Large-Language-Model-Aided Intent Verification via Grounded Obligation Models

Surinder Sood, John Goodacre, and Lucas Cordeiro
The University of Manchester, Manchester, United Kingdom
surinder.sood@gmail.com, {john.goodacre,lucas.cordeiro}@manchester.ac.uk

***Abstract*—Engineering specifications remain largely prose, making operator intent difficult to verify and enforce. We present Intent Verification from Natural-Language Specifications to Grounded Kernel Enforcement (INVAR), which extracts feature catalogs and labelled transition systems (LTSs), decomposes natural-language intent into source-grounded obligations, and uses a formal model checker (NuSMV) to check evidence on a neutral Spec LTS and satisfaction on a restricted Intent LTS. Its grounded-obligation layer ties each temporal atom to source evidence and reports absent catalog, model, or runtime support as a named GAP rather than a vacuous success. From four hardware standards, INVAR extracts 8,173 features and 7,023 state machines. Audit-guided repair raises reference coverage from 55.3/65.2/42.4% to 97.4/87.0/100.0% for Transmission Control Protocol (TCP)/Universal Serial Bus (USB) 2.0/Compute Express Link (CXL) 4.0, respectively. The repaired TCP catalog achieves 91.7% strict and 95.5% lenient precision; normal-BM25 development-slice audits give 95.0% (97.5% lenient) for USB and 87.5% (100.0% lenient) for CXL, while their coverage values remain intent-scoped. Two intents exercise A3 decomposition in depth; a ten-intent, reference-grounded verification benchmark over five specifications grounds and verifies 30/34 atoms and reports the remaining four plus one clause-level coverage gap. A diagnostic comparison with nl2spec and NL2LTL shows that proposition grounding and parser coverage remain substantial limitations for the baseline outputs. Eight QEMU scenarios exercise USB, PCI Express (PCIe), and CXL actions; the expected verdicts are obtained after three runtime-model faults are repaired. The results establish traceable failure reporting for safety and reachability properties in heterogeneous platform specifications and an emulated Linux runtime.**

***Index Terms*—large-language-model-aided verification, obligation models, SoC security, runtime verification, labelled transition systems, NuSMV, linear temporal logic, specification mining, intent verification**

I. Introduction

Engineering systems are often governed by detailed specifications that define permitted behaviour, configuration constraints and state transitions. These specifications may describe hardware protocols, firmware interfaces, software application programming interfaces (APIs) or operational control planes. They are authoritative documents, but they are rarely available as executable models that can be checked against a user-defined security or safety intent.

Specifications are written for human engineers, with normative behaviour distributed across prose, tables and figures rather than exposed as formal transition systems. User intent is also informal and may combine permission, denial, reachability and temporal constraints. It must therefore be formalised without losing the concepts that make the intent meaningful. A further translation is then needed at runtime, where the observable events are implementation signals rather than specification clauses.

Existing work addresses fragments of this gap. Runtime verification frameworks [1] check system traces against formal properties but require properties to be hand-written. Large language model (LLM)-assisted specification mining [2], [3] extracts properties from natural language but does not produce verifiable formal models grounded to source documents. Extended Berkeley Packet Filter (eBPF)-based monitoring [4], [5] observes kernel events with no connection to specification semantics.

INVAR (Intent Verification from Natural-Language Specifications to Grounded Kernel Enforcement) connects these stages through a semi-automated tool flow. Given a specification document and a natural-language intent, it can:

1) extract a feature catalog and state machines from a specification Portable Document Format (PDF) file using a multi-pass LLM pipeline, with analyst review for catalog gaps; this paper evaluates extraction on protocol-specification PDFs;
2) materialise a neutral Spec LTS containing specification-permitted behaviour without intent filtering;
3) formalise intent as a compositional obligation verification model whose atomic predicates are grounded to specification features;
4) compile each grounded atom into paired Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) NuSMV queries: evidence and non-vacuity checks over the Spec LTS, and intent-satisfaction checks over the Intent LTS;
5) derive an Intent LTS and deploy it through a runtime observation adapter when probe bindings are available.

The main contribution is the grounded obligation pipeline: INVAR decomposes intent into traceable atoms, uses the neutral Spec LTS to establish their evidence and non-vacuity, uses the restricted Intent LTS to establish intent satisfaction, and reports catalog or runtime-observability gaps. This connects protocol and interconnect specifications to security-policy checking for heterogeneous system-on-chip and platform integration. Hardware protocol standards are the main evaluation domain, with TCP, standardised in Request for Comments (RFC) 9293, as one non-hardware data point;

portability beyond protocol specifications remains future work.

II. Background

A labelled transition system (LTS) is $M = (S, s_0, \Sigma, \rightarrow, AP, L)$: states S, initial state s_0, event alphabet Σ, transition relation $\rightarrow$, propositions AP, and valuation L. INVAR builds a neutral Spec LTS M_{spec} containing specification-permitted behaviour and an Intent LTS M_{intent} that retains allowed behaviour and attaches verified constraints.

Linear Temporal Logic (LTL) is interpreted universally over initial infinite paths; G, F, X and O mean globally, eventually, next and historically. Computation Tree Logic (CTL) $EF\,p$ asks whether some path eventually reaches p. NuSMV totalises dead ends by stuttering and we add no fairness constraints, so Section III-E transfers only finite-path reachability and invariant safety under restriction, not arbitrary LTL. Same-model LTL conjuncts compose by Boolean conjunction [6].

Runtime observation is adapter-independent: events must map to LTS labels and values. Our Linux prototype uses extended Berkeley Packet Filter (eBPF) tracepoints or kprobes and forwards events to a userspace verifier over TCP.

