

---

# Supplementary Material for: *SpecMAS: A Multi-Agent System for Self-Verifying System Generation via Formal Model Checking*

---

Rishabh Agrawal<sup>1\*</sup> Kaushik Tushar Ranade<sup>1\*</sup> Aja Khanal<sup>1</sup>

Kalyan S. Basu<sup>2</sup> Apurva Narayan<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Western Ontario, London, Ontario

<sup>2</sup>ICASSD, Cambridge, Ontario

{ragrawa9, kranade, akhanal3, apurva.narayan}@uwo.ca ks.basu@gmail.com

## Overview

This supplementary material provides additional results, implementation details, and experimental settings that complement the main paper.

## 1 Implementation Details

### 1.1 Model Choice and Configuration

For our experiments, we use two models as our LLM backbone. For general execution and planning, we use Qwen 2.5 32b 8bit[13] quantized, and for tool calling Qwen 3 32b 8bit quantized [14]. Similarly, for LLM as Judge evaluation, we use the Full Qwen 3 32B model with thinking mode. To compare the effectiveness of our system, we put it up against these models.

1. Claude 3.7 Sonnet with extended thinking [3]
2. Gemini 2.5 pro [12]
3. Qwen 3 235A-22B MOE with thinking [14]
4. DeepSeek R1 [7]

All of the above models were used through their respective Web interfaces to ensure consistency in output generation.

---

\*Equal Contribution

## 2 SpecMAS Walkthrough: SOP\_01 (Shuttle Guidance Controller)

This section illustrates the SpecMAS pipeline on SOP\_01 (Shuttle Guidance Controller), showing intermediate outputs at each step.

### Step Map

Step	Input	Output (artifact)
1. State and dynamic variables extraction	sop_01.pdf	states.smv
2. Transition extraction	sop_01.pdf, states.smv	trans.smv
3. Kripke structure formalization	sop_01.pdf, states.smv, trans.smv	kripke.smv
4. Property mining (CTL/LTL)	sop_01.pdf	properties.smv
5. Final NuSMV model synthesis	kripke.smv, properties.smv	model.smv
6. Debugging & counterexample refinement	model.smv	model.smv

Table 1: Pipeline for SOP\_01. Intermediate outputs shown in the following subsections.

### 2.1 Step 1: State and Dynamic Variables Extraction

**Goal.** Identify all system states and monitored parameters.

**Input.** sop\_01.pdf

**Output.** states.smv

Listing 1: Excerpt from states.smv

```
VAR
  -- States from SOP: flight modes and operational states
  system_state: {normal_operation, emergency_operation, mm102, mm103,
    mm601};
  -- Dynamic variables from SOP monitoring parameters
  step: {undef, 1, a1, 2, exit}; -- From 11.1 System States step
  transitions
  region: {reg1, reg2, reg3, reg4, reg102, reg_1}; -- reg_1 is
    initial state from 3.1.1
  velocity: nat; -- Velocity in m/s (unbounded unless specified)
  altitude: nat; -- Altitude in meters
  dynamic_pressure: nat; -- q bar in Pascals
  angle_of_attack: nat; -- Angle in degrees
  apogee: nat; -- Apogee in meters
  hdot: nat; -- Vertical speed in m/s
  cont_3EO_start: boolean; -- From 2.1 Flags
  RTLS_abort_declared: boolean; -- From 2.1 Flags
  region_selected: boolean; -- From 2.1 Flags
  MECO: boolean; -- Main Engine Cut Off condition

ASSIGN
  -- Initial values from SOP 3.1.1 Startup Procedure
  init(step) = undef; -- "Initialize step to undefined"
  init(region) = reg_1; -- "Initialize region register to reg-1 (
    unset)"
  init(cont_3EO_start) = FALSE; -- "Set all internal control flags to
    default or false"
  init(RTLS_abort_declared) = FALSE;
  init(region_selected) = FALSE;
  init(MECO) = FALSE;
  init(system_state) = normal_operation; -- Default state unless
    triggered
```

## 2.2 Step 2: Transition Extraction

**Goal.** Encode SOP conditional flows as guarded transitions

**Input.** sop\_01.pdf, states.smv

**Output.** trans.smv

Listing 2: Excerpt from trans.smv

```
NEXT(region) := case
  -- Initialize to reg102 during startup (3.1.1)
  step = undef: reg102;
  -- Assign reg102 for early ascent (mm102 mode) (11.2.1)
  mode = mm102: reg102;
  -- Assign reg1 or reg2 based on apogee/dynamic pressure (3.2.1,
    11.2.1)
  apogee_comparison: reg2;
  velocity_threshold: reg1;
  -- Transition to reg102 if velocity exceeds limit (11.2.1)
  velocity_above_limit: reg102;
  TRUE: region;
esac;

NEXT(region_selected) := case
  -- Set TRUE after region assignment (3.1.2, 4.1)
  (region = reg1 OR region = reg2 OR region = reg102): TRUE;
  -- Reset if emergency condition no longer exists (3.2.2)
  NOT cont_3EO_start AND NOT RTLS_abort_declared: FALSE;
  TRUE: region_selected;
esac;

NEXT(cont_3EO_start) := case
  -- Trigger when MECO occurs and velocity is low (4.1)
  MECO_confirmed AND velocity_low: TRUE;
  -- Reset if region not selected (3.2.2)
  NOT region_selected: FALSE;
  TRUE: cont_3EO_start;
esac;

NEXT(RTLS_abort_declared) := case
  -- Set when RTLS conditions met (4.1)
  region = reg2 AND velocity_above_threshold: TRUE;
  -- Reset if abort no longer needed (4.2)
  NOT cont_3EO_start: FALSE;
  TRUE: RTLS_abort_declared;
esac;

NEXT(step) := case
  -- Startup sequence (3.1.1)
  step = undef: 1;
  -- Transition through logic steps (11.1)
  step = 1: a1;
  step = a1: 2;
  step = 2: exit;
  TRUE: step;
esac;

-- Uncovered:
-- reg3/reg4 region logic (SOP mentions reg1-reg4 but only reg1, reg2
  declared)
-- mm103/mode601 transitions (not mapped to declared states)
-- step "exit" behavior (no defined post-exit state)
-- "Dump" and "pitch adjustment" commands (unmodeled dynamic variables
  )
-- Sensor threshold calculations (abstracted as Boolean guards)
```

## 2.3 Step 3: Kripke Structure Formalization

**Goal.** Combine states and transitions into a Kripke structure.

**Input.** sop.01.pdf, states.smv, trans.smv

**Output.** kripke.smv

Listing 3: Kripke structure fragment

```
-- Merged Triggers:
-- velocity_threshold: reg1, apogee_comparison: reg2,
   dynamic_pressure_check: toggle
-- cont_3EO_start (4.1), RTLS_abort_declared (2.1), region_selected
   (2.1), mm102 (2.2), MECO (3.1.2)
-- reg102 initialization (3.1.1), step transitions (11.1)

VAR
  region: {reg1, reg2, reg102};           -- Declared system states
  region_selected: boolean;               -- Flag for successful region
      assignment
  cont_3EO_start: boolean;               -- Trigger for emergency mode
  RTLS_abort_declared: boolean;          -- RTLS abort flag
  step: {undef, 1, a1, 2, exit};         -- Logic flow state
  mode: {mm102, mm103, mm601};           -- Flight mode (untracked in
      output, but referenced in SOP)

NEXT(region) := case
  -- Initialize to reg102 during startup (3.1.1)
  step = undef: reg102;
  -- Assign reg102 for early ascent (mm102 mode) (11.2.1)
  mode = mm102: reg102;
  -- Assign reg1 or reg2 based on apogee/dynamic pressure (3.2.1,
    11.2.1)
  apogee_comparison: reg2;
  velocity_threshold: reg1;
  -- Transition to reg102 if velocity exceeds limit (11.2.1)
  velocity_above_limit: reg102;
  TRUE: region;
esac;

NEXT(region_selected) := case
  -- Set TRUE after region assignment (3.1.2, 4.1)
  (region = reg1 OR region = reg2 OR region = reg102): TRUE;
  -- Reset if emergency condition no longer exists (3.2.2)
  NOT cont_3EO_start AND NOT RTLS_abort_declared: FALSE;
  TRUE: region_selected;
esac;

NEXT(cont_3EO_start) := case
  -- Trigger when MECO occurs and velocity is low (4.1)
  MECO_confirmed AND velocity_low: TRUE;
  -- Reset if region not selected (3.2.2)
  NOT region_selected: FALSE;
  TRUE: cont_3EO_start;
esac;

NEXT(RTLS_abort_declared) := case
  -- Set when RTLS conditions met (4.1)
  region = reg2 AND velocity_above_threshold: TRUE;
  -- Reset if abort no longer needed (4.2)
  NOT cont_3EO_start: FALSE;
  TRUE: RTLS_abort_declared;
esac;

NEXT(step) := case
```

```

-- Startup sequence (3.1.1)
step = undef: 1;
-- Transition through logic steps (11.1)
step = 1: a1;
step = a1: 2;
step = 2: exit;
TRUE: step;
esac;

-- Uncovered:
-- reg3/reg4 region logic (SOP mentions reg1-reg4 but only reg1, reg2
  declared)
-- mm103/mode601 transitions (not mapped to declared states)
-- step "exit" behavior (no defined post-exit state)
-- "Dump" and "pitch adjustment" commands (unmodeled dynamic variables
  )
-- Sensor threshold calculations (abstracted as Boolean guards)

