

Neuro-Symbolic Natural Language Requirements to Verified Implementations for Model-Based Systems Engineering in SysML v2

This work was funded by DARPA . The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.



Collins Aerospace
An **RTX** Business

Amer Tahat, Senior Engineer.

David Hardin, Chief Technologist.

Isaac Amundson, Fellow Scientist.

Darren Cofer, Principal Senior Fellow Scientist.

May 12, 2026

How can High-Assurance System Development be Conducted using Generative AI Assistance?

- A strength of LLMs is their ability to process natural language and produce code-like artifacts.
 - But, a weakness of this approach is that the generated artifacts have no documented rationale, allocation/traceability, etc.
 - Additionally, hallucinations continue to plague LLM-generated output.
 - A further weakness is that the probabilistic nature of LLMs provides no guarantee that the same requirements will generate the same code from one run to the next.
- This makes LLM-generated code-like artifacts difficult for humans to inspect, evaluate, and maintain, all of which are needed for high-assurance systems development and certification.

High-Assurance Systems Development with LLMs (cont'd.)

- **Observation 1:** We can exploit the strength of LLMs to generate system artifacts that can be assured via formal verification, inspection, and functional/system testing.
 - Thus, we improve the ability to inspect, evaluate, maintain, and most importantly, *understand* the generated artifacts.
- **Observation 2:** We can also utilize the ability of LLMs to produce system artifacts in a number of computer science languages to produce System Architecture Models, generate Data Format Parser Specifications, as well as produce memory-safe language code.
 - Thus, we can utilize LLMs to produce higher-level artifacts that are easier to inspect and assure, as well as achieve DARPA I2O Resilient Software Systems goals.

DARPA PROVERS

Pipelined Reasoning Of Verifiers Enabling Robust Systems

- Develop automated, scalable **formal methods** tools that are integrated into traditional development pipelines using “proof engineering” techniques
- Enable traditional product engineers to incrementally produce and maintain **high-assurance** national security systems
- Collins-led PROVERS Team: **INSPECTA** (Industrial-Scale Proof Engineering for Critical Trustworthy Applications)



Generative AI and PROVERS Technologies

- **Claim: The more code that LLMs generate, the greater the need for PROVERS technologies**
 - In our opinion, it is folly to just generate a “pile of code” from natural language requirements, wherein the human developers have no understanding of the code, its architecture, nor how it was generated.
- Since we’re having a tireless electronic worker generate code, we would be foolish not to have it generate human-inspectable systems models with contracts, as well as memory-safe code with the necessary annotations, kick off the appropriate verification tasks, and have it grind away.
 - Additionally, we can fine tune LLMs on verification tasks to improve their performance on verification tasks.

DARPA PROVERS-Spec Code Proof (SCP) Seedling

Pipelined Neuro-Symbolic Reasoning For Robust Systems

❖ **Assess** the potential of integrating **Gen AI** agents into the **INSPECTA** toolchain:

❑ Can we enable *traditional* product engineers to incrementally produce and maintain high-assurance **national security** systems using **safe, efficient, and GenAI-FM** pipeline?

