

Ontology-grounded logic reasoning for validating structured LLM outputs

András Földvári, András Pataricza

0000-0002-4559-6990, 0000-0002-6516-129X

Budapest University of Technology and Economics

Department of Artificial Intelligence and Systems Engineering

Budapest, Hungary

Email: {foldvari.andras, pataricza.andras}@vik.bme.hu

Abstract—Large language models (LLMs) are increasingly used to produce structured artifacts that downstream pipelines consume directly, yet no correctness guarantees exist for these outputs in critical applications. Typical guardrail solutions for validation of LLM outputs act as placeholders rather than providing the strict correctness guarantees needed. We propose a conceptual framework and pilot implementation that grounds validation of structured LLM responses in embedded formal methods. The apparatus rests on three commitments. First, it emits three-valued verdicts that preserve the diagnostic distinction between acceptance, structured review, and rejection. Second, it expresses validation as short declarative logic-program modules over a shared ontology vocabulary, so analyses compose without ad-hoc integration code and remain auditable. Third, it keeps a domain-agnostic analytical core and extends it per domain through a bounded extension layer that lifts existing metamodels and knowledge bases into the shared vocabulary.

I. INTRODUCTION

LARGE language models (LLMs) promise a revolutionary increase in productivity while simultaneously reducing the skills needed to perform a task. Their use goes well beyond serving as a user-friendly human-machine interface by supporting information acquisition, fusion, formalization, reasoning, and code generation. Their ability to produce structured artifacts makes them a first stage in complex workflows, where downstream pipelines consume their outputs directly.

However, no correctness guarantees exist for the produced artifacts in critical applications, which can invalidate subsequent steps in a workflow due to LLMs' irreproducible behavior stemming from complex, nondeterministic operations and sensitivity to incomplete training data (bias, errors, adversarial attacks).

Existing Verification and Validation (V&V) mechanisms for LLM outputs (typically *guardrails*) provide partial filtering but no correctness guarantees, and traditional quality assurance methods do not transfer to such tremendously complex black-box components. Our approach instead aims for strict correctness guarantees by adopting well-proven, rigorous formal methods.

The trustworthiness of a structured LLM response does not require a single binary Go/NoGo decision like those delivered by verifiers in current industrial use. Rather, it requires a structured, rich multi-analysis diagnostic V&V approach.

LLMs could fail in multiple characteristic ways. *Hallucination* creates entities that do not exist in the referenced knowledge base. *Abstraction level confusion*, for instance, delivers a concrete instance instead of a class or vice versa. *Faults in reasoning* may omit consequences that follow from the input by transitive reasoning, or they postulate claims unsupported by their inputs. When an automated workflow or a human user (an engineer, a compliance officer, or a safety analyst) consumes such an erroneous artifact, the error propagates through subsequent steps and can have severe operational consequences.

The engineer must learn not only that a response is suspect but also which dimension of the response's claims the suspicion arises from, and whether the available evidence supports rejection, conditional acceptance, or a structured review by a human. The approach must also be operationally tractable: low authoring cost per new domain, low runtime cost per response, and reuse of existing metamodel and well-formedness-rule infrastructure where it already exists.

Current industrial practice (surveyed in Sec. II-B) compresses validation into a Go/NoGo answer, which is the source of the operational discomfort with current LLM V&V. Such a verdict cannot distinguish a benign anomaly from a hallucination, cannot tell the engineer *which* of several independent analyses are in tension, and cannot route a boundary case to a structured review queue rather than a rejection. It also leaves the validator's own discriminative assumptions unexposed to audit.

A reusable apparatus, grounded in a per-domain metamodel and extended with meta-level patterns drawn from the model-transformation tradition, can produce this diagnostic at operationally acceptable authoring cost.

We propose a conceptual framework and pilot implementation along these dimensions:

- *Three-valued verdicts* We argue that the acceptance/review/rejection decision belongs in the verdict-information operator itself. Two boundary claims that arrive in review for structurally different reasons remain distinguishable. That distinction is the diagnostic the engineer acts on.
- *Logical reasoning* The per-claim analyses are short logic-program modules over a shared ontology vocabulary.