III. The INVAR Framework

Figure 1 shows the INVAR pipeline. Intent 1, defined and evaluated under O2 in Section VI-A, illustrates Figure 1: A1/A2 extract a catalog and neutral Spec LTS from USB HID, USB 2.0, PCIe and CXL PDFs; a separate RAG-assisted reference-audit branch retrieves source and catalog evidence for author-validated precision, recall and named-gap decisions; A3 decomposes the intent into USB HID allow atoms and CXL hot-add deny atoms; the LTSs and grounded atoms are compiled into NuSMV input, and the NuSMV symbolic model checker evaluates Spec-LTS evidence queries and Intent-LTS satisfaction queries; runtime events are then mapped to Intent LTS labels for `invar-verifier` JavaScript Object Notation (JSON) verdicts.

A. Specification Ingestion

The pipeline ingests natural-language specification documents through the A1/A2 extraction stage shown in Figure 1; the relevant A1 implementation subtasks are A1a and A1c. Pass A3, defined in Section III-E, decomposes natural-language intent into grounded obligations and an Intent LTS.

- **A1 feature extraction** has two subtasks: **A1a** extracts observable features such as protocol fields, register values, and state conditions, with evidence anchored to section numbers and pages; **A1c** resolves inter-feature relationships such as read-to-discover, write-to-control, and state-change, producing a grounded feature catalog.
- **A2 state-machine extraction** groups related features into per-feature state machines, identifying states and event-driven transitions.

The output is a `.invar.json` catalog with state machines. Four hardware standards yield 8,173 features and 7,023 machines; adding TCP gives 8,190 and 7,040. Pass B cross-checks the catalog, assigns confidence to missing, spurious or corrected candidates, and locally re-extracts high-confidence misses. Because this is a self-audit, Section VI-A0a separately freezes source-derived inventories, retrieves source and catalog evidence with retrieval-augmented generation (RAG) [7], and requires author adjudication. Retrieval supplies evidence, not truth or automatic repair.

B. Spec LTS Materialisation

From the `.invar.json` state machines, INVAR derives the Spec LTS M_{spec}, a neutral logical product comprising all extracted feature machines with no intent filtering. Every extracted feature, state and transition remains in this factored representation. For an individual obligation, the NuSMV encoding materialises only the referenced machines and their dependencies, avoiding full Cartesian enumeration while preserving the relevant product behaviour.

M_{spec} is saved as a separate artefact, `.spec.lts.json`, used exclusively at design time rather than deployed to the runtime verifier.

C. Framework Assumptions and Verdict Semantics

INVAR's claims are conditional on three stage-specific assumptions. *Specification processing* assumes that the extracted Spec LTS preserves the source behaviour relevant to an obligation. *Intent processing* assumes that decomposition captures the stated intent clauses and that each resulting atom is grounded to the intended catalog evidence. *Runtime observation* assumes that the adapter faithfully maps exercised system actions and values to the required Intent-LTS event labels and properties. These are scope conditions for interpreting the checks, not guarantees that the LLM-derived catalog or decomposition is complete.

Verdicts are therefore model- and evidence-relative. `PASS` means that a grounded obligation holds on the relevant generated model, or that an observed runtime trace conforms to the Intent LTS; it does not prove the source PDF or natural-language intent complete. `FAIL` denotes a model-checking counterexample or runtime violation. `GAP(catalog/model/runtime)` identifies evidence required for a claim but absent at the named pipeline stage. Aggregation is conservative: `FAIL` dominates `GAP`, which dominates `PASS`.

D. Model Coverage and Traceability

INVAR records two coverage levels. *Specification-model coverage* tracks extracted features represented in M_{spec} as states, transitions or labels; because the Spec LTS is built before intent filtering, uncovered concepts become catalog gaps rather than silently ignored facts. *Intent coverage* tracks decomposed atoms grounded to M_{spec} and compiled into per-atom CTL/LTL NuSMV properties evaluated against both LTS encodings; named GAP verdicts expose missing or over-specified atoms.

Traceability carries source provenance through features, state-machine transitions, obligation atoms, `DEFINE` clauses and NuSMV verdicts, giving a chain from an intent clause to

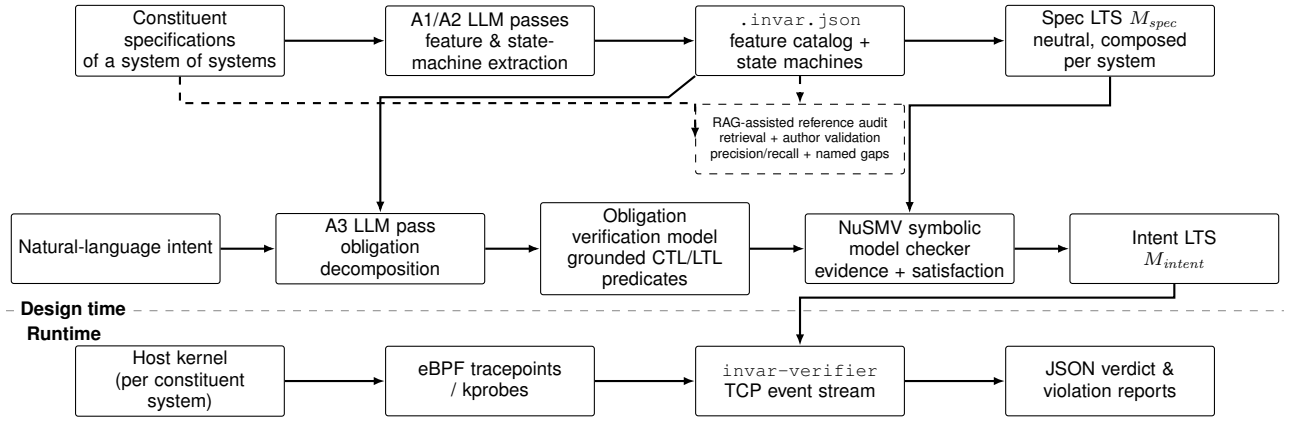


Fig. 1. INVAR design-time and runtime pipeline. The dashed branch supports an author-adjudicated extraction audit; it does not alter the catalog. NuSMV checks Spec-LTS evidence and Intent-LTS satisfaction. Same-model LTL obligations form Φ; CTL queries remain separate.