Barriers to formal methods adoption

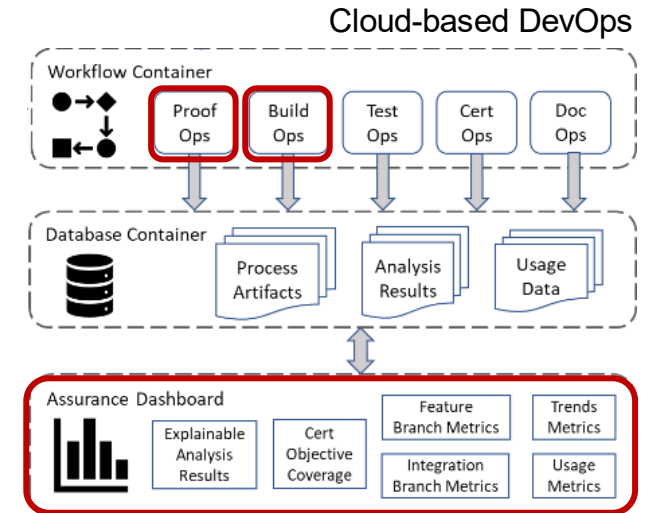
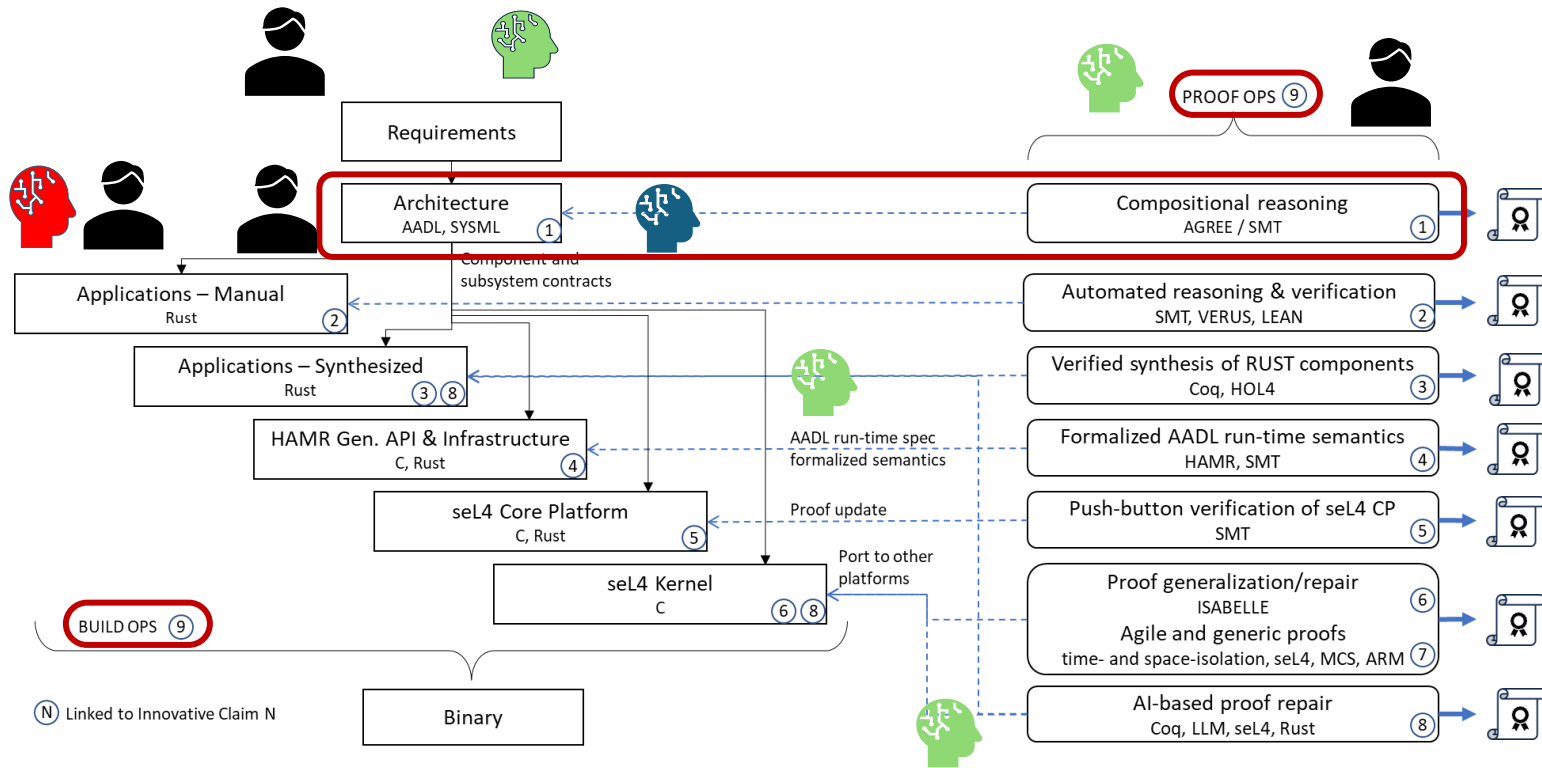
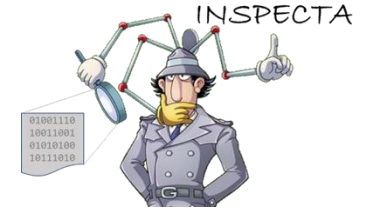
- Scalability to support real-world systems
- The UI targets formal-methods experts, creating a learning curve for others.
- Lack of commercially licensed / supported tools
- Formal methods skillset required
 - Property specification language
 - Explainable counterexamples

GenAI's Potential: Accelerates system development — but **trust, correctness, and consistency** cannot be compromised.