```

## 2.4 Step 4: Property Mining (CTL/LTL)

**Goal.** Extract safety/liveness properties from SOP rules.

**Input.** sop\_01.pdf

**Output.** properties.smv

Listing 4: CTL properties

```

-- Specifications

SPEC AG ((mode = mm102) -> ((mode != mm601) | emergency_condition))
SPEC AG (region_selected -> region in {reg1, reg2, reg3, reg4, reg102
  })
SPEC AG ((step = s0) -> (!cont_3EO_start & !RTLS_abort_declared))
SPEC AG (!((step = s1) & region_assignment_blocked))
SPEC AF ((abnormal_flight_status & MEC0_confirmed) -> cont_3EO_start)
SPEC AF ((mode = mm601) -> (pitch_maneuver_executed &
  dump_maneuver_executed))
SPEC AF (region_assignment_conditions -> region_selected)
SPEC AG (environmental_threshold_exceeded -> emergency_condition)
SPEC AG (security_verified -> tamper_detection_clear)
SPEC AG (maintenance_complete -> validation_complete)
SPEC AG (document_authorized -> document_authorized_event)
SPEC AG (abnormal_flight_status <-> (flight_status = abnormal))
SPEC AG (region_assignment_conditions -> (region != reg1_unset))
SPEC AG ((step = s_exit) -> !emergency_condition)

```

## 2.5 Step 5: Final NuSMV Model Synthesis

**Goal.** Merge Kripke structure and properties into one model.

**Input.** kripke.smv, properties.smv

**Output.** model.smv

In this step, the outputs of the Kripke structure formalization (Step 3) and property mining (Step 4) are merged into a unified NuSMV model. This integration introduces several structural refinements that were not explicit in the intermediate artifacts. Firstly, the state variables and transitions extracted in Step 3 were consolidated into a single MODULE main, ensuring a consistent and centralized representation of the system. Each variable domain was tightened to be finite and closed, and explicit initialization assignments were added to align with the SOP's pre-launch conditions. This eliminated the underspecified or open-ended ranges present in the earlier Kripke description.

Second, transition rules that were originally overlapping or incomplete were normalized into mutually exclusive and exhaustive case statements. Default self-loops were explicitly added to guarantee finiteness, preventing deadlocks when no conditions fired. Guards that were previously encoded with cross-dependencies were simplified into current-state predicates, with derived conditions factored into DEFINE macros. This restructuring preserved the semantics of the Kripke structure while making the NuSMV specification deterministic and verifier-friendly.

Then, the safety and liveness properties mined in Step 4 were integrated directly into the model as both SPEC and, where appropriate, INVAR statements. Encoding safety constraints as invariants not only reduced vacuity but also pruned unreachable states from the transition system, thereby tightening the correspondence between the SOP’s intent and the model checker’s search space. Finally, sanity checks were introduced to rule out invalid combinations (e.g., abort signals raised without region assignment), ensuring that the unified model.smv reflected the SOP more faithfully than the separate intermediate artifacts.

Overall, Step 5 produced a self-contained model that combined the structural completeness of the Kripke formalization with the semantic guarantees of mined temporal properties. Compared to the raw outputs of Steps 3 and 4, the synthesized NuSMV model is deterministic, closed under its transition relation, and directly amenable to verification. The complete artifact is provided in the supplementary ZIP, while here we highlight some of the key structural improvements.

Listing 5: Safety-Property refactored as INVAR

```
-- From Step 4 (kept verbatim)
SPEC AG (RTLS_abort_declared -> region != reg1_unset)

-- Step 5: also enforce at the transition relation level
INVAR !(RTLS_abort_declared & (region = reg1_unset))
```

Listing 6: Derived predicate moved to Define to simplify guards

```
ASSIGN
  next(RTLS_abort_declared) := case
    rtls_ready : TRUE;
    TRUE       : RTLS_abort_declared;
  esac;

DEFINE
  rtls_ready := (region = reg1) & (velocity < 70);
```

Listing 7: Closed open domains to finite ranges

```
-- Before (Step 3): unbounded / implicit
VAR velocity : integer;

-- After (Step 5): bounded (prevents ghost states)
VAR velocity : 0..500;
```

Listing 8: Aligning Kripke and SMV INIT block

```
-- Before
-- (initial state implied in narrative)

-- After
ASSIGN
  init(mode)    := mm102;
  init(region)  := reg1_unset;
  init(velocity) := 0;
```

## 2.6 Step 6: Debugging & Counterexample Refinement

**Goal.** Run NuSMV, identify counterexamples, and refine the model. **Input.** `model.smv` **Output.** Updated `model.smv`

In the final stage, we resolved syntax inconsistencies in the aggregated model and used counterexample traces to refine the semantics of the NuSMV specification. Several minor issues in the Step 5 artifact—such as undeclared identifiers, incomplete initialization, and non-total transition assignments—were corrected to ensure that the model compiled deterministically from a single well-defined initial state.

Beyond syntax, counterexamples highlighted cases where the model’s encoding diverged from the intended SOP semantics. For example, we observed that `security_verified` and `tamper_detection_clear` could both become TRUE, violating their intended mutual exclusion; that `maintenance_complete` could arise after `validation_complete` had already reverted, breaking monotonicity; and that `document_authorized` could hold without a preceding `document_authorized_event`, blurring the distinction between events and states. Similarly, environmental conditions such as `environmental_threshold_exceeded` were not reliably propagated to trigger `emergency_condition`, and exit states could be reached while the system remained in emergency. Each of these traces motivated targeted refinements: latching or gating variables to prevent non-monotone behavior, restructuring events as one-cycle pulses, and strengthening invariants to eliminate unreachable or contradictory states.

The resulting `model.01.smv` preserves all passing properties from Step 5 (e.g., mode constraints, abnormal-flight equivalence, maneuver completion guarantees) while producing counterexamples only in scenarios consistent with SOP-allowed violations. This step thus closes the loop between model synthesis and verification: syntactic correctness ensures executability, and counterexample refinement ensures that the surviving traces are faithful to the domain specification.

Listing 9: Counterexample trace (excerpt)

```
--BEFORE
SPEC AG (security_verified -> !tamper_detection_clear) is false

--REFINEMENT FIX
INVAR !(security_verified & tamper_detection_clear)

ASSIGN
  next(tamper_detection_clear) := case
    security_verified : FALSE; -- gate off when verified
    TRUE              : tamper_detection_clear;
  esac;
```

## 3 Extended Benchmark Evaluation Results

Following are the Evaluation Results for the entire Mini SpecBench. The \* in the execution compliance columns indicate that the `.smv` files for that model didn’t execute.

Table 2: LLM-Judge evaluation results across benchmark systems. Each cell shows a metric or compliance score, with models grouped by task.