TABLE I
BACK-REFERENCE AUDIT FINDINGS AND FIXES.

Finding	Risk during grounding	Fix in INVAR
Broad provenance	A section citation can hide the exact evidence used.	Store the feature, transition, register bit or table row with each atom and verdict.
Cross-specification vocabulary	Similar USB, PCIe and CXL events can have different semantics.	Normalise terms only at the intent layer; preserve domain-specific atoms and provenance.
Catalog/model coverage	Evidence may be absent or an extracted feature may remain unclassified.	Emit `GAP(catalog)` for absent evidence and `GAP(model)` for evidence omitted from the obligation model.
Runtime observability	A grounded concept may lack a valid kernel binding.	Emit `GAP(runtime)` and require a binding grounded by measurement before claiming enforcement.

CTL/LTL, Spec LTS variables, source sections and runtime bindings.

Back-reference audit. During intent grounding, INVAR checks each atom against its specification evidence. Table I summarises the resulting findings, risks and fixes; this atom-level traceability check is distinct from the catalog precision/recall audit in Section VI-A0a.

Intent 1 illustrates all four categories: its eight atoms retain exact USB, PCIe and CXL evidence while preserving domain terminology. All eight have catalog support, but seven additional extracted CXL hot-add flows remain unclassified, producing `GAP(model)` rather than `GAP(catalog)`. Its generated USB/CXL binding names are unavailable on the evaluated kernel, so O3 uses measured mappings before making runtime claims.

E. Natural-Language Intent Processing

The user provides a natural-language intent statement. Pass A3 classifies each catalog feature as allow, deny, or do-not-care, asks for tracepoint or kprobe bindings, and produces the Intent LTS M_{intent}. The A3 prompt preserves feature, state and transition identifiers byte-for-byte while adding event bindings, so runtime artefacts remain traceable. If a named kernel event is synthetic or unavailable, the loader reports the missing attach point rather than enforcing an ungrounded rule.

INVAR validates M_{intent} as a restriction of M_{spec} before deployment. The check requires the initial state to be preserved, $S_{intent} \subseteq S_{spec}$, $\rightarrow_{intent} \subseteq \rightarrow_{spec}$ under the preserved transition identifiers, and $L_{intent}(s) = L_{spec}(s)$ for every retained state s. Denying a feature removes its transitions; allowing it keeps them unchanged.

Lemma 1 (Restriction). If M_{intent} is a label-preserving restriction of M_{spec}, every finite labelled path of M_{intent} is a finite labelled path of M_{spec}. Hence (i) an invariant state-safety property $G\,p$ that holds on M_{spec} also holds on M_{intent}; and (ii) an existential reachability property $EF\,p$ that holds on M_{intent} also holds on M_{spec}. Neither converse holds, and arbitrary LTL transfer is not claimed when restriction introduces a deadlock.

Proof. The initial state is shared, each retained step is a step of M_{spec}, and retained states have the same valuation. Induction on path length gives finite-path inclusion. Every state reachable in M_{intent} is therefore reachable in M_{spec}, proving (i); any finite witness for $EF\,p$ in M_{intent} is also a witness in M_{spec}, proving (ii). □

Accordingly, every grounded atom is evaluated on both encodings, but the two queries answer different questions. For an *allow* atom, $M_{spec} \models EF\,e$ establishes specification evidence and $M_{intent} \models EF\,e$ establishes that filtering preserves the permitted behaviour. For a *deny* atom, $M_{spec} \models EF\,e$ establishes observability and non-vacuity, while $M_{intent} \models G\,\neg e$ establishes exclusion. For a property-at-event atom, $M_{spec} \models EF\,e$ establishes that its trigger is observable, while $M_{intent} \models G(e \rightarrow p)$ establishes satisfaction; where removal of a violating alternative is claimed, $EF(e \wedge \neg p)$ is also checked on M_{spec}. NuSMV denotes the former existential reachability properties with CTL `SPEC` and the universal safety properties with `LTLSPEC`. Liveness obligations are not claimed to carry automatically under restriction. A template such as $G(B \rightarrow F(A))$ is admitted only with an explicit

progress or fairness assumption, and runtime checking compiles it to bounded liveness, e.g. $G(B \rightarrow F_{\leq T}(A))$, so the verifier can raise a violation when the deadline or event-count bound expires.

IV. Compositional Intent Verification

A. Obligation Verification Model

INVAR first constructs an obligation verification model: grounded atomic predicates, their provenance, and the evidence and satisfaction queries required for their obligation class. It then compiles this model together with the Spec or Intent LTS encoding into `.smv` input for the NuSMV symbolic model checker. Thus NuSMV is the verification engine and input language, not the obligation model itself.