Other Challenges: Computational cost (token/watt), (useful token/time) is exploding.

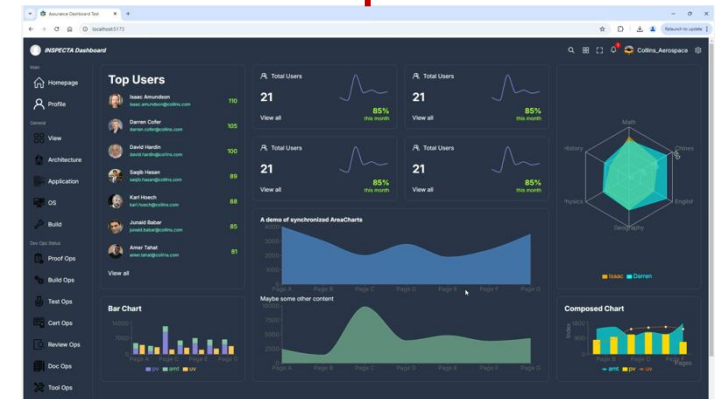
INSPECTA-(SCP-Copilot)

Industrial-Scale Proof Engineering for Critical Trustworthy Applications



Neuro-Symbolic SCP Copilot:

→ Creates a trust fabric for AI-accelerated **MBSE** engineering: (Spec, Code, and Proofs).



Isolette Use Case (First Golden Example and Benchmark)

Table A-14. The Manage Monitor Mode Function Constants

Heat Controller Task natural language functional requirements (control laws)

Component interface

FSE Engineer

Component contract

Contracts incorporated via SysMLv2 language construct (essentially, a comment that HAMR parses, highlights syntax, etc.)

Periodic Thread w/ data ports

...Developer adds application code to contract-annotated skeleton, and verification/testing tools check conformance to contracts.

AADL Model Implied Semantics

Adding application code to skeleton

Application Code in Rust

```

    pub fn timeTriggered-API: thermostat_rt_mhs_Full_Api {
        (old(api).regulator_mode == Isolette_Data_Model::Regulator_Mode::Failed_Regulat
        (api.heat_control == Isolette_Data_Model::On_Off::Off)
        // END MARKER TIME TRIGGERED ENSURES
        // ----- Get values of input ports -----
        let lower: Temp_i = api.get_lower_desired_temp();
        let upper: Temp_i = api.get_upper_desired_temp();
        let regulator_mode: Regulator_Mode = api.get_regulator_mode();
        let currentTemp: TempWstatus_i = api.get_current_tempWstatus();
        //===== compute / control logic =====
        // current command defaults to value of last command (REQ-MHS-4)
        let mut currentCmd: On_Off = self.lastCmd;
        match regulator_mode {
        // ----- Set values of output ports -----
        api.put_heat_control(currentCmd);
        self.lastCmd = currentCmd
        }
    }
  
```

This example shows a complete **end-to-end process**: from requirements, to contracts, to automatically tested and verified application code, to deployment on seL4, Linux, JVM, JavaScript. [2]

Naïve Application of Codex GPT 5 Max to Requirement Files

```
NOTICE FOR THE USER
```

- I started encoding the remaining FAA requirements (hysteresis, "no change", etc.) directly in the SysML GUMBO clauses. To do so I tried to introduce state variables such as `last_alarm_cmd` and `last_heat_cmd` so that we could talk about previous actuator commands (which is necessary for REQ-MA-3 and REQ-MHS-4). However the SysML parser rejects every variant of the `state ...` syntax I've tried, and it also treats `@pre` as an unsupported "classification test" operator. In other words, the current toolchain doesn't appear to accept any of the GUMBO constructs that refer to component state or pre-state values which makes it impossible to capture the remaining requirements faithfully.

Could you share an example (or enable access to the original AADL GUMBO subclause) that shows the specific syntax this environment expects for declaring state variables and referencing their pre/post values? Once I know the accepted form I will finish adding the REQ-MA-3/5 and REQ-MHS-4 cases.

Token usage: total=1,067,603 input=981,770 (+ 18,673,920 cached) output=85,833 (reasoning 54,080)

To continue this session, run `codex resume 019ab4d3-d077-7051-9b23-aea8551e9513`

```
(.venv) amertahat@amertahat-ThinkPad-T15g-Gen-1:~/SPEC_CODE_PROOF/isolette/sysml$
```

Despite 33+ min, ~19M input, 85K output, and 54K reasoning tokens, Codex remains confused, exposes Gumbo pre-training limits and AADL bias, and cannot finish a single Gumbo block task in SysML v2.