Model	Structural Alignment	Property Fidelity	Semantic Fidelity	Code Bonus	Code Compliance	Execution Compliance	Final Compliance Score
<b>Shuttle Autopilot Guidance System</b>							
SpecMAS	4.84	5.17	5.66	0.19	0.45 ± 0.10	0.97 ± 0.05	0.76 ± 0.01
Claude	5.66	6.50	6.00	0.25	0.53 ± 0.08	0.34 ± 0.12	0.42 ± 0.10
DeepSeek	4.33	4.33	5.00	0.03	0.37 ± 0.07	0.45 ± 0.17	0.42 ± 0.07
Gemini	4.83	5.66	5.00	-0.23	0.36 ± 0.04	0.60 ± 0.00	0.51 ± 0.01
Qwen	5.16	5.16	5.33	0.20	0.45 ± 0.04	0.04 ± 0.05	0.20 ± 0.04
<b>PCI Bus Protocol</b>							

Continued on next page

Table 2 – continued from previous page

Model	Structural Alignment	Property Fidelity	Semantic Fidelity	Code Bonus	Code Compliance	Execution Compliance	Final Compliance Score
SpecMAS	5.50	5.50	6.50	0.64	0.59 $\pm$ 0.08	0.90 $\pm$ 0.14	0.78 $\pm$ 0.05
Claude	7.00	7.84	7.16	0.36	0.66 $\pm$ 0.01	0.14 $\pm$ 0.09	0.34 $\pm$ 0.05
DeepSeek	4.33	4.83	5.84	0.04	0.41 $\pm$ 0.00	0.66 $\pm$ 0.09	0.56 $\pm$ 0.06
Gemini	6.34	7.16	7.00	0.12	0.57 $\pm$ 0.10	0.80 $\pm$ 0.14	0.71 $\pm$ 0.13
Qwen	5.00	5.16	4.67	0.66	0.53 $\pm$ 0.01	0.50 $\pm$ 0.00	0.52 $\pm$ 0.01
<b>Robot Controller</b>							
SpecMAS	4.00	4.50	3.50	0.28	0.38 $\pm$ 0.06	0.90 $\pm$ 0.07	0.69 $\pm$ 0.07
Claude	6.83	8.34	8.00	0.73	0.76 $\pm$ 0.04	0.00 $\pm$ 0.00	0.30 $\pm$ 0.02
DeepSeek	5.16	6.16	5.50	-0.06	0.44 $\pm$ 0.04	0.64 $\pm$ 0.06	0.56 $\pm$ 0.02
Gemini	8.84	9.34	8.34	0.22	0.75 $\pm$ 0.03	0.00 $\pm$ 0.00	0.30 $\pm$ 0.01
Qwen	6.66	7.17	5.66	0.34	0.58 $\pm$ 0.05	0.00 $\pm$ 0.00	0.24 $\pm$ 0.02
<b>Priority Queue Buffer</b>							
SpecMAS	6.50	7.50	6.50	0.25	0.60 $\pm$ 0.12	0.97 $\pm$ 0.05	0.82 $\pm$ 0.02
Claude	8.16	8.67	7.34	0.24	0.70 $\pm$ 0.06	0.88 $\pm$ 0.11	0.81 $\pm$ 0.04
DeepSeek	6.34	6.84	6.16	-0.09	0.50 $\pm$ 0.07	0.92 $\pm$ 0.06	0.75 $\pm$ 0.01
Gemini	4.66	6.84	5.66	0.52	0.56 $\pm$ 0.01	0.84 $\pm$ 0.05	0.72 $\pm$ 0.02
Qwen	5.50	6.16	6.00	-0.06	0.45 $\pm$ 0.06	0.16 $\pm$ 0.06	0.28 $\pm$ 0.06
<b>Traffic Collision Avoidance System</b>							
SpecMAS	3.84	5.34	5.17	0.35	0.45 $\pm$ 0.10	0.86 $\pm$ 0.01	0.70 $\pm$ 0.04
Claude	5.50	6.00	6.67	0.64	0.62 $\pm$ 0.02	0.00 $\pm$ 0.00	0.24 $\pm$ 0.01
DeepSeek	3.50	3.84	3.50	-0.12	0.26 $\pm$ 0.04	0.80 $\pm$ 0.09	0.58 $\pm$ 0.04
Gemini	4.66	7.16	5.50	0.62	0.59 $\pm$ 0.07	0.00 $\pm$ 0.00	0.24 $\pm$ 0.03
Qwen	4.66	5.50	5.16	0.11	0.42 $\pm$ 0.06	0.00 $\pm$ 0.00	0.17 $\pm$ 0.03
<b>Ring Oscillator</b>							
SpecMAS	5.83	8.17	5.00	0.14	0.54 $\pm$ 0.06	0.94 $\pm$ 0.08	0.78 $\pm$ 0.03
Claude	6.33	7.50	6.67	0.22	0.59 $\pm$ 0.10	0.00 $\pm$ 0.00	0.24 $\pm$ 0.04
DeepSeek	6.34	8.34	8.16	0.34	0.68 $\pm$ 0.05	0.77 $\pm$ 0.08	0.74 $\pm$ 0.06
Gemini	8.00	9.84	8.50	0.14	0.73 $\pm$ 0.10	0.97 $\pm$ 0.05	0.87 $\pm$ 0.01
Qwen	6.00	8.00	6.33	0.25	0.59 $\pm$ 0.16	1.00 $\pm$ 0.00	0.84 $\pm$ 0.06
<b>Mutual Exclusion Protocol for Two-Process Coordination</b>							
SpecMAS	8.50	7.84	7.00	0.19	0.66 $\pm$ 0.09	0.85 $\pm$ 0.07	0.77 $\pm$ 0.08
Claude	9.66	9.50	9.34	0.02	0.76 $\pm$ 0.13	0.95 $\pm$ 0.07	0.88 $\pm$ 0.09
DeepSeek	8.50	9.16	7.16	0.03	0.66 $\pm$ 0.13	1.00 $\pm$ 0.00	0.86 $\pm$ 0.05
Gemini	8.00	9.84	6.50	0.38	0.73 $\pm$ 0.08	0.85 $\pm$ 0.07	0.80 $\pm$ 0.08
Qwen	9.66	10.00	9.50	0.06	0.79 $\pm$ 0.06	0.10 $\pm$ 0.00	0.38 $\pm$ 0.02
<b>Two Process Semaphore</b>							
SpecMAS	5.84	6.17	5.00	0.34	0.52 $\pm$ 0.03	0.90 $\pm$ 0.14	0.75 $\pm$ 0.10
Claude	6.17	8.50	8.00	-0.03	0.60 $\pm$ 0.06	0.77 $\pm$ 0.04	0.70 $\pm$ 0.00
DeepSeek	7.16	8.83	6.50	0.04	0.60 $\pm$ 0.02	0.26 $\pm$ 0.05	0.40 $\pm$ 0.02
Gemini	9.84	10.00	9.50	0.53	0.89 $\pm$ 0.08	0.08 $\pm$ 0.11	0.40 $\pm$ 0.10
Qwen	8.66	9.00	8.83	0.44	0.79 $\pm$ 0.11	0.87 $\pm$ 0.03	0.84 $\pm$ 0.06
<b>Gigamax Cache Coherence Protocol</b>							
SpecMAS	5.00	5.67	4.67	0.46	0.50 $\pm$ 0.04	0.84 $\pm$ 0.06	0.70 $\pm$ 0.05
Claude	6.50	8.84	6.84	0.27	0.64 $\pm$ 0.01	0.06 $\pm$ 0.04	0.29 $\pm$ 0.01
DeepSeek	6.16	7.16	6.50	0.14	0.56 $\pm$ 0.01	0.66 $\pm$ 0.05	0.62 $\pm$ 0.02
Gemini	8.50	9.34	7.66	0.40	0.76 $\pm$ 0.01	0.00 $\pm$ 0.00	0.30 $\pm$ 0.01
Qwen	6.50	7.16	6.84	-0.08	0.53 $\pm$ 0.04	0.61 $\pm$ 0.01	0.57 $\pm$ 0.01
<b>BRP Protocol</b>							
SpecMAS	5.66	6.84	5.50	0.38	0.55 $\pm$ 0.02	0.90 $\pm$ 0.14	0.76 $\pm$ 0.08
Claude	6.66	8.16	6.84	0.10	0.60 $\pm$ 0.08	0.38 $\pm$ 0.04	0.46 $\pm$ 0.05
DeepSeek	5.00	5.16	5.34	0.20	0.45 $\pm$ 0.01	0.50 $\pm$ 0.14	0.48 $\pm$ 0.08