We call an *atomic obligation*, or *atom*, a minimal independently checkable temporal obligation obtained by decomposing the user intent. Each atom is grounded to one or more features of the Spec LTS and is emitted as a NuSMV `DEFINE` clause together with the corresponding CTL `SPEC` and/or LTL `LTLSPEC` queries. We use *atom* in this obligation-level sense, not merely as an LTL atomic proposition.

- **DEFINE clauses.** Each clause encodes one atom consisting of one state variable, one comparison operator, one constant value and one specification section.
- **Per-atom verification queries.** Each grounded atom produces paired queries: a CTL reachability/observability query over M_{spec}, and the class-specific CTL or LTL satisfaction query over M_{intent}.
- **Composed LTL satisfaction formula Φ.** For LTL atomic obligations $\varphi_1, \ldots, \varphi_n$ checked on the same model, $\Phi = \bigwedge_{i=1}^{n} \varphi_i$. CTL evidence and allow-reachability queries remain separate, so Φ is only the LTL component.

Atomic obligations may therefore be CTL or LTL. Let E_k and S_k be atom k's evidence and satisfaction queries. CTL and LTL results combine at the verdict level as

$$\mathcal{V}(I) = \bigwedge_{k=1}^{m} \left([M_{spec} \models E_k] \wedge [M_{intent} \models S_k] \right). \qquad (1)$$

where S_k follows the obligation class. Intent 1 instantiates the query pairs as

$$(E_k, S_k) = \begin{cases} (EF\, a_k, EF\, a_k), & k = 1, \ldots, 5, \\ (EF\, b_j, G\, \neg b_j), & k = 5 + j,\ j = 1, \ldots, 3. \end{cases} \qquad (2)$$

Thus the five allow atoms are CTL, while the three LTL deny atoms compose as $\Phi_B = G\, \neg(b_1 \vee b_2 \vee b_3)$. The $EF\, b_j$ terms establish Spec-LTS evidence/non-vacuity and are not conjuncts of Φ_B. Equations 1–2 aggregate verdicts; they are not one mixed-logic NuSMV formula.

B. Atomic Decomposition via Canonical Templates

Adapting contract-based decomposition principles [8], INVAR decomposes a composite intent into independently checkable, source-grounded obligations. Canonical templates cover reachability $EF(e)$, property-at-event $G(e \rightarrow p)$, exclusion $G\neg e$, precedence $G(B \rightarrow O(A))$ and response $G(B \rightarrow F(A))$. A clause referencing independent properties is split before generation and annotated with its source section. The evaluation uses reachability and safety; liveness additionally requires design-time progress/fairness and a runtime bound.

C. LTL Composition Soundness

Observation 1 (LTL Composition Soundness). Let $\varphi_1, \ldots, \varphi_n$ be LTL formulae each valid over a model M. Then $\Phi = \varphi_1 \wedge \cdots \wedge \varphi_n$ is valid over M.

Proof. Immediate from the conjunction-preservation property of LTL [6] by induction on n. □

Corollary. After NuSMV confirms every LTL satisfaction atom on the same model, Φ is valid there without another run. This standard rule is not claimed as novel: INVAR contributes traceable generation, paired evidence/satisfaction checks and named gaps. Lemma 1 is used separately for reachability and invariant safety. Local atoms may use preserved subsystem projections; cross-subsystem atoms use the referenced product. All `PASS` verdicts remain model-relative.

D. Gap Detection

When an atom references a concept absent from the Spec LTS, the generator reports a named gap instead of emitting a vacuous DEFINE.

> GAP: `ob_network_class`; the PCIe Class Code register (network class `0x02`) is absent from the catalog. Verification requires a catalog extension.

Gap detection follows directly from the grounding requirement: every atom must cite a catalog feature and source section. Non-vacuity is atom-level: implication atoms need reachable antecedents, and deny atoms need observable denied events; otherwise INVAR reports vacuity/gap, not PASS.

V. Implementation

The Rust/TypeScript implementation generates both LTSs, grounded `DEFINE` clauses and paired CTL/LTL queries; invokes NuSMV and maps counterexamples to source features; and runs `invar-verifier`, which consumes packed eBPF events, walks M_{intent} and emits JSON verdicts. The loader reports unavailable probes instead of silently weakening a rule.

VI. Evaluation

We evaluate five archived protocol specifications: USB HID 1.11 [9], USB 2.0 [10], PCIe 4.0 v0.3 [11], CXL 4.0 [12] and TCP RFC 9293 [13]. NuSMV 2.6.0 [14] checks evidence on M_{spec} and satisfaction on M_{intent}. Runtime tests use an Ubuntu 24.04 QEMU 8.2 q35 guest (kernel 6.8.0-136-generic). Nine non-composite intents are also compared with nl2spec [2] and NL2LTL [3]. We ask whether INVAR (O1) extracts a traceable model substrate, (O2) checks composed obligations without hiding gaps and (O3) produces expected runtime verdicts while exposing observability gaps. We make no performance claim.

A. Results

a) O1: Extraction and grounding.: Across the four hardware standards, INVAR extracts 8,173 features and 7,023 state machines. Pass B reports 927 missing and 173 spurious candidates, but these are self-audit proxies rather than source-grounded accuracy.

RAG-assisted source-grounded audit. Before consulting Pass B, we freeze a rubric, a seeded feature sample and a specification-derived recall inventory. Label-free BM25 ranking [15] preserves the closest immutable source and catalog evidence for author adjudication. HID covers all 59 reference entries, but only 24/40 sampled features are correct (60%; 95% Wilson interval 45–74% [16]): nine are non-features, five duplicates and two misdescriptions. TCP's baseline 17 features are correct, but represent only 21/38 normative Section 3 subsections (55.3%); eight misses are Section 3.10 event-processing rules. Pass B reported 100% for both, showing why its values are proxies. Table II summarises the Pass-B and source-grounded results.