Theoretical Background: Transformers Learn In-Context by Gradient Descent

- Von Oswald et al.[3], Transformers Learn In-Context by Gradient Descent, ICML 2023
 - ❑ In-Context Learning (ICL) is equivalent to performing implicit gradient-based fine-tuning on context data during the forward pass.

From Transient Context to Persistent Meta Rules

- **The Core Transformation:**

- We systematically transfer ephemeral In-Context Learning (ICL) knowledge into a permanent, **reusable format**

- **Meta-Rules Definition:**

- A reusable, persistent abstraction that maps transient ICL insights into explicit **formalization logic**

- **Flow:**

- Transient Knowledge → Systematic Extraction → Persistent Asset

Meta-Rules Learning/Reasoning as a Form of Transductive ML and Inference

Inductive Learning / Proof Analogy Inductive Inference

- **Observe Examples (base cases)**
 - $2+2=4$, $2+6=8$,
 $4+6=10$
- **Form of hypothesis (inductive hypothesis)**
 - (even + even is even)?
 - $n + m = k$, where n , m , and k are all even nat?
- **Generalize and Prove it**
 - Use hypothesis to justify ***all*** similar future cases
 - **For all** n, m : even nat: even + even is correct.

Transductive Learning [4] / Our Meta-Rules Transductive Inference

- **Observe Examples (task context)**
 - Prompt examples, repair history, successful patterns
- **Transductive hypothesis (Meta-Rule candidate(s))**
 - Rule candidates are extracted from context and conversation history
- **Generalize and Specialize (*refine*** them using verifier feedback
- **Retain only** rules that help produce accepted artifacts under **budget**

Takeaway: Meta-rules are not expected to be correct on the first iteration. They become useful only when iterative generation, repair, and verifier feedback drive them toward an acceptable state

Examples of Contract Embeddings Meta-Rules

When and how to use GUMBO Spec

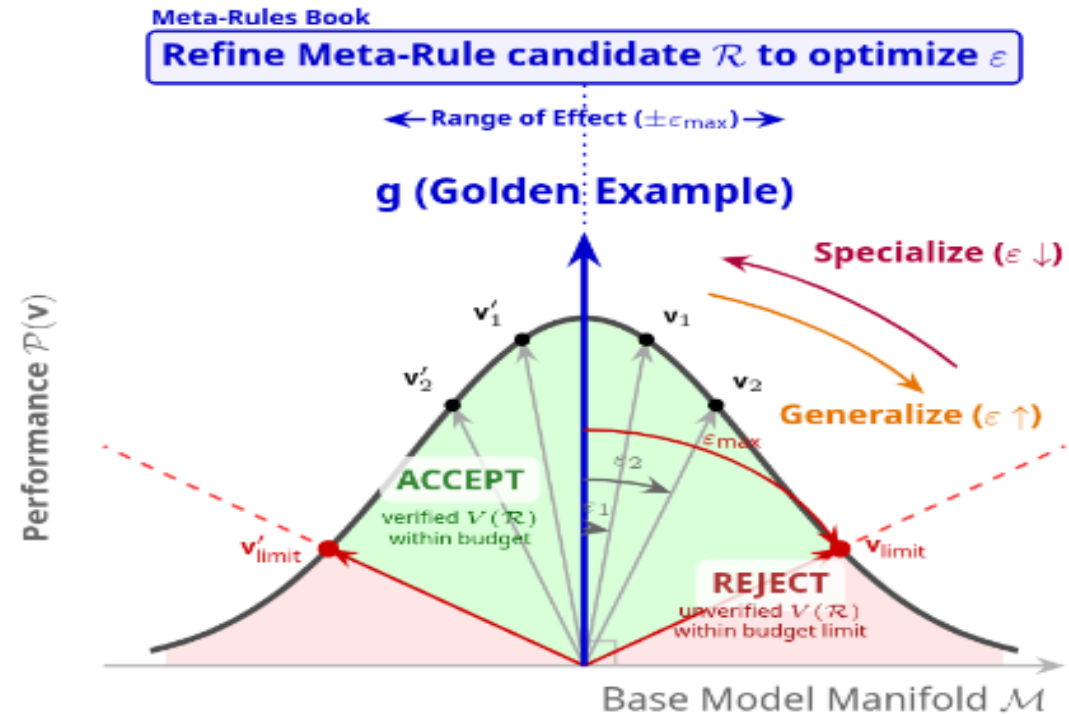
```
language "GUMBO" /*{  
  state  
  // (optional) persistent variables for  
  // previous outputs, timers, accumulators  
  lastCmd: Isolette_Data_Model::On_Off;  
  
  functions  
  // (optional) pure helper definitions; must  
  // be side-effect-free
```

```
R1) "If <guard>, then <out> shall be <value>"  
  →  
  case <ID> "...|<pdf#page=N>":  
    assume <guard>;  
    guarantee <out> == <value>;
```

