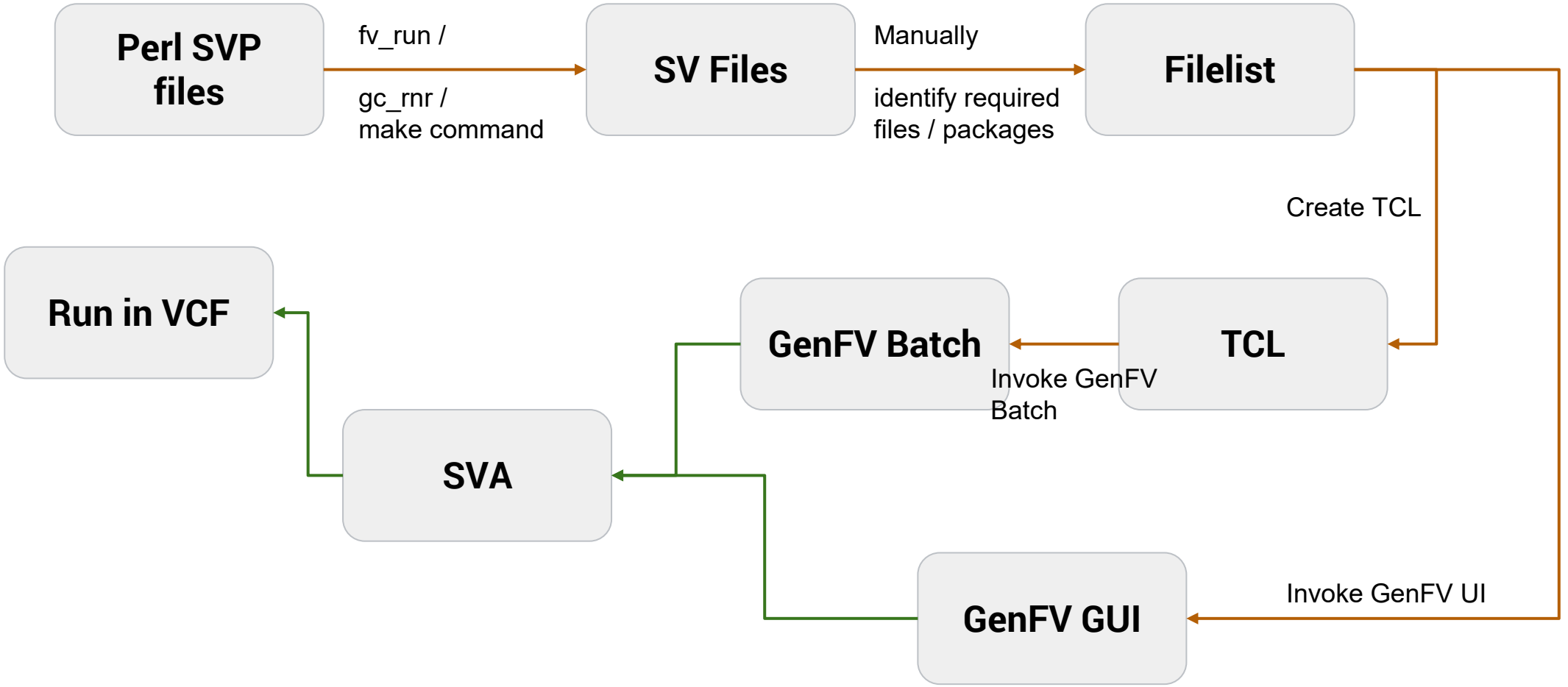
Can be built from the rich dataset of in-house examples across FV setups, and also updated by users at run-time.

Allows improving the LLM without training its weights.

```
// Example of a RAG database entry
{
"Name": "lorem_ipsum_assertion",
"Type": "Assert",
"Query": "Create a SystemVerilog assertion
to check that lorem ipsum implies dolor sit
amet"
"Accepted_response":
"
lorem_ipsum_assertion: assert property(
          @(posedge clk)
          lorem_ipsum |-> dolor_sit_amet
);
"
"Time": "2025-07-29 07:36:02.424108"
}
```