They derive verdicts from the *absence* of supporting evidence, aggregate over witnesses, and compose declaratively without ad-hoc integration code. Each analysis is a declarative program rather than a procedural rule, and the apparatus is auditable in the engineering sense.

- *Domain-agnostic approach with domain-specific extensions* The analytical core is uniform across domains. Per-domain use adds an extension layer that lifts the engineer’s existing metamodels, well-formedness rules, and knowledge bases into the shared vocabulary the core reasons over. Domain-specific modelling work is still required, but it is bounded to this extension layer and reuses assets the engineer already maintains rather than restarting from scratch.

We ground the paper in a single running example throughout (Sec. V): an LLM decomposing one functional-safety requirement into sub-requirements. Such a task admits many valid answers, and the LLM, like a human, works from the context it is given. Our aim is therefore not to judge whether a decomposition is the correct one, but whether a given decomposition fits the established engineering frame — the metamodel, the knowledge base, and the rules already in force.

Section II reviews the formal apparatus the paper builds on and surveys related LLM-output validator families. Section III situates our position in a three-stage architectural picture of structured LLM validation. Section IV describes the apparatus abstractly. Section V instantiates it on an MBSE running example. Section VI closes with directions for further work.

II. BACKGROUND

This section provides the background the rest of the paper builds on. Section II-A reviews the formal machinery the apparatus consumes — Answer Set Programming, a description-logic type layer, SHACL-style conformance, and rough-set three-way decisions. Section II-B surveys the neighbouring families of LLM-output validators and positions our approach against them.

A. Formal preliminaries

The framework is organised as a modular logic program. A single reasoning core hosts an ontology-inspired type layer, a SHACL-style conformance layer, and a rough-set interpretation layer, all sharing one fact space and one solver call.

We are using *Answer Set Programming* [1] (ASP), a declarative logic-programming paradigm based on the stable-model semantics, which provides the logic layer on which every validation mechanism is built. Every layer below is a short program in the shared formalism, executed by the Clingo [2] ASP solver.

In [3], the authors introduced a taxonomy of validity conditions on model transformations that abstract over specific metamodels: refinement realisation, conservative extension under signature change, derivability from evidence, and abstraction. The patterns were formulated as *generic* obligations precisely so they could be reapplied across signatures. The framework instantiates four of them as *signature-independent*

meta-transformations encoded as parameterised ASP patterns, so that every case study reuses the same logical core and differs only in the supplied fact vocabulary.

The meta-transformations need a typed fact graph to operate over. The framework supplies one by re-encoding a fragment of *Description Logic* (DL) ontology semantics [4]. It covers subclass hierarchy, property domain and range, disjointness, and OWL 2 property-chain axioms as ASP rules over typed predicates.

Stable-model semantics is natively closed-world, a fact unsupported by any rule is false, which is exactly the regime needed to phrase “this hazard is absent from the corpus” as a derivable verdict. Also, the type, conformance, and rough-set layers share a single fact space and a single solver call, so a clash in one layer is visible to the next without serialisation through an external reasoner. The DL fragment we re-encode is intentionally lightweight. It captures the consistency conditions that are load-bearing for LLM-output audit (type discipline, disjointness, chain implication) without inheriting the open-world commitments that would block negation-as-failure.

The *Shapes Constraint Language* [5] (SHACL) declares shape graphs that phrase structural obligations on a typed fact graph (property cardinality, value-class membership, datatype, and node kind). The framework re-encodes SHACL’s structural fragment as ASP rules with aggregation over the inferred-type relation produced by the layer below. The violations appear as explicit ASP facts that the interpretation layer processes alongside the type-layer clashes, without invoking a separate SHACL engine.