Continued on next page



**Table 2 – continued from previous page**

<b>Model</b>	Structural Alignment	Property Fidelity	Semantic Fidelity	Code Bonus	Code Compliance	Execution Compliance	Final Compliance Score
Gemini	7.34	8.84	8.17	0.52	$0.75 \pm 0.04$	$0.00 \pm 0.00$	$0.30 \pm 0.01$
Qwen	6.67	8.00	7.17	-0.14	$0.55 \pm 0.04$	$0.16 \pm 0.04$	$0.32 \pm 0.04$

### 3.1 Evaluation Results

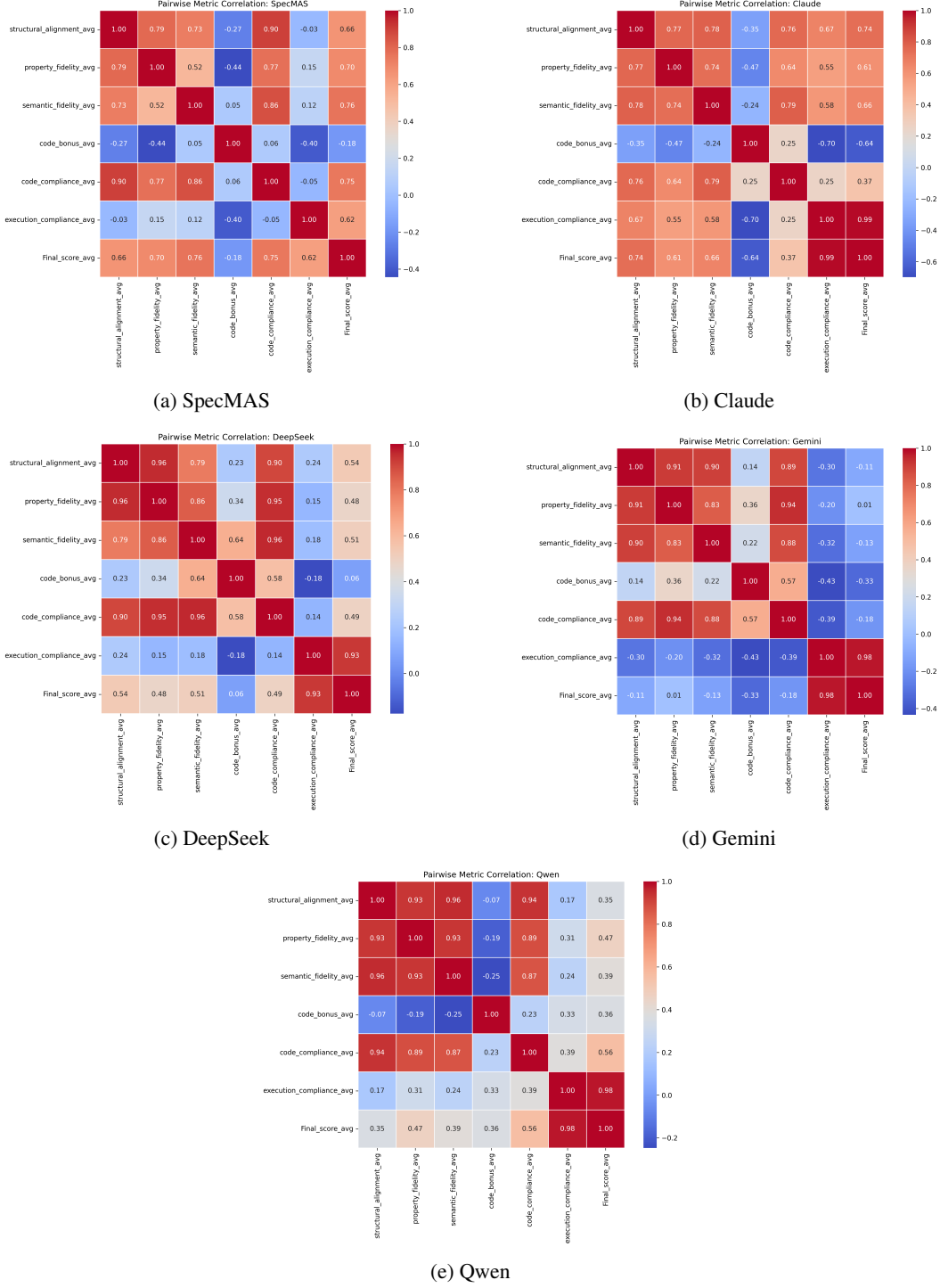


Figure 1: Pairwise correlation of evaluation metrics per model. These heatmaps reveal how different metrics relate to each other within each LLM across all SOPs.

In Figure 1, we present per-model correlation heatmaps that visualize the pairwise Pearson correlations between the key evaluation metrics used in our benchmark: structural alignment, property fidelity, semantic fidelity, code bonus, code compliance, execution compliance and final composite score. Each heatmap

reflects metric relationships aggregated across SOPs evaluated by the corresponding model, reveals how different evaluations signals interact within a model’s operational behaviour.

### 3.1.1 Insights

Specification metrics are internally consistent- across the models. All the three major metrics show strong inter-correlations in nearly every model ( $r > 0.8$ ), indicating that models either ”get the spec right” holistically or fail consistently across all the SPEC validates This validates our design decision to treat these dimensions as separate but yet aligned indicators of Specification quality.

Another interesting observation is that code compliance is model-dependent- some models align with expert specs, others don’t. Claude and Qwen show a positive correlation for code compliance with other ( $r \approx 0.7 - 0.9$ ), suggesting that their .smv outputs are grounded in the logical structure of the specification. In contrast, Baseline and DeepSeek display weak or even negative correlations, implying their code correctness is decoupled from Spec understanding.

## 4 Extended Ablation Results

Table 3: Extended ablation results showing average metric scores across benchmark SOPs. Standard deviations are reported for Code Compliance, Execution Compliance, and Final Score.

Model	Structural Alignment	Property Fidelity	Semantic Fidelity	Code Bonus	Code Compliance	Execution Compliance	Final Compliance Score
<b>Shuttle Autopilot Guidance System</b>							
Agent	6.00	8.00	7.33	-0.33	0.50	0.86	0.72
Agent+ Debugger	6.00	6.33	6.67	0.50	0.60	1.00	<b>0.84</b>
Qwen	5.33	5.00	6.00	0.08	0.45	0.20	0.30
<b>PCI Bus Protocol</b>							
Agent	4.67	4.00	5.67	0.50	0.48	0.80	0.67
Agent+ Debugger	5.33	5.00	5.00	0.62	0.53	1.00	<b>0.81</b>
Qwen	5.67	5.33	5.33	0.00	0.43	0.13	0.25
<b>Robot Controller</b>							
Agent	3.33	4.00	2.33	0.08	0.27	0.56	0.44
Agent+ Debugger	2.67	4.67	3.00	0.17	0.31	1.00	<b>0.72</b>
Qwen	6.00	8.33	7.00	0.50	0.67	0.00	0.27
<b>Priority Queue Buffer</b>							
Agent	6.33	8.00	6.67	0.25	0.61	0.20	0.36
Agent+ Debugger	7.67	8.00	7.33	0.29	0.67	0.80	<b>0.75</b>
Qwen	5.33	3.67	4.00	0.00	0.34	0.90	0.68
<b>Traffic Collision Avoidance System</b>							
Agent	4.33	5.00	5.33	0.25	0.44	0.80	0.66
Agent+ Debugger	4.33	5.00	5.00	0.88	0.56	1.00	<b>0.82</b>
Qwen	3.33	3.67	3.67	0.25	0.34	0.00	0.14
<b>Ring Oscillator</b>							
Agent	5.67	8.00	5.33	0.50	0.60	0.00	0.24
Agent+ Debugger	4.00	6.67	5.00	0.49	0.51	0.80	<b>0.68</b>
Qwen	6.67	6.67	8.33	0.00	0.58	1.00	0.83
<b>Mutual Exclusion Protocol for Two-Process Coordination</b>							
Agent	8.00	8.00	7.00	0.08	0.63	0.00	0.25
Agent+ Debugger	8.33	5.67	6.67	0.00	0.55	0.90	0.76
Qwen	7.33	8.67	8.33	0.00	0.65	1.00	<b>0.86</b>
<b>Two Process Semaphore</b>							
Agent	5.00	6.33	4.33	0.25	0.47	0.00	0.19
Agent+ Debugger	6.00	7.00	6.00	0.46	0.60	1.00	<b>0.84</b>
Qwen	6.67	9.00	5.33	-0.12	0.53	0.20	0.33
<b>Gigamax Cache Coherence Protocol</b>							
Agent	5.00	4.67	4.33	0.50	0.48	0.30	0.37