GenFV Usage Standard User Flow





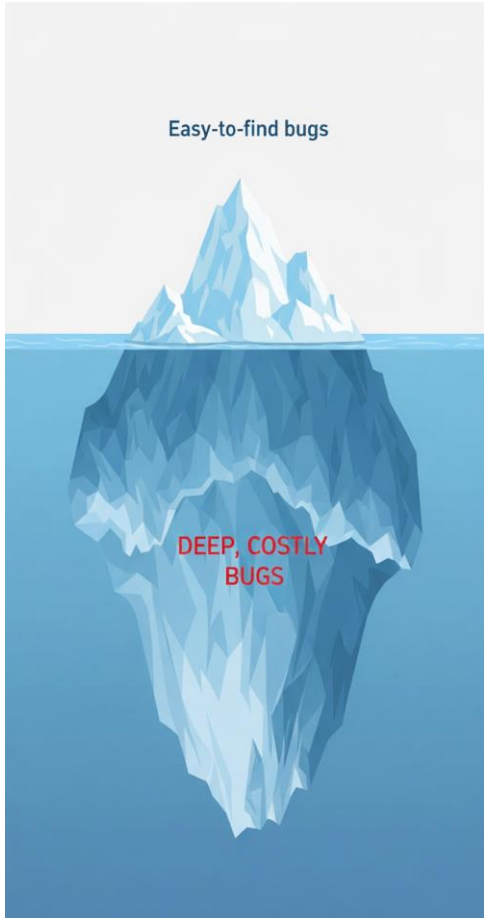
Google

GenFV: Scaling Bounded Proofs to Full proof using AI

manirban@, bindumadhava@

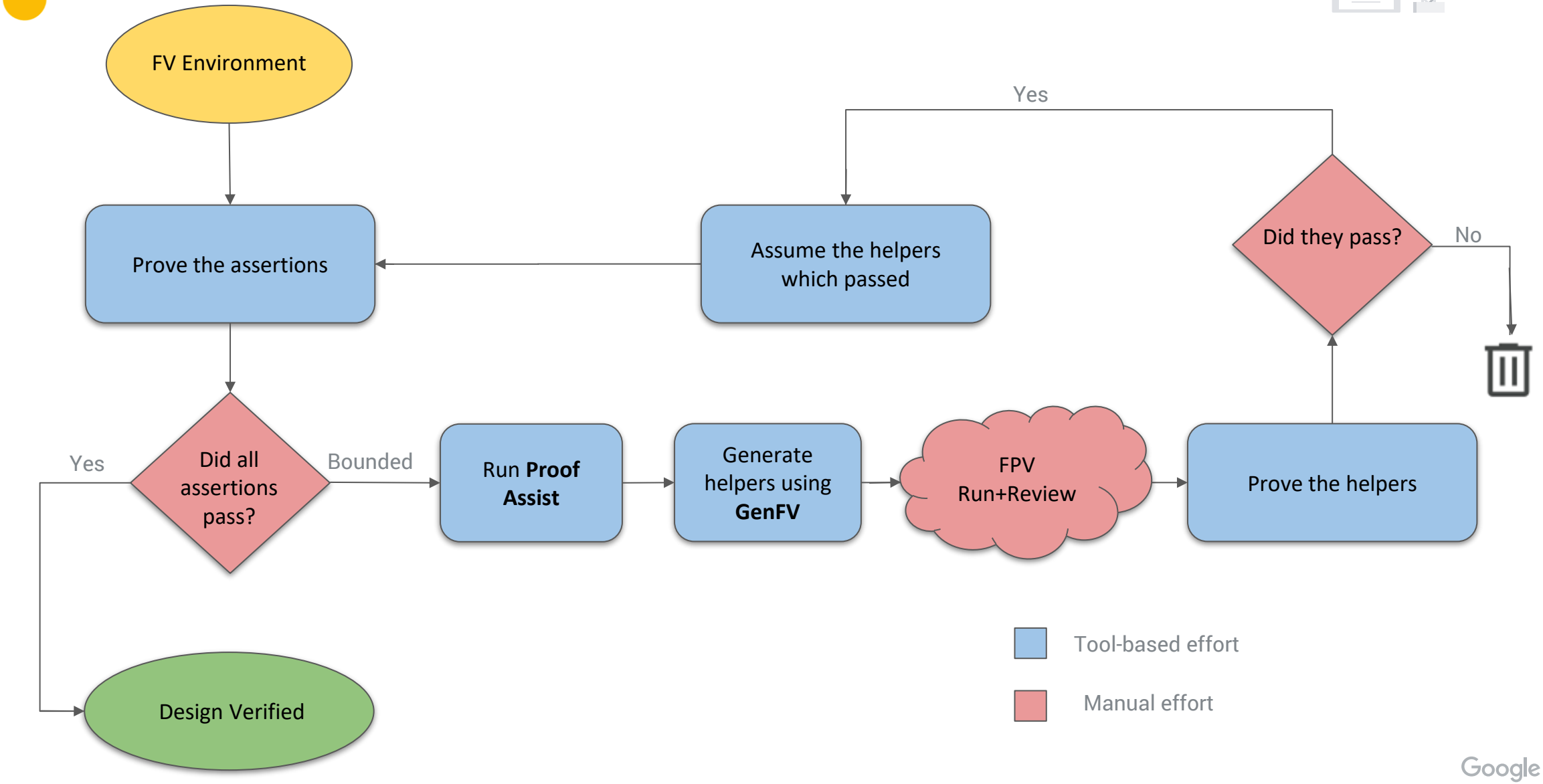
The FV Conundrum: Convergence

- Formal Verification (FV) proves that a design mathematically adheres to a spec.
- FV Good to find corner case bugs which is difficult to find in Simulation
- The problem: Any complex FV environment struggles from convergence issues. Keeps some end-to-end properties “Bounded”
    - ***What*** are bounded properties?: Typical End-to-End properties which the tools find it difficult to prove due to complexity
    - ***Why*** they are not good?: These could be hiding corner case bugs
    - ***How*** to go from Bounded proof to full proof?: Write white-box assertions called helpers which help reduce complexity
    - ***Why this is not done***?: It is a manual effort and requires deep understanding of the RTL, uArch and the FV TB



Can AI help in generation of Helper Properties?

AI Assisted Helper Generation: GenFV



GenFV Journey and Roadmap

Completed WIP Future Steps

Part 1