Practically Verified Semantic Cone

Meta-Rules Extraction and Validation

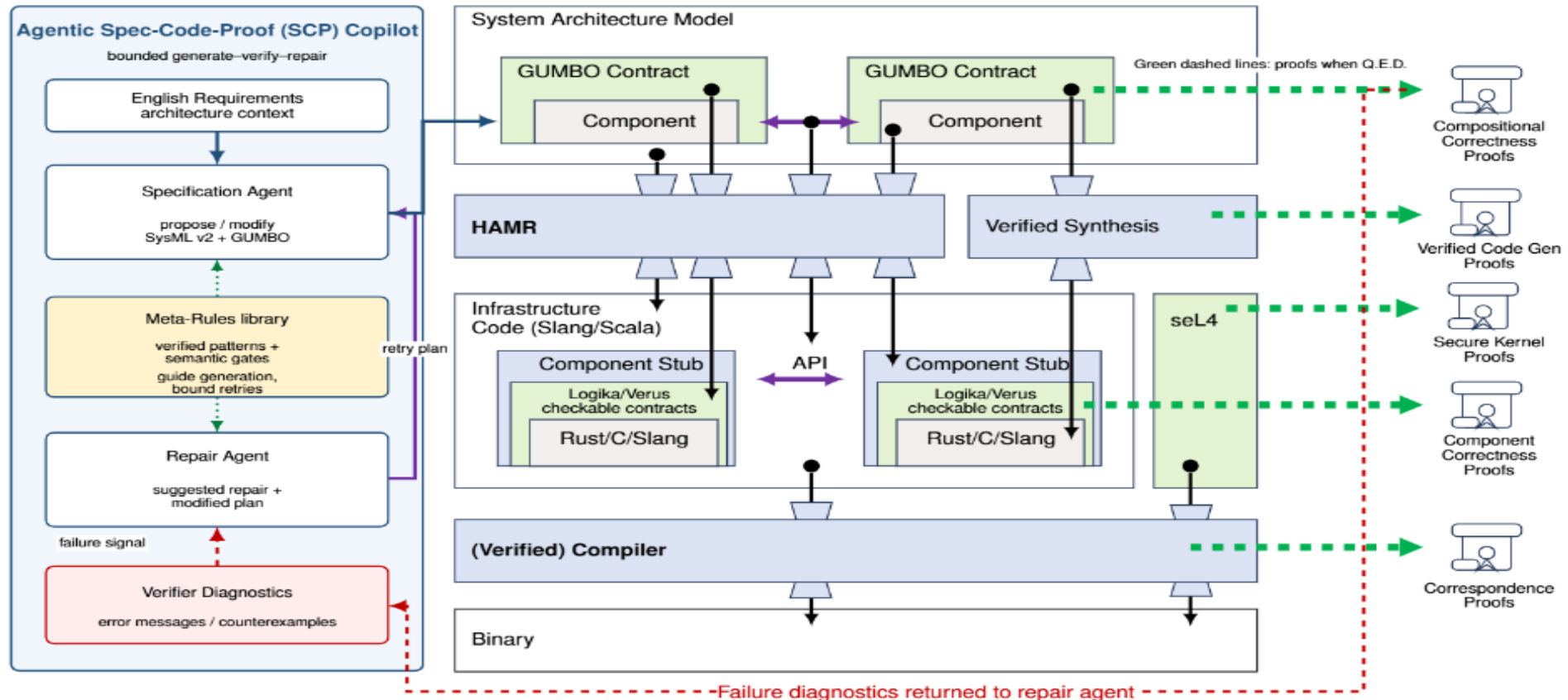
- Start from a golden example: requirement + verified contract g .
- Extract candidate rule r ; test it on a small sample of R . For a given ϵ ,
- Accept generated r only:
 1. Generation stays within of ϵ
 2. Verify/repair under budget B keep r if enough samples pass,
 3. Otherwise, reject, refine with human guidance, or change ϵ then repeat.
- Goal is to optimize ϵ for a given budget



R = requirements, g = golden contract, \hat{g} = generated contract, ϵ = similarity threshold, B = budget, m = candidate Meta-Rule

Practically Verified Semantic Cone: Refine Meta-Rule candidates R to optimize ϵ as a verifiable surrogate for fine-tuning. The rule creates a local performance maximum centered on g that we aim to expand via Generalization while remaining within the budget-verifiable (ACCEPT) region.