The verdicts emitted by the validation layers are partitioned by a rough-set interpretation layer on top. Rough set theory [6] was introduced as a model of approximate classification under indiscernibility. Objects sharing all available attributes form an equivalence class, and the target concept is approximated by its *lower* and *upper* bounds plus the *boundary* between them. The framework applies it directly over the verdicts emitted by the layers above. Each LLM-output element is described by the set of oracle verdicts that apply to it, and the resulting indiscernibility partition forms the *partition* stage of the apparatus.

This way, instead of a binary verdict, we get a *three-way decision* primitive as described in [7], [8] with the boundary region carrying the items that warrant human review. Similarly, the indiscernibility partition admits the notion of a *reduct* as described in [9]: a minimal subset of oracle attributes that preserves the same partition.

B. Related work

This section reviews approaches addressing LLM-output validation. Four families recur, each tackling a subset of the problem through a different formal substrate.

Validators built on SHACL pair shape constraints with an external oracle. The *Licensing Oracle* [10] validates LLM claims against a structured knowledge graph using SHACL shapes, returning a binary verdict with deterministic abstention

guarantees. It delivers structural conformance, but does not phrase non-monotonic schema evolution, conditional per-role aggregation, bidirectional roundtrip, nor reduct-based discriminative audit.

OWL/DL reasoners underpin a second family. NeOnGPT [11] drives an LLM through an ontology-construction pipeline, re-prompting until Hermit and Pellet report the result consistent, and [12] verifies individual RDF triples (from Wikidata) against external evidence retrieved from Wikipedia and web search.

Constraint-satisfaction and SMT solvers underpin a third family, exemplified by *Eidoku* [13], which encodes LLM reasoning verification as a CSP with a scalar violation cost combining graph-connectivity, feature-space, and logical-entailment proxies.

A fourth line uses LLMs themselves as judges and curators of knowledge graphs. In [14], the authors deploy LLMs as zero-shot judges to validate RDF-triple insertions across four task heads (class/property alignment, URI standardisation, semantic consistency, syntactic correctness). Because each verdict is the prose output of the judging LLM, the audit trail inherits the LLM’s sensitivities. Two runs may flag the same triple differently, and prompt rewordings shift the decision boundary — an effect also reported for LLM-based screening in regulated domains, where keyword-anchored prompts yield markedly more consistent verdicts than open-ended ones [15]. The framework operates strictly above the LLM layer. Its verdict is a function of the LLM’s structured output and static ASP rules, so the same graph routes to the same verdict regardless of which LLM produced it.

Across these families, validation is compressed into a binary or scalar verdict over a single substrate, and each leaves at least one of the questions this paper raises unaddressed: non-monotonic schema evolution, conditional per-role aggregation, three-valued routing of residual uncertainty, or a reduct-based audit of which evidence drove the verdict. The framework is complementary rather than competing. It operates strictly above the LLM, reuses the engineer’s existing ontologies and metamodels through a thin bridge, and composes these checks as declarative analyses over a single formalism, so it returns a structured per-claim diagnostic where the surveyed validators return one accept/reject.

III. CONCEPTUAL WORKFLOW

We propose a conceptual three-stage structured LLM output validation pipeline (Fig. 1). The pipeline starts with an LLM response, and emits, per claim, either a compliant verdict or one of two non-acceptance signals: a *retry/reject* signal for hard failures, or a *review* signal for cases that require a human refining question.

The LLM produces a structured *response* to a prompt. The response consists of *claims* (discrete assertions about the world). A claim enters the pipeline as a *response claim*.

To make the claims analysable, the per-domain bridge encodes the response as a *response model*, a graph whose typed nodes are *model elements* drawn from the *domain signature*.

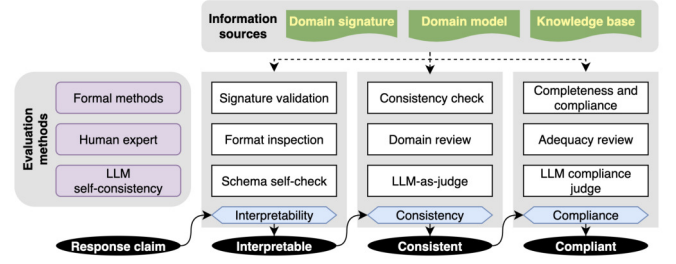


Fig. 1. The three-stage architecture for structured LLM output validation.