The repaired TCP experiment makes retrieval part of extraction and repair: structured parsing and retrieval feed grounded extraction, a normative/testability gate, Pass-B omission repair, exact-clause rewriting, semantic merging and a frozen-inventory coverage regression. The resulting 132-feature catalog covers 37/38 entries. Its complete evidence census labels 121 as correct, five partial, five non-features and one duplicate. Thus, the repaired pipeline raises TCP reference recall from 55.3% to 97.4% while achieving 91.7% strict and 95.5% lenient precision.

Applying audit-guided repair to frozen, intent-scoped inventories raises USB 2.0 target coverage from 15/23 (65.2%) to 20/23 (87.0%) and CXL 4.0 from 14/33 (42.4%) to 33/33 (100.0%). Normal-BM25 audits of 40-feature development samples from substantive USB Chapter 8–9 and CXL pages 75–110 slices give strict (lenient) precision of 95.0% (97.5%) and 87.5% (100.0%), respectively. Because the coverage inventories guided repair and the precision samples cover slices, neither measurement is an independent whole-specification estimate.

b) O2: Obligation verification. Intent 1, cross-specification.: **Intent.** "Only permit hotplug of USB HID devices on USB host controllers reachable via CXL/PCIe fabric; deny any hot add of any new CXL endpoints at runtime."

Specifications. USB HID 1.11, USB 2.0, PCIe 4.0 v0.3 draft, and CXL 4.0. All section references below were checked against these exact archived editions; the PCIe artifacts were extracted from the 2014 Version 0.3 draft and are not presented as results for a later final release.

The intent decomposes into two clauses across four specification domains. Clause A allows USB HID hotplug via the CXL/PCIe fabric, while Clause B denies CXL endpoint hot-add at runtime.

Obligation decomposition. INVAR first separates the intent into an allow clause and a deny clause, then decomposes each clause into independently checkable atoms, each grounded in a state or transition of an extracted feature state machine:

$$\begin{aligned} A_i &= EF\, a_i \quad (i = 1, \ldots, 5), \\ \Phi_B &= G\, \neg(b_1 \vee b_2 \vee b_3), \\ \Phi_B^{seq} &= G(attach \rightarrow G\, \neg(b_1 \vee b_2 \vee b_3)). \end{aligned} \tag{3}$$

Here $a_1, \ldots, a_5$ are the target states of five allow atoms spanning USB HID, USB 2.0, PCIe and CXL, and $b_1, \ldots, b_3$ are the CXL transition labels for hot-add, binding and runtime reconfiguration. Clause A permits USB HID hotplug, so it is an allow clause: its atoms, and $EF\, attach$ for the attach transitions themselves, require the permitted behaviour to stay reachable after filtering. Which USB host controllers are reachable via the fabric is a fact about the deployed topology, which no single specification states; we take the controllers in scope to be those behind the CXL/PCIe fabric as a scope assumption rather than a checked property. Φ_B denies the CXL events at every point at runtime; it implies the sequenced reading Φ_B^{seq}, that once a USB HID device has attached no new CXL endpoint can be added.

Models. For Intent 1, A3 classifies 15 catalog features—8 allow and 7 deny—to construct M_{intent}. M_{spec} is the interleaving product of their extracted state machines (68 local states, 76 transitions; 1.07×10^{10} reachable states). M_{intent} disables every transition of a denied feature (80,256 reachable states), so it restricts M_{spec} by construction. Both are checked with NuSMV 2.6.0.

Table III summarises the clause-level decomposition and verdicts; Table IV reports each grounded atom, source reference and model-checking result.

Every allow atom, and the USB attach itself, stays reachable on M_{intent}, so the restriction has not removed the behaviour Clause A permits. Each denied event is reachable on M_{spec}, so $G\, \neg b_i$ *fails* there, as it must: the deny atoms name behaviour

TABLE II
BASELINE AND REPAIRED SOURCE-GROUNDED RESULTS, WITH PASS-B PROXIES. USB/CXL REPAIRED PRECISION USES NORMAL-BM25 DEVELOPMENT-SLICE SAMPLES; RECALL IS INTENT-SCOPED TARGET COVERAGE. VALUES ARE P/R PERCENTAGES AND ARE NOT JOINT WHOLE-SPECIFICATION ESTIMATES; N/R MEANS PASS B WAS NOT RERUN AFTER REPAIR.

Spec.	Precision inventory	Recall inventory	Pass B P/R	Reference P/R
HID	40/361 features	59 normative entries	100/100	60 (45–74)/100
TCP baseline	17/17 features	38 entries; 21 represented	100/100	100/55.3
TCP repaired	132-feature census	38 entries; 37 represented	n/r	91.7 strict 95.5 len./97.4
USB baseline	40/1,293 feature sample	23 targets; 15 represented	97.7/88.4	47.5 strict/65.2
USB repaired	40/66 slice sample	23 targets; 20 represented	n/r	95.0 strict 97.5 len./87.0
CXL baseline	40/3,503 feature sample	33 targets; 14 represented	98.1/97.0	62.5 strict/42.4
CXL repaired	40/44 slice sample	33 targets; 33 represented	n/r	87.5 strict 100.0 len./100.0

TABLE III
INTENT 1 OBLIGATION DECOMPOSITION.