Agentic SCP/INSPECTA Copilot workflow



Agentic SCP/INSPECTA workflow integrating the high-assurance development layers with a Meta-Rule-SCP Copilot guided generate-verify-repair loop. Green dashed arrows denote successful proof artifacts; the red dashed arrow denotes verifier-failure diagnostics returned to the repair agent for a bounded retry.

Sample Copilot Transcript (Gumbo Generation and Repair)

464
465
466
467
468
469

```
out port internal_failure : DataPort { out := type : Isolette_Data_Model::Failure_Flag_i; }
```

Symbolic Hallucination Detection

Meta-Rule-guided, Verifiable repair

```
assume regulator_mode == Isolette_Data_Model::Regulator_Mode.Normal_Regulator_Mode;  
guarantee displayed_temp.degrees == round_to_nearest_int(current_tempwstatus.degrees);  
guarantee displayed_temp.degrees == current_tempwstatus.degrees;
```

F/V System Integration-Level check

- Ran sireum hamr sysml logika --sourcepath isolette/sysml
- Ignoring System to Thread connection: Monitor_Temperature_i_Instance.upper_alarm_tempwstatus -> mmi.upper_alarm
- +22 lines
- * file:///home/amertahat/SPEC_CODE_PROOF/isolette/sysml/Regulate.sysml
- [182, 14] An integration clause must refer to exactly one port

Detection & Explanation Provided

The integration clause error arises because integration assumes constraints involving multiple ports, which violates the rule that each assume in integration must reference exactly one port. The plan is to remove cross-port assumptions from the integration and relocate ordering assumptions either into the compute layer or split them appropriately. Next, I'll inspect the relevant lines around 170-190 in the system model to confirm and adjust the assumptions accordingly.

- Explored
 - Read Regulate.sysml
- Identifying cross-port integration assumption issues (23s • see to interrupt)

Neuro-Symbolic Analysis (cross-port integration assumptions issues)

59% context left · ? for shortcuts

Reasoning tokens are being saved while Symbolic tools do the analysis

Full Video: <https://youtu.be/BGtiUfd8LCQ> SCP-Codex gpt 5.2 max (AFTER Meta-Rules). (Regulate) short version, complete code level verification success.

Exp-1:Tasks Completed- Isolette Benchmark (After introduction of Meta-Rules)

100% of tasks completed fully automatically, covering:

- English → Formal Gumbo
- Gumbo verification (Hamr/Logika)
- Integration verification
- Hamr → | Scala code generation
- Hamr/Logika code-level verification

Exp-2: SCP Self-Healing For Slang (Scala subset) Application Logic

Test Harmonization Tasks

- **Self-Healing for Slang application logic**
- **Human Interaction: Minimal.** The workflow executes automatically, requiring only high-level approval
- **Build, Run, and Verification Success** Compiles cleanly and passes all Sireum/Logika checks.

```
• Edited isolette/hamr/slang/src/test/bridge/isolette/bridge/isolette.scala
23   defaultInitializeConfig(verbose = verbose)
24 -   defaultComputeConfig(verbose = verbose)
25   defaultComputeWLConfig(verbose = verbose)
24 +   defaultComputeConfig(verbose = verbose)
25   )

• Patched the test harnesses to only use configs
- Removed defaultInitializeConfig from ManageTest
- Removed defaultComputeWLConfig from ManageTest

Edits:
- isolette/hamr/slang/src/test/bridge/isolette/bridge/isolette.scala
- isolette/hamr/slang/src/test/bridge/isolette/bridge/isolette.scala
- ./sireum proyek run isolette/hamr/slang isolette

> yes

✓ You approved codex to run ./sireum proyek run isolette/hamr/slang isolette

↳ Interacted with background terminal ./sireum proyek run isolette/hamr/slang isolette
  (waited)

• Ran ./sireum proyek run isolette/hamr/slang isolette
  Compiling module: Isolette_Single_Sensor_Instance
  ... +118 lines
  { "log" : "info", "title" : "Isolette_Single_Sensor_Instance_1771567227434" }

• Build and run succeeded. Main and test sources are up-to-date.
  If you want the app to keep running interactively, use the command:
  > isolette/hamr/slang/bin/run-logika.cmd

↳ Interacted with background terminal ./isolette/hamr/slang/bin/run-logika.cmd
  (waited)

• Ran ./isolette/hamr/slang/bin/run-logika.cmd
  Initializing runtime library ...
  ... +60 lines
  Verification succeeded in 15 s!
```