Continued on next page

**Table 3 – continued from previous page**

<b>Model</b>	<b>Structural Alignment</b>	<b>Property Fidelity</b>	<b>Semantic Fidelity</b>	<b>Code Bonus</b>	<b>Code Compliance</b>	<b>Execution Compliance</b>	<b>Final Compliance Score</b>
Agent+ Debugger	4.00	6.33	5.67	0.44	0.51	0.80	<b>0.68</b>
Qwen	4.33	8.00	5.67	0.19	0.52	0.20	0.33
<b>BRP Protocol</b>							
Agent	6.33	8.00	6.67	0.43	0.65	0.14	0.34
Agent+ Debugger	7.33	8.67	7.00	-0.25	0.57	0.80	<b>0.71</b>
Qwen	6.33	6.33	6.67	0.33	0.58	0.00	0.23

## 4.1 Ablation Results

In Figure 2, we present a series of slope charts that visualize the trajectory of model performance across three key stages of our evaluation pipeline, Specification Quality -> Execution Compliance -> Final score. Each line represents a model’s performance on a specific SOP of the MiniSpecBench, highlighting how initial spec quality (fails to translate) into successful execution and final usefulness.

### 4.1.1 Notable Patterns

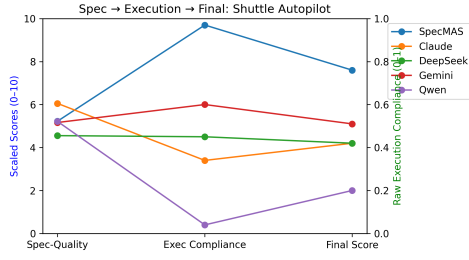
Across SOPs like Ring Oscillator and Priority Queue Buffer, we observe cases where multiple models like Gemini and Claude achieve a high average Spec quality but collapse at the execution stage. This emphasizes that even well-aligned logical structures fail to produce executable files under subtle syntax or semantic violations. So the Spec Quality alone is not sufficient to gauge the correctness of a generated specification. Furthermore, what’s interesting to observe is the steepest drops occur between the spec and execution stage - not between execution and final. This indicates that spec -> code translation is the major point of failure. Specifically in SOPs like GigaMax Cache Coherence and Traffic Collision Avoidance System, models like Qwen and DeepSeek show strong specs that are completely undercut by execution non-compliance, highlighting brittle code synthesis.

Claude and our System often exhibit flatter curves, indicating higher preservation of quality across the pipeline. For example, in priority queue buffer execution scores for all the models except Qwen align tightly with their final score, reflecting reliable generation. Across all the SOP runs, the Spec-MAS agent is by far the high-performing high performing system. There are some outliers, such as Mutual Exclusion and Ring Oscillator where our agent is either outperformed by other models or is at same performance level. The reason behind this could be that these two SOP topics are related to classical computer science problems, whose solutions are explored in details on the internet. Since these LLMs are trained on massive internet data, there’s a good possibility that these LLMs might be red data points, datapoints and thereby improving their score. One of the takeaways from the analysis would be to construct a better grouped benchmark in the future, that classifies tasks based on the “difficulty” of verification rather than domain buckets.

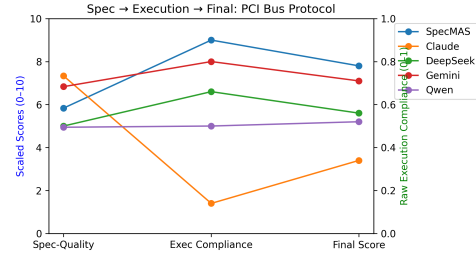
Figure 3. is an overall summary of our ablation study. We focused on three crucial aspects of our system:-

- Qwen 32B 8-bit quantized - Only the LLM backbone without Planning, tools, or execution debugger.
- Model & Specification Synthesis - Only planning and tools without execution debugger
- Model & Specification Synthesis + Debugger tool - Full planning + Tools + Debugger Agent

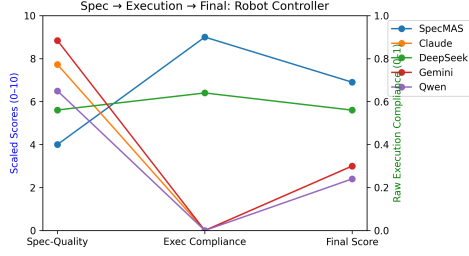
The Figure 3a decomposes the final score for each model-SOP pair into: 1) Code Compliance(40% weightage)- the syntactic and structural validity of generated SMV files. 2) Execution Compliance(60 %) - whether the generated model runs successfully on NuSMV with errors, counterexample traces or None.



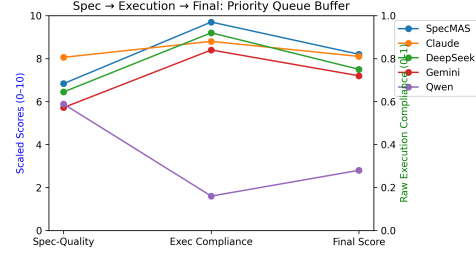
(a) Shuttle Autopilot Guidance



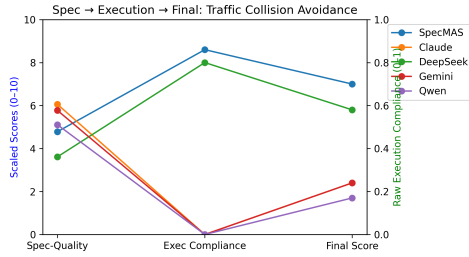
(b) PCI Bus protocol



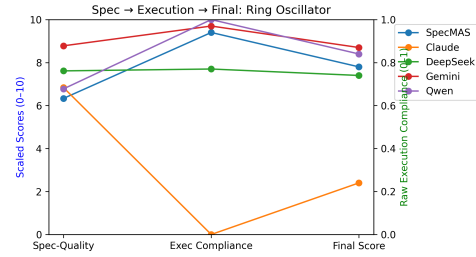
(c) Robot Controller



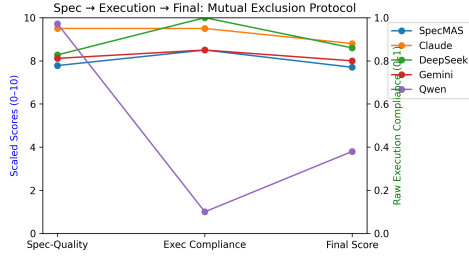
(d) Priority Queue Buffer



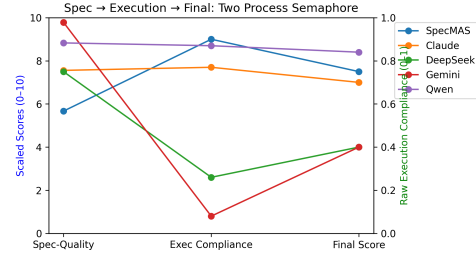
(e) Traffic Collision Avoidance System



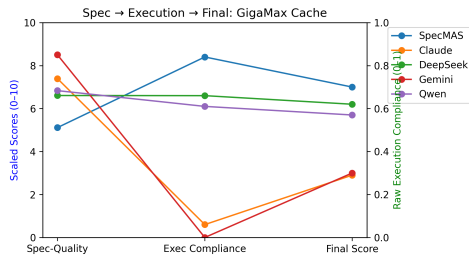
(f) Ring Oscillator



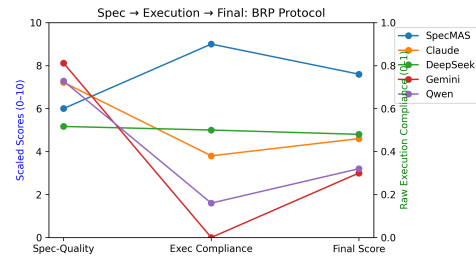
(g) Mutual Exclusion Protocol



(h) Two processes Semaphore



(i) GigaMax Cache Coherence



(j) BRP message protocol

Figure 2: Spec→Exec→Final slope graphs for each SOP benchmark. Models are evaluated across average Spec Quality, Execution Compliance, and Final Score. SpecMAS consistently achieves higher final scores with better execution reliability.