Clause	Sub-intent and obligation	Atoms	Verdict
A	Permit USB HID hotplug through a reachable USB host over CXL/PCIe: $A_1, \ldots, A_5$ and $EF\ attach$ on M_{intent}; fabric reachability is a scope assumption	HID class; hub detect; USB host; PCIe fabric; CXL fabric	PASS
B	Deny CXL endpoint hot-add, binding and runtime reconfiguration: Φ_B and Φ_B^{seq} on M_{intent}; observability on M_{spec}	CXL hot-add; switching binding; runtime configuration	PASS (classified); coverage and runtime GAP(model)

TABLE IV
INTENT 1 ATOMIC OBLIGATIONS, EACH GROUNDED IN AN EXTRACTED STATE (a_i) OR TRANSITION (b_i); REFERENCES WERE CHECKED AGAINST THE ARCHIVED SOURCE EDITIONS. COLUMNS GIVE THE NUSMV VERDICT ON EACH MODEL; "REACH" AND "OBSERVABLE" MEAN EF HOLDS.

Atom	Grounded obligation	Ref.	M_{spec}	M_{intent}
Clause A: allow USB HID hotplug via CXL/PCIe fabric ($EF\ a_i$)				
A1	HID interface class code checked	HID 4.1	reach	PASS
A2	Hub detects a connected device	USB 2.0 11.1	reach	PASS
A3	USB host reaches an active device	USB 2.0 4.9	reach	PASS
A4	PCIe fabric topology discovered	PCIe 1.3	reach	PASS
A5	CXL virtual hierarchy established	CXL 7.7.6.1	reach	PASS
Clause B: deny CXL endpoint hot-add at runtime ($EF\ b_i$; $G\ \neg b_i$)				
B1	No CXL hot-add (4 transitions)	CXL 7.7.12.6, 12.1	observable	PASS
B2	No switching port bind	CXL 7.7.12.6.2	observable	PASS
B3	No runtime configuration trigger	CXL 7.7.12.6.2	observable	PASS
Φ_B	Composed deny, $G\ \neg(b_1 \vee b_2 \vee b_3)$		FAIL	PASS
Φ_B^{seq}	No CXL event after a USB attach		FAIL	PASS

the specification permits and are not vacuous. On M_{intent} every deny atom, the composed Φ_B and its sequenced reading hold. Three limits remain. First, the exclusion covers only the features A3 classified as deny. An audit of every extracted CXL feature whose name denotes hot-add finds seven more that A3 left unclassified, including incremental and managed add flows in CXL 4.0 Section 14.7.5 and binding flows in Sections 14.7.7 and 7.2.3 [12]. Unclassified features are neither monitored nor excluded, so Clause B has a clause-level GAP(model): the model check is sound for what was classified, and the classification is incomplete. Second, the extracted machines are not coupled across specifications, so the models cannot order a USB attach after the fabric comes up; NuSMV finds a path that attaches a USB device with PCIe uninitialised. This is why fabric reachability is a scope assumption, not a verdict. Third, the same Clause B gap also appears in the runtime-model output because the generated Intent LTS lacks denied-event mappings and excluded-event index entries for CXL hot-add; the runtime runs of O3 supply them by measurement.

c) Intent 2, PCIe single specification.: **Intent.** "Only allow hotplug of single endpoints of network class devices."

TABLE V
INTENT 2 ATOMIC OBLIGATIONS, WITH THE NUSMV VERDICT ON EACH MODEL; SECTIONS WERE CHECKED AGAINST THE ARCHIVED PCIE 4.0 V0.3 DRAFT.

Atom	Clause	Section	M_{spec}	M_{intent}
ob_hotplug_event	insertion reachable	6.7.3	reach	PASS
ob_pcie_endpoint	only PCIe Endpoints	7.1	observable	PASS
ob_network_class	class code is 0x02	–	GAP(catalog)	
ob_single_function	header type bit 7 is 0	7.5.1.5	observable	PASS
Φ'	grounded subset		FAIL	PASS

Specification. PCIe 4.0 v0.3 draft.

This extends the existing network-class PCIe hot-plug proof-of-concept by requiring a single-function endpoint.

Obligation decomposition. The intent is decomposed into one allow-only safety clause:

$$\Phi_{PCIe} = G(H \rightarrow E \wedge N \wedge S), \tag{4}$$

where H is the hotplug-event atom, E is the PCIe-endpoint atom, N is the network-class atom and S is the single-function atom. Under Lemma 1 these atoms take different forms. H is an allow atom: hot-plug insertion must stay reachable, $EF\,h$. E restricts *which* function may appear, so it is a deny atom over the alternatives: $G\,\neg(\ell \vee r \vee x)$, where ℓ, r and x are the transitions that detect a Legacy Endpoint, a Root Complex Integrated Endpoint, or a function without a Type 00h header. S is a property constraint on the Type 00h transition t, whose observed header_type register reads 0x00 for a single-function endpoint and 0x80 for a multi-function one: $G(t \rightarrow header_type = \texttt{0x00})$. The grounded subset composes on M_{intent} as $\Phi' = G(\neg(\ell \vee r \vee x) \wedge (t \rightarrow header_type = \texttt{0x00}))$.

Models. For Intent 2, A3 classifies seven PCIe 4.0 catalog features to construct M_{intent}. M_{spec} is their product (27 states, 26 transitions; 214,080 reachable states), with header_type free over the values a Type 00h header permits. M_{intent} disables the Legacy and Root Complex Integrated Endpoint features and the non-Type-00h transition, and applies the header_type constraint (2,508 reachable states).