Key Result:

- 100% of tasks completed highly automatically, covering:
 - English → Formal Gumbo Contracts (Previous Experiment)
 - HAMR/Logika Scala Test harmonization
 - Build and run application code successfully
 - Gumbo verification (HAMR/Logika)
 - Integration verification

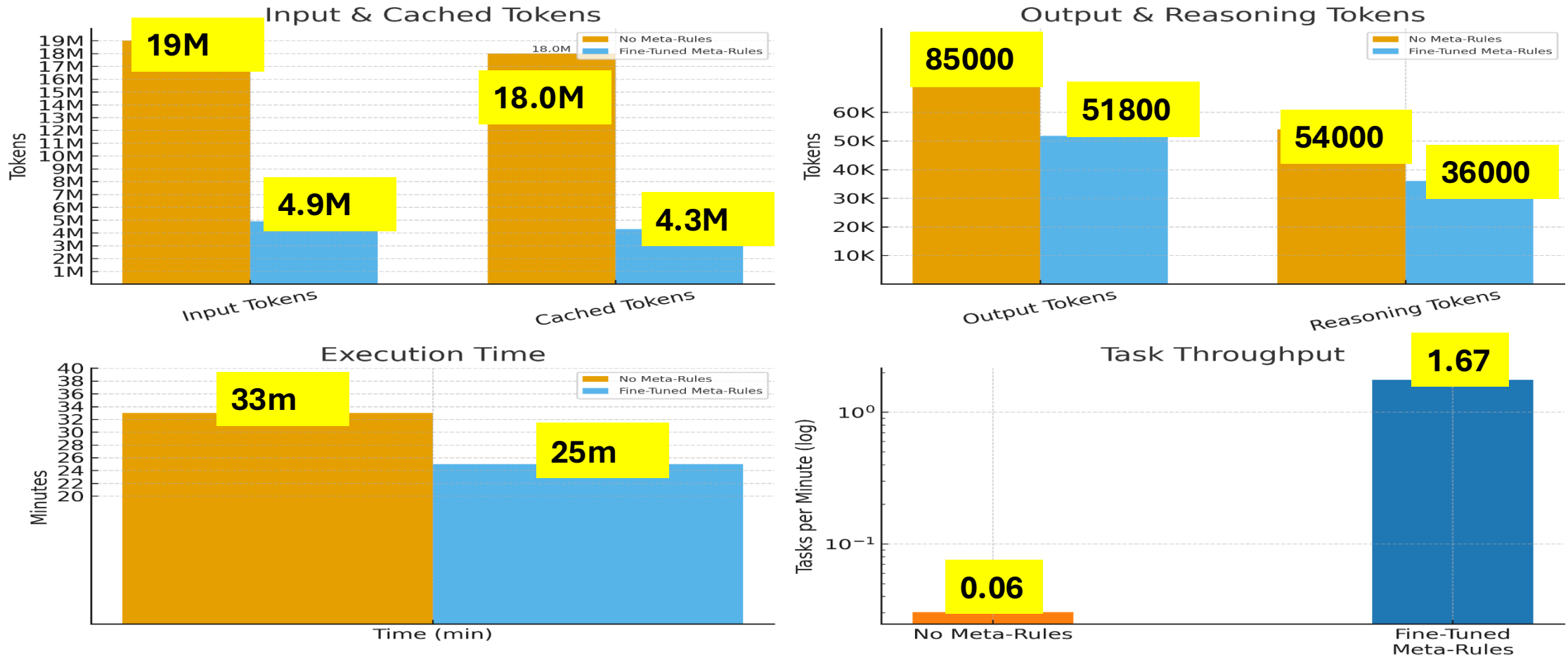
Despite ~17 initial type errors (repaired), SCP achieved 100% formal verification (FV) repair success, with ~12,209 output tokens and ~3,320 reasoning tokens.
Total tokens used: ~3.3 million (~3 million cached) end-to-end. < 20 minutes