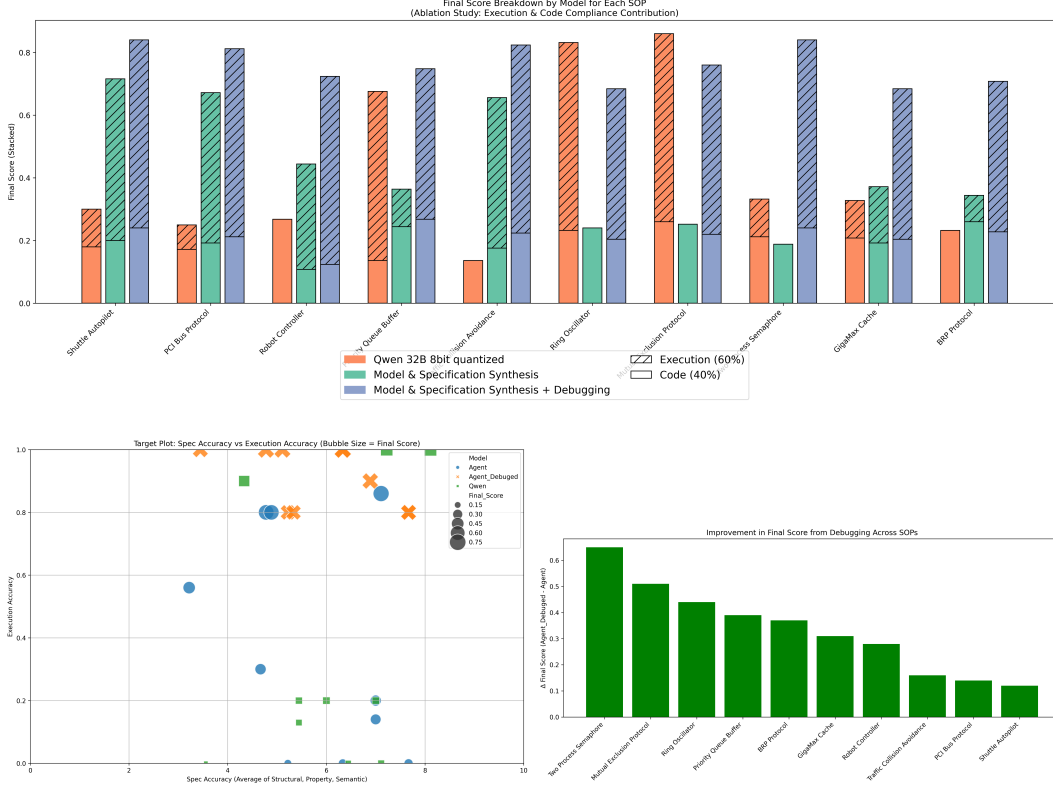


Figure 3: Caption

#### 4.1.2 Insights

Agent with the full debugger outperforms across nearly all SOPs, especially in execution heavy benchmark files like **Shuttle autopilot Guidance**, **Robot Controller**, **Gigamax Cache coherence**, indicating that it’s debugging pipeline actively resolves failure points. Furthermore, for most tasks Qwen’s performance is lower than others, except *Ring Oscillator* and *Mutual Exclusion*, which are the outliers, who’s possible reasons have been discussed previously. Implying that more sophisticated Agentic solutions hamper the quality of specification for simple tasks. Conversely, a few of the specifications generated by Qwen backbone LLM simply don’t execute, suggesting that for complex tasks, simply using off-the-shelf LLMs isn’t the best choice.

Figure 3b on the other hand, visualizes the relationship between specification fidelity(x-axis), execution accuracy (y-axis), and final score (bubble size). Each point represents a (model,SOP) pair.

#### 4.1.3 Analysis

Full Agent with Debugged is consistently in the top-right quadrant, indicating high spec and execution accuracy. This shows that introducing formal verification and a counterexample refinement-based debugger not only preserves spec quality but also improves runtime compliance. As expected, the Qwen LLM backbone clusters in the bottom-middle region, often producing reasonable specs that simply fail in execution. This affirms our analysis in the paper, that high spec quality doesn’t imply executability without alignment to verification constraints. As for the Agent without the debugger, it has scattered performance, with many points floating mid-left. It seems to be that it captures spec structure but suffers from inconsistent syntax and logical grounding and the counter-example refinement, leading to failures at the model-checking stage.

In summary the plot confirms our assumed hypotheses, that execution robustness is nontrivially dependent on post-processing. Without a counter-example-based debugging and knowledge tools, models often tend to hallucinate syntax and fail verification. Which is also confirmed by Figure 3c.

The tight clustering of Agent with debugger is high performing zone that proves debugging acts as an execution anchor for formality while preserving the specification.

## 5 Knowledge Base

For the purpose of providing context to our Agents we use a Self-Corrective Multi-Hop RAG(SCMRAG) [2]. The RAG vectorstore contains necessary information about Model Checking books, papers, and articles, which are sufficient to answer range of queries generated by any of our agents to obtain context. References to the documents are included in this file. Here's the names of the documents we used to create the knowledge vector store:-

Furthermore, we use a CRAG agent that acts as both a context tool and SOP information retriever. We've implemented a conditional tool called variable `sop_flag` that the Agent toggles depending on the requirements of the query. Prompt Templates for the CMRAG and CRAG Agent are shared in the code files with the names `crag_agent` and `knowledge_search_templates`

- Tutorial on Parameterized Model Checking of Fault-Tolerant Distributed Algorithms [8]
- Formal Techniques for Distributed Objects, Components, and Systems [1]
- Stochastic Model Checking [9]
- Statistical Model Checking: An Overview [10]
- Model-Checking - A Tutorial Introduction [11]
- Runtime Verification [4]
- NuSMV2.7.0 User Manual [5]
- NuSMV2.7Tutorial [6]

## 6 Implementation Details

### 6.1 Prompts

We provide a few specific prompts that are important and crucial to understanding the working of our system. Full detailed prompt files can found in the codebase folder.

#### 6.1.1 Agent Planner Prompt

**Agent Planner System Prompt**

You are an expert AI assistant trained in Finite State Modelling and graph formal methods. Your task is to develop a comprehensive plan to extract **Kripke Structures** from a given Standard Operating Procedure (SOP) text using a limited set of tools. You must reason creatively and rigorously to achieve a complete and verifiable formal model of the system described.

A **Kripke Structure** is defined as a tuple:

$$M = (S, I, R, L)$$

where:

- $S$  is a set of states,
- $I \subseteq S$  is the set of initial states,
- $R \subseteq S \times S$  is a transition relation (i.e., directed edges between states),
- $L : S \rightarrow 2^{AP}$  is a labeling function mapping each state to the set of atomic propositions true in that state.

Your plan must include the following:

1. **Understanding Core Requirements:** Analyze the structure of the SOP and the necessary elements for constructing a faithful Kripke model.

2. **Tool-Oriented Strategy:** Leverage the strengths of each tool in the provided toolset to maximize extraction accuracy, robustness, and completeness.
3. **Full Coverage Guarantee:** Design a process that attempts to extract *all possible states, transitions, and labels* implied by the SOP — including conditional flows, parallelism, loops, and termination criteria.

You are restricted to using only the tools listed below in your solution strategy:

`{tool_names}`

The description for each tool is as follows:

`{tools}`

**Objective:** Produce an actionable and modular plan that uses the available tools to generate a sound Kripke structure model from the SOP, suitable for downstream model checking and formal verification tasks.

### TASK Information Extraction Prompt

Please generate a plan to investigate and collect information about Kripke Structures. Write a two-step PLAN to describe how to solve the following query:

- What are Kripke Structures?

When writing the plan:

- Write the PLAN starting with the header 'PLAN:' followed by a numbered list of steps (e.g., 1. 2.). Do not add any unnecessary steps.
- The final step should always be: "Based on the above steps taken, please provide a conclusive answer for the given query."
- End the updated plan with the marker: <END\_OF\_PLAN> (on a new line, without a number).

### Kripke Extraction Planner Prompt

Generate a detailed plan to construct a Kripke Structure representation of a given system using NuSMV syntax. This plan must follow a structured set of steps involving document analysis, state extraction, and formal model generation.