Table V reports the grounding and paired Spec/Intent model results for Intent 2.

ob_network_class is a catalog gap: A1 missed the PCIe class code register, so no feature, field or property can carry 0x02. A2 cannot recover absent A1 features; Pass-B/back-reference audit can flag them, but here it remains GAP. ob_single_function is grounded: A1 extracted the Header Type multi-function bit (Section 7.5.1.5), and the Type 00h endpoint machine observes the header_type register on which the constraint is placed.

Each grounded atom passes on M_{intent} and is non-vacuous: insertion and single-function Type 00h endpoints stay reachable, while the excluded functions and multi-function headers are observable on M_{spec}, where Φ' fails. As in Intent 1, the

TABLE VI

TEN-INTENT BENCHMARK. I1–I2 EXERCISE A3; I3–I10 USE AUTHOR-GROUNDED OBLIGATIONS. INVAR REPORTS PASSES AND GAPS (C: CATALOG, M: MODEL). BASELINES REPORT CLOSED-WORLD $E_c/G/V_c$; PF IS PARSER FAILURE.

#	Intent (source)	Specs	INVAR pass; gap	nl2spec $E_c/G/V_c$	NL2LTL $E_c/G/V_c$
1	USB HID hot-plug via fabric; deny CXL hot-add (Intent 1)	HID, USB 2.0, PCIe, CXL	8/8; M: coverage/mapping	–	–
2	Single-function network-class PCIe endpoints (Intent 2)	PCIe	3/4; C: class code	Y/2-of-3/0	N/0-of-1/0
3	No USB device inserted after boot (UC1)	USB 2.0	2/2; –	Y/1-of-4/0	PF
4	USB keyboards and mice only (UC2)	HID	4/4; –	N/2-of-4/1	PF
5	USB keyboards only	HID	4/4; –	N/1-of-3/1	PF
6	Bluetooth keyboard/mouse via USB dongle only (UC3)	USB 2.0	1/2; C: Bluetooth	N/2-of-5/0	PF
7	Single-function USB mass-storage only	USB 2.0	1/2; M: #functions	N/1-of-2/1	PF
8	CXL Type 3 memory devices only (UC5)	CXL	3/3; –	N/0-of-4/1	PF
9	No extra host may share CXL memory (UC5)	CXL	3/3; –	N/0-of-2/1	PF
10	Outbound TCP to port 443 only (UC6)	TCP	1/2; M: port	Y/1-of-2/0	PF
	Total	5 specs	30/34; 4 atoms, 1 clause	3/9; 10/29; 5	0/9; 0/1; 0 (8 PF)

TABLE VII

VERIFIER VERDICTS ON THE UBUNTU 24.04 QEMU 8.2 GUEST (KERNEL 6.8.0-136-GENERIC, A Q35 MACHINE). INTENT 1 CLAUSE A ROWS USE THE REPAIRED RUNTIME MODEL.

Intent	Runtime scenario/action	Verdict
2	Network controller, single function	`ALLOW`, whole session
2	Audio endpoint	`VIOLATION(property)`: class 4, not 2
2	Network controller, multi-function	`VIOLATION(property)`: multi-function
1A	USB host controller, keyboard behind it	`ALLOW`*
1A	USB host controller, storage behind it	`VIOLATION(property)`: interface class 8, not 3*
1B	CXL Type 3 memory device	`VIOLATION(intent)`: probe, bind, detach
1B	(unbind and rebind its memdev)	`VIOLATION(intent)`: detach, bind
1B	(request a region)	`VIOLATION(intent)`: region request

*Both Intent 1A rows use the author-confirmed three-repair model; as extracted, both also produce an unrelated property violation and bypasses.

hot-plug machine is not coupled to the endpoint machines, so the models cannot tie an insertion to the function it adds; that link is made at runtime by the event's device identity. The composed Φ' covers the grounded subset only; the gap atom is reported separately. The hand-crafted runtime test artefact exercises verdicts but does not fix the catalog gap; O3 runs it on real hot-plug, with the E and S checks it lacked added. **Reference-grounded benchmark.** Table VI adds eight intents to the two A3 case studies. Authors ground I3–I10 directly, so those rows evaluate checking and gap reporting rather than automatic decomposition. NuSMV completes 111 consistency checks without parser or model-checker errors. The benchmark covers reachability/non-vacuity and safety, not universal eventuality or fairness-dependent liveness.

NL-to-LTL diagnostics. For I2–I10 we freeze an author-reviewed expected LTL formula and proposition binding solely as a scoring oracle. Under a declared closed-world completion, E_c is bidirectional model-relative equivalence, G counts grounded generated atoms, and V_c counts silent passes from false ungrounded antecedents. NL2LTL's strict `PATTERN/SYMBOLS` parser rejects eight responses (PF); PF is not an LTL-syntax verdict.

d) O3: Runtime verifier validation.: Generated tracepoint names do not exist on the evaluated kernel, so bindings are grounded by measurement and independently re-measured. This covers 20/23 USB/HID transitions, all four PCIe transitions for each intent, and the three exercised CXL actions; three USB error states and three Fabric-Manager transitions have no host counterpart. The adapter reads class and header properties from kernel objects.

Table VII summarises the runtime scenarios and observed verifier verdicts.

With measured bindings, each reported verdict matches the intent. Clause A additionally needs three documented runtime-model repairs: missing USB roots, a one-session/four-machine mismatch and a hub-level class constraint that rejected interface-class keyboards. Both as-extracted and repaired outcomes are retained; a virtual machine can request but not commit a CXL region.

This is a focused feasibility study of ten intents, two in depth; the Intent 2 catalog gap aligns with the NuSMV GAP verdict and runtime property violations.