Verification Cost and Coverage Metrics

Metric	Baseline (prompt-only)	Meta-Rule-guided SCP
T_{wall} (minutes)	33	25
N_{in}	981,770	317,485
N_{cache}	18,673,920	4,302,976
N_{out}	85,833	51,833
N_{reason}	54,080	36,032
N_{total}	19,741,523	4,672,294
Γ_{net} (tokens/min)	$\approx 598K$	$\approx 187K$
$ U $ (targeted proof-bearing units)	42	42
$ U_{\text{verified}} $ (verified units)	2	42
ρ (verification coverage)	4.76%	100%
Θ (verified throughput, units/min)	0.06	1.68
κ (token cost per verified unit)	$\approx 9.87M$	$\approx 111K$
S (end-to-end success)	0	1

Meta-Rule-guided SCP reaches full verification coverage while reducing total token use, token burn rate, and cost per verified unit compared with the prompt-only baseline.

Experimental Evaluation of the Incumbent System



Performance comparison: Before (Orange) vs. After Meta-Rules (Blue) shows a massive reduction in input/output tokens and execution time, while Task Throughput increases significantly.

Meta-Rules Compared with Fine-Tuning and Prompting ICL

	Fine-tuning	Prompting/ICL	Meta-Rules
Persistent across runs	Yes	No	Yes (external)
Model weights tuning	Explicit	Implicit	Implicit
Cost	High	Med	Low
Auditability / review	Low	Low-Med	High
Governance / rollback	Low	Low	High
Runtime context overhead	Low	High	Low-Med
Data / IP constraints	High	Low-Med	Low
FV acceptance gates	N/A	N/A	Primary

Takeaway: Meta-Rules provide persistent, auditable specialization without modifying model weights, while preserving formal-verification acceptance gates for high-assurance workflows.

Conclusions

- **Summary:**

- Built a reusable, Spec-Proof-Code Copilot for SysML v2, integrated with INSPECTA HAMR/Logika and supporting Scala/Slang implementations.
- Demonstrated proof-of-concept on the Isolette use case:
 - +40 Contracts.
 - English requirements extracted from 37 appended pages specific to Isolette of a multi-system 147-page PDF documentation: tables, figures, and plain English texts.
 - Documentation complexity representative of realistic industrial settings.
 - **Scala/slang self-healing application logic, Build, Run test, FV the app.**
- Integrated into VS Codium IDE, achieving significant cost reduction via novel optimizations.

- **Up Next:**

- Support Rust/Verus on seL4 build, and SysEng agents. **(Collins, KS, KU, & Stanford)**
- Evaluate on DornerWorks use case.

- **Demo:**

- **Video:** See Spec-Proof-Code SysMLv2 Copilot here:
 - <https://youtu.be/BGtiUfd8LCQ> **(AFTER)**, https://youtu.be/Pmvp7g37_Nk **Codex-baseline gpt 5.1 max (BEFORE Meta-Rules)**

A Shared Research Enclave for Reasoning-Model Adaptation

Shared Research Enclave

1. Program-approved snapshots and images can be carried forward for follow-on exploration, integration, and productization—without exporting proprietary training data.
2. Create approved model or container images at key milestones so performers can reproduce and extend results.



**Takeaway: The missing link:
Shared compute, governed checkpoints, and exportable images transform isolated demos into
scalable, auditable model-adaptation workflows.**

Thank You! Questions?

This work was funded by DARPA .

The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.



References

[1] David L. Lempia and Steven P. Miller, FAA Requirements Engineering Management Handbook, 2009

[2] J. Hatcliff, “Model-based development for seL4 Microkit/Rust with integrated formal methods using HAMR,” Presentation at the seL4 Summit 2025, 2025, accessed: 2025-05-19. [Online]. Available: <https://sel4summit2025.sched.com/event/26GD3>

[3] J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov, “Transformers learn incontext by gradient descent,” in Proceedings of the 40th International Conference on Machine Learning. PMLR, 2023, pp. 35 151–35 174.

[4] A. Achille and S. Soatto, “AI agents as universal task solvers,” Entropy, vol. 28, no. 3, 2026. [Online]. Available: <https://www.mdpi.com/1099-4300/28/3/332>