**Your PLAN must address the following steps:**

1. Retrieve specific information and locate relevant sections within the document using a `crag_tool` call. Ensure the query is formulated as a single combined string to retrieve all relevant sections of the SOP document.
2. From the retrieved text, generate a comprehensive list of possible states in the system:
  - Extract all operational modes (distinct states the system can be in).
  - Extract all dynamic variables or quantities that change during execution.
  - Identify atomic propositions associated with each state or transition.
3. Verify that the output from Step 2 is coherent (i.e., properly typed, complete, and consistent). Then, concatenate the list of operational modes, dynamic variables, and atomic propositions into a single string `candidate_states`, and pass it to the `create_states` tool.
4. Save the NuSMV code returned by `create_states` into a file named `states.smv`.
5. Using the SOP information from Step 1 again, construct a list of possible state transitions:
  - Identify document phrases or triggers that initiate state transitions.



- Map all valid transitions between extracted states.
6. Verify that the list of transitions is coherent (typed, complete, and consistent).
  7. Construct a final input for the `create_transitions` tool by concatenating the extracted states and dynamic variables from Step 2 into a single string `states`, and the validated transitions into a string `candidate_triggers`. Pass both to `create_transitions`.
  8. Save the output from `create_transitions` into a file named `trans.smv`.
  9. Using the output SMV file, populate the `ASSIGN` section with appropriate `init()` values for dynamic variables and the `initial_state` based on Step 2.
  10. Save the fully populated NuSMV code into a final file named `kripke.smv`.
  11. Based on the above steps taken, please provide a conclusive answer for the given TASKS.

### 6.1.2 NuSMV Coding Agent Prompts

#### NuSMV Agent System Prompt

Please generate a plan to investigate and collect information about Kripke Structures. Write a two step PLAN to describe how to solve the following TASK:

- A detailed description on how to represent a Kripke Structure and the associated Properties for a system in NuSMV model checker?

When writing the plan:

- Write the PLAN starting with the header 'PLAN:' followed by a numbered list of steps (e.g., 1. 2.). Do not add any unnecessary steps.
- The final step should always be: "Based on the above steps taken, please provide a conclusive answer for the TASK"
- End the updated plan with the marker: `<END_OF_PLAN>` (on a new line, without a number).

#### NuSMV Planner Prompt

Your task is to devise a detailed, step-by-step PLAN that **merges** the given NuSMV code segments into a single, coherent, and runnable NuSMV file. The final model must load in NuSMV without syntax errors **and** remain semantically consistent (all variables referenced exactly match their declarations, no duplicate or missing sections).

Your PLAN must describe how to solve the following TASKS:

1. Use the `crag_tool` to gather any required information about the NuSMV model checker and the specifics of writing a NuSMV model file.
2. Assemble the full NuSMV model file using the contents of `states.smv`, `kripke.smv` and `properties.smv`. The `nusmv_codepad` tool can be used for this task.
3. Save the output from Step 2 to a file called `{model_name}`. The `save_nusmv` tool can be used to perform this task.

Ensure the plan is complete, logically ordered and specific about how states, transitions and properties can be extracted from the given documents to describe the system. The purpose of this plan is to guide the creation of a NuSMV model file that accurately describes the underlying system and its behaviours.

When writing the plan:

- Write the PLAN starting with the header 'PLAN:' followed by a numbered list of steps (e.g., 1. 2. 3.). Do not add any unnecessary steps.

- The final step should always be: "Based on the above steps taken, please provide a conclusive answer for the given TASKS."
- End the updated plan with the marker: <END\_OF\_PLAN> (on a new line, without a number).

#### **NuSMV Agent Executor Prompt**

Please perform the following tasks in order:

1. List all contents of the current working directory.
2. Read the NuSMV model file named {model\_name}.
3. Execute the file using the `execute_nusmv` tool.
4. If execution fails or if there are counterexample traces present for any specification:
  - Always use the `crag_tool` to first gather information about the current Standard Operating Procedures (SOP) and potential fixes to address the errors or counterexamples.
  - Use the `debug_nusmv` tool to revise the code and fix all errors or counterexample traces.
  - Overwrite the existing NuSMV model file with the updated code using the `save_nusmv` tool.
  - Re-execute the file using the `execute_nusmv` tool.
  - Repeat Step 4 until the program executes successfully without errors.
5. Once execution is successful, provide a conclusive answer based on the output.

## LLM-as-Judge Eval Prompt

Your task is to evaluate and judge the following AI Agent-generated NuSMV model file with the Expert-written (ground truth) NuSMV model for the given Standard Operating Procedure Document.

**Standard Operating Procedure Document:** {SOP.TXT}

**Expert SMV model file:** {expert\_smv}

**Agent generated SMV model file:** {agent\_smv}

### Instructions for the LLM Judge

- Always refer back to the SOP for intent: which variables, modes, transitions and properties *should* exist.
- When estimating scores for each criterion, explicitly reference the SOP to justify your scoring, especially for ambiguities or subjective judgments.
- Do not expect identical names or ordering.
- **Map** variables, modules, DEFINES and properties by their **intent** (e.g., “current operational mode,” “door flag,” “movement guard”), not by their literal identifiers.
- **Compare** the *intended behavior* and *structure* as derived from the SOP, tolerating synonyms or rephrasing.

### Evaluation Criteria:

#### 1) Structural Alignment

- **Role Coverage:** Up to 10 points based on coverage of critical system variables (state enums, counters, flags).
- **Transition Logic:** Up to 10 points based on alignment with state-machine transitions from SOP.
- **Module-Define Usage:** Up to 10 points based on whether the modular decomposition matches the expert model’s abstraction.
- **Exploration Count:** Count extra DEFINE, MODULE, or INVAR blocks present beyond the SOP.

#### 2) Property Fidelity

- **Coverage:** Up to 10 points for quantity of CTL/LTL properties mapped to SOP or expert.
- **Logical Equivalence:** Up to 10 points for correct temporal relationships and structure.
- **Operator Correctness:** Up to 10 points for proper use of AG, AF, AX, etc.
- **Relevance Count:** Count contextually relevant properties present in the agent file but absent in the expert file.

#### 3) Semantic Fidelity

- **Behavior Match:** Up to 10 points for whether the agent’s behavior matches execution semantics from SOP/expert.
- **Edge-Case Handling:** Up to 10 points for handling edge conditions (e.g., failover, fairness).
- **Naming Clarity:** Up to 10 points for intuitive, role-representative identifiers.
- **Penalty Count:** Count of hallucinated system behaviors unsupported by SOP or expert file.

#### 4) Conciseness

- **Additional Concepts:** Count extra variables, states, or transitions not in the expert model.
- **Redundant Modules:** Count unused or repeated code blocks.

- **Additional Properties:** Count surplus LTL/CTL specs that don't match the expert's model.

**5) Overall Score:** Rate the overall quality of the Agent model on a scale of 0–10.

**Your response must follow the JSON schema below:**

```
{
  "structural_alignment": {
    "score": {
      "role_coverage": 0,
      "transition_logic": 0,
      "module_define_usage": 0,
      "exploration_count": 0
    },
    "explanation": "Explanation of how structural intent
    matched the expert/SOP,
    including examples of renamed roles,
    similar transitions, or decomposed logic."
  },
  "property_fidelity": {
    "score": {
      "coverage": 0,
      "logical_equivalence": 0,
      "operator_correctness": 0,
      "relevance_count": 0
    },
    "explanation": "Explanation of how CTL/LTL properties
    compare semantically to expert/SOP,
    including notes on phrasing,
    abstraction, or missed properties."
  },
  "semantic_fidelity": {
    "score": {
      "behavior_match": 0,
      "edge_case_handling": 0,
      "naming_clarity": 0,
      "penalty_count": 0
    },
    "explanation": "Explain how closely the model behavior
    matches SOP/expert intent,
    with examples of correct
    or hallucinated semantics."
  },
  "conciseness": {
    "score": {
      "additional_concepts": 0,
      "redundant_modules": 0,
      "additional_properties": 0
    },
    "explanation": "List redundant elements,
    spurious roles, or additional specs.
    Note if comments helped clarify renamed or restructured logic."
  },
  "overall_score": 0,
  "summary": "Brief summary comparing abstract behavior, structure,
  and verification outcome to the expert reference and SOP."
}
```

### LLM-as-Judge Eval Prompt

Your task is to assess the Agent-generated SMV model file for **Verifiability & Correctness** by:

- Counting the number of instances of counterexample traces for SPEC violation in the CLI output of the Agent-generated SMV model file.
- Counting the number of instances of minor issues in the model code (e.g., missing init for a mapped concept, incorrect type range) in the Agent-generated SMV model file.