Automation boundary. INVAR automates candidate extraction, A3 decomposition for I1–I2, grounding checks, NuSMV model/property generation and verdict collection. Human decisions remain in source-audit adjudication, author-grounding of I3–I10, measured runtime bindings and the three runtime-model repairs. The evaluation therefore demonstrates semi-automated, traceable checking and explicit gap exposure; it does not establish fully automatic formalisation, unassisted reproducibility or reduced engineering effort. Both as-extracted and repaired runtime artefacts are retained.

B. Threats to Validity

Internal validity. Maintained artifacts risk scenario overfitting; measured adapters and the three runtime repairs are author decisions, although independently re-measured. The same Codex installation generated and repaired the audits, so their results remain pending author adjudication. **Construct validity.** Pass B values are proxies; HID uses a seeded sample, while the USB/CXL target-coverage regressions use repair-guided, intent-scoped inventories. Their precision values use development samples from substantive specification slices; neither precision nor coverage is an independent whole-specification estimate. Baseline E_c and V_c depend on the declared closed-world completion. **External validity.** Numbered hardware

standards ease provenance; repaired experiments cover three protocols, but portability beyond protocols remains unproven. Runtime evidence is Linux/eBPF-specific and QEMU-based. **Conclusion validity.** We evaluate safety/reachability correctness, not universal liveness, performance or probe overhead. Only I1–I2 exercise A3 in depth; I3–I10 test author-grounded checking. Human effort was not timed, so no labour-reduction claim is made.

VII. Related Work

Runtime verification and monitoring. MOP [1] checks traces against manually written properties. INVAR derives grounded obligations from specification text before monitoring. Falco [4] and Tetragon [5] enforce kernel-level eBPF policies [17]; INVAR instead uses eBPF as an event adapter for an Intent LTS with explicit catalog and runtime GAP verdicts.

LLM formalisation, hardware and mining. nl2spec [2] and NL2LTL [3] generate temporal properties; INVAR additionally grounds propositions, audits provenance and checks paired evidence/satisfaction queries. Hardware-security work verifies register-transfer-level (RTL) properties [18], while specification mining covers traces [19] and RFC-derived finite-state machines (FSMs) [20]. INVAR links document-derived machines to model-checking verdicts and runtime evidence.

VIII. Conclusion

INVAR carries natural-language intent from source specifications to model checking and runtime evidence while preserving clause-level traceability. It decomposes intent into temporal atoms, grounds them to a neutral Spec LTS, compiles CTL/LTL NuSMV properties, checks Spec evidence and Intent satisfaction, and reports PASS, FAIL or named GAP evidence. Audit-guided repair raises reference coverage from 55.3/65.2/42.4% to 97.4/87.0/100.0% for TCP/USB 2.0/CXL 4.0. Strict (lenient) precision is 91.7% (95.5%) for TCP, 95.0% (97.5%) for the USB development slice and 87.5% (100.0%) for the CXL development slice; evaluation exposes missing catalog or runtime observables instead of silently accepting vacuous properties.

Runtime binding remains the main boundary: A3 generates candidate descriptors, but the QEMU study uses an experiment-specific adapter. Future work will automate and validate Intent-LTS-to-kernel mapping and runtime-model event acceptance, support platforms beyond Linux/eBPF, strengthen temporal composition, and use retrieval-assisted catalog repair while preserving auditability.

References

[1] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Rosu, "An overview of the MOP runtime verification framework," *International Journal on Software Tools for Technology Transfer*, vol. 14, no. 3, pp. 249–289, 2012.

[2] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in *Proc. CAV*, 2023, pp. 383–396.

[3] F. Fuggitti and T. Chakraborti, "NL2LTL: A Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas," in *Proc. AAAI*, vol. 37, no. 13, 2023, pp. 16428–16430.

[4] Falco Project, "Cloud native runtime security," CNCF, 2023. https://falco.org.

[5] Isovalent, "Tetragon: eBPF-based security observability and runtime enforcement," 2023. https://tetragon.io.

[6] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[7] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.

[8] S. Sood, A. Malik, and P. S. Roop, "A novel approach to real-time contract based reasoning for hybrid systems," in *Proc. 20th ACM-IEEE Int. Conf. Formal Methods and Models for System Design (MEMOCODE)*, Shanghai, China, 2022, pp. 1–11, doi: 10.1109/MEMOCODE57689.2022.9954599.

[9] USB Implementers Forum, "Device class definition for human interface devices (HID), version 1.11," Jun. 2001.

[10] USB Implementers Forum, "Universal Serial Bus specification, revision 2.0," Apr. 2000.

[11] PCI-SIG, "PCI Express base specification, revision 4.0, version 0.3," draft, Feb. 2014.

[12] CXL Consortium, "Compute Express Link specification, revision 4.0, version 1.0," evaluation copy, Nov. 2025.

[13] W. Eddy, Ed., "Transmission Control Protocol (TCP)," RFC 9293, IETF, Aug. 2022.

[14] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Proc. CAV*, LNCS 1633, 1999, pp. 495–499.

[15] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[16] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927.

[17] D. Thaler, Ed., "BPF instruction set architecture (ISA)," RFC 9669, IETF, Oct. 2024.

[18] N. Farzana, F. Rahman, M. Tehranipoor, and F. Farahmandi, "SoC security verification using property checking," in *Proc. ITC*, 2019, pp. 1–10.

[19] G. Ammons, R. Bodik, and J. R. Larus, "Mining specifications," in *Proc. POPL*, 2002, pp. 4–16.

[20] M. L. Pacheco et al., "Automated attack synthesis by extracting finite state machines from protocol specification documents," in *Proc. IEEE Symp. Security and Privacy*, 2022, pp. 51–68.