GenFV Setup with Gemini 1.5 Pro and first level evaluation

Part 2

GenFV Setup with Gemini Document RAG for context aware SVA generation

Part 3

GenFV for RTL-embedded property synthesis

Part 4

Vertex AI SVA RAG setup for GenFV

Part 5

SVA Annotation

Helper Properties

Integration with Gemini 2.5 Pro

Part 6

RAG Fine-Tuning

SYNOPSYS®

2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

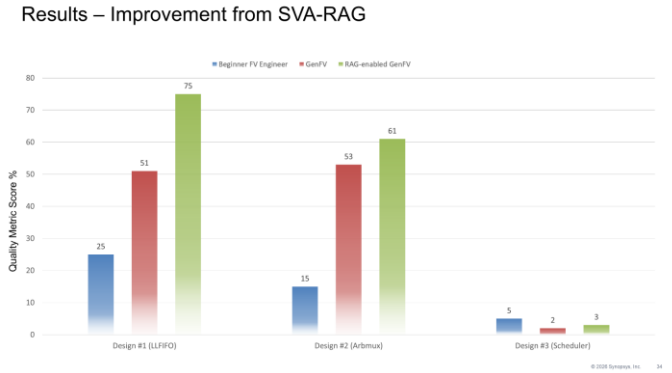
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

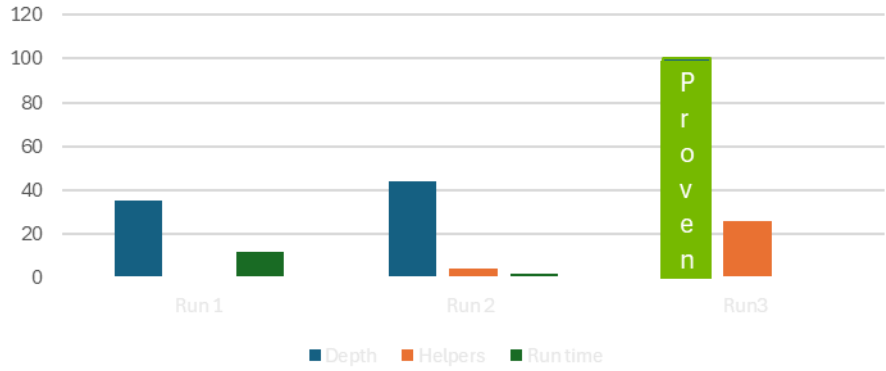
Concluding Remarks

Xiaolin Chen, Synopsys

Summary

- RAG is the key for Formal Advisor Checker Agent
  - Formal Advisor property generation with RAG a boost productivity with superior accuracy
  - Improve VIP/AIP instantiation productivity and accuracy
- Helper assertion flow with Formal Advisor has shown great potential to help achieve full proof
  - Continue improving scalability
- AI may be leveraged for formal test planning
- AI has great potential to improve debugging





Thank You

Xiaolin Chen, Synopsys
Pradip Prajapati, Synopsys
Ghaith Bany Hamad, Nvidia
Bindumadhava S S, Google