**Standard Operating Procedure Document:** {SOP.TXT}

**Expert SMV model file:** {expert\_smv}

**Agent generated SMV model file:** {agent\_smv}

**NuSMV CLI Output:** {cli\_output}

**Your response should exactly follow the following JSON schema (fill zeros and placeholder values):**

```
{
  "counts": {
    "counterexample_traces": 0,
    "minor_issues": 0
  },
  "explanation": "Describe any verification errors,
  type mismatches, or
  reasons why expert properties
  did/didn't verify in the generated model."
}
```

## 6.2 Expanded Metrics definitions

In this section, we talk about the precise definitions of the temporal logic properties and evaluation metrics used in this work, along with the source code (available in the supplementary ZIP), which specifies the full list of CTL/LTL properties, their logical structure, and the associated evaluation scripts used for automated verification and scoring.

### 6.2.1 Property Definitions

#### [A]. Structural Alignment

**Definition** Structural Alignment measures how closely the architecture of the agent-generated NuSMV model reflects the expert written model in terms of system components, variable roles, transition structure, and module organization. It focuses on what elements exist and how they interconnect, rather than on their exact names or order. This includes the presence of key variables (modes, flags, regions), the correctness of transition logic, and the modular decomposition (use of MODULE, DEFINE, and ASSIGN blocks).

**Significance** A model can only reproduce the intended behavior if its structural backbone mirrors that of the original system. High structural alignment ensures that the AI agent has captured the same state-space semantics and functional decomposition that the human expert encoded. It acts as the foundation for semantic and property-level fidelity; without structural integrity, even correct logical formulas could evaluate over the wrong domain.

$$\text{Structural Alignment} = \frac{\text{Role Coverage} + \text{Transition Logic} + \text{Module Define Usage}}{3} \quad (1)$$

#### [B]. Property Fidelity

**Definition** Property Fidelity quantifies how accurately the agent-generated model captures the formal properties (CTL/LTL specifications) that define desired system behavior-safety, liveness, reach-

ability, and fairness constraints. It evaluates whether the agent includes equivalent or logically consistent temporal formulas compared to those defined in the expert model and SOP.

**Significance** These temporal properties represent the requirements the system must satisfy. Property fidelity, therefore, tests whether the agent not only builds a syntactically correct model but also understands what that model is supposed to guarantee. High fidelity implies that the AI has correctly internalized the verification goals, ensuring that the resulting model can be used for automated checking, counterexample generation, and certification under the same standards as the expert-written reference.

$$\text{Property Fidelity} = \frac{\text{Coverage} + \text{Logical Equivalence} + \text{Operator Correctness}}{3} \quad (2)$$

### [C]. Semantic Fidelity

**Definition** Semantic Fidelity assesses whether the agent’s model behaves like the expert model under execution semantics—that is, whether its transitions, guards, and responses to conditions match the intent of the SOP. It examines alignment in dynamic behavior, treatment of edge cases, naming clarity, and the avoidance of spurious or non-SOP-derived system actions.

**significance** While structural alignment focuses on “form,” semantic fidelity measures “function.” Two models may look similar but behave differently; semantic fidelity ensures that, given the same inputs or conditions, both evolve in logically consistent ways. It captures the true depth of understanding: that the agent can not only reproduce code structure but also infer the correct operational semantics from natural language specifications.

$$\text{Semantic Fidelity} = \frac{\text{Behaviour Match} + \text{Edge Cases} + \text{Naming Clarity}}{3} \quad (3)$$

### Code Bonus

$$\text{Code Bonus} = \frac{(+E) + (+R) + (-P) + (+A_c) + (-R_m) + (+A_p)}{4} \quad (4)$$

where,

E = Exploration Count

R = Relevance Count

P = Penalty Count

$A_c$  = Additional Concepts

$R_m$  = Redundant Modules

$A_p$  = Additional Properties

(All count metrics are normalized. + terms denote the specific metrics are rewarding metrics i.e exploration, relevance etc. and - terms denote penalizing metrics i.e Redundant Properties, penalty etc.)

**Code Compliance** From equations (1), (2), (3) we can define M,

$$M \in \text{Structural, Property, Semantic with subscores } s_k \in [0, 10]$$

Then code compliance can be defined as,

$$\text{Code Compliance} = [0.8 \times \frac{M}{30} + 0.2 \times \text{Code Bonus}] \quad (5)$$

**Execution Compliance** From 6.1.2 judge evaluation, we get two additional metrics that are related to the successful execution of the NuSMV model. These are counterexample traces and minor issues in the model file. Similar to previous property definitions, we normalize these scores across the benchmark to get norm\_counterexample traces and norm\_minor issues. Then we can

define execution compliance as,

$$\mathbf{Execution\ Compliance} = [0.8 \times \text{norm\_counterexample\_traces} + 0.2 \times \text{norm\_minor\_issues}] \quad (6)$$

**Final Compliance Score** Finally, we can define Final compliance score as mentioned in our paper, this score is defined as:

$$\mathbf{Final\ Compliance} = \text{clip}_{[0,1]}(0.5 \times \text{Code Compliance} + 0.5 \times \text{Execution Compliance}) \quad (7)$$

## References

- [1] *Formal Techniques for Distributed Objects, Components, and Systems: 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15–19, 2020, Proceedings*, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-50085-6.
- [2] Rishabh Agrawal, Murtaza Asrani, Hadi Youssef, and Apurva Narayan. Scmrag: Self-corrective multihop retrieval augmented generation system for llm agents. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '25*, page 50–58, Richland, SC, 2025. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9798400714269.
- [3] Anthropic. The claude 3 model family: Opus, sonnet, haiku. March 2024. URL [https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model\\_Card\\_Claude\\_3.pdf](https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf). Published March 4, 2024; accessed May 16, 2025.
- [4] Ezio Bartocci and Rupak Majumdar. *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, volume 9333. 01 2015. ISBN 978-3-319-23819-7. doi: 10.1007/978-3-319-23820-3.
- [5] Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev. *NuSMV 2.6 User Manual*. Fondazione Bruno Kessler (FBK), 2014. URL <https://nusmv.fbk.eu/userman/v26/nusmv.pdf>. Available at <https://nusmv.fbk.eu/userman/v26/nusmv.pdf>.
- [6] Roberto Cavada, Alessandro Cimatti, Emanuele Olivetti, Marco Pistore, and Marco Roveri. *NuSMV 2.6 Tutorial*. Fondazione Bruno Kessler (FBK), 2014. URL <https://nusmv.fbk.eu/tutorial/v26/tutorial.pdf>. Available at <https://nusmv.fbk.eu/tutorial/v26/tutorial.pdf>.
- [7] DeepSeek-AI and Team. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. 2025. doi: <https://doi.org/10.48550/arXiv.2501.12948>. URL <https://arxiv.org/pdf/2501.12948>.
- [8] Annu Gmeiner, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Tutorial on parameterized model checking of fault-tolerant distributed algorithms. In *Advanced Lectures of the 14th International School on Formal Methods for Executable Software Models - Volume 8483*, page 122–171, Berlin, Heidelberg, 2014. Springer-Verlag. ISBN 9783319073163. doi: 10.1007/978-3-319-07317-0\_4. URL [https://doi.org/10.1007/978-3-319-07317-0\\_4](https://doi.org/10.1007/978-3-319-07317-0_4).
- [9] Marta Kwiatkowska, Gethin Norman, and David Parker. *Stochastic Model Checking*, pages 220–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72522-0. doi: 10.1007/978-3-540-72522-0\_6. URL [https://doi.org/10.1007/978-3-540-72522-0\\_6](https://doi.org/10.1007/978-3-540-72522-0_6).
- [10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Roşu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification*, pages 122–135, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16612-9.
- [11] Markus Müller-Olm, David Schmidt, and Bernhard Steffen. Model-checking: A tutorial introduction. volume 1694, pages 330–354, 01 1999.
- [12] Gemini Team. Gemini: A family of highly capable multimodal models. 2024. URL <https://arxiv.org/abs/2312.11805>.
- [13] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.



- [14] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jing Zhou, Jingren Zhou, Junyan Lin, Kai Dang, Keqin Bao, Ke-Pei Yang, Le Yu, Li-Chun Deng, Mei Li, Min Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. 2025. URL <https://api.semanticscholar.org/CorpusID:278602855>.