The framework’s analyses reason over this response model against two further information sources, the *domain model* and the *knowledge base*.

The three columns of Figure 1 are the sequential pipeline steps. *Interpretable* checks that the claim is syntactically and schematically well-formed. It must parse against the declared output schema and conform to the structural signature expected by the target domain. *Consistency* checks whether the parsed claim agrees with the domain knowledge already in the system, along four axes: the schema’s structural constraints, the source artefacts that anchor each asserted fact, the refinement relations between abstract and concrete model elements (and their layer-coherence), and the system’s behaviour under schema evolution. *Compliance* checks if an interpretable and consistent claim falls inside the target class that the use case at hand treats as acceptable.

The three rows are the alternative evaluation-method sources. In principle any cell can be answered by any row, though their cost, latency, and trust profiles differ. *Formal methods* answer through declarative specifications. *Signature validation* ensures that the response parses against the declared output schema, so that downstream analyses have a typed object to reason over. *Consistency check* derives type-clashes and constraint violations from the schema together with the domain model, catching contradictions before any compliance question is asked. *Completeness and compliance* verifies that the surviving claim carries the content the use case requires and satisfies the declared acceptance rules.

Human experts answer through review: *format inspection* (well-formedness in the declared notation), *domain review* (asserted facts cohere with project context), and *adequacy review* (fitness for the actual use). *LLM self-consistency* answers by re-prompting a sibling model as *schema self-check*, *LLM-as-judge* against retrieved domain context, or *LLM compliance judge* against the acceptance criteria. Both rows are complementary to the formal-methods row this paper develops and lie outside its scope.

The top panel lists three information sources that feed every validation step. The *domain signature* is the declared vocabulary of the domain: the types of entities and the structural relations that may hold between them. The *domain model* records the concrete situation the response refers to — the particular project, document, or deployment scenario at hand — as factual assertions over that vocabulary. The

knowledge base is everything else that the validator can appeal to (e.g., external oracles, registry snapshots, labelled historical examples).

The signature is declared once per domain, the domain model is collected per validation run, the knowledge base supplies the per-stage oracle inputs and the empirical-prior history.

The framework presented in Section IV instantiates the formal-methods row as a set of composable analyses over a shared ontology, and refines the compliance gate in two ways.

IV. ONTOLOGY-GROUNDED FOLEGYRMAL VALIDATION

This section presents the formal-methods layer of the framework that is organised around three properties:

- *Structured per-claim verdicts.* A claim that cannot be accepted is reported along multiple verdict dimensions, so the engineer can see which specific aspect of the claim is in doubt.
- *Composable analyses over a shared formalism.* The validation questions are expressed in a single formal language, so they compose into one audited result rather than producing parallel outputs that have to be reconciled.
- *Reuse of existing engineering assets.* The framework consumes the domain ontologies, metamodels, and traceability data the engineer already maintains, through a thin per-domain bridge.

A. Formal validation framework

The framework is organised as two layers (Fig. 2). A *domain-agnostic core* hosts the validation analyses themselves: each analysis is a short program over a shared vocabulary. A *per-domain bridge* is the domain-specific layer attached to the core, a thin translation layer that maps domain-native facts onto the small set of generic relations the analyses expect, so the core sees uniform input regardless of the originating domain. The analyses described below all live in the core; everything above the bridge is reusable.

The bridge targets domains that already maintain structured domain knowledge (ontologies, metamodels, taxonomies, traceability registries) as part of normal engineering practice (adoption that is broadening within MBSE workflows [16] and industrial regulated settings) so the framework can consume these existing assets directly.

The core is written in ASP. The three ASP features the framework leans on are: (1) negation-as-failure for verdicts that fire on the absence of supporting facts (where an LLM's silence is itself informative), (2) aggregation for the cardinality, witness-counting, and equivalence-class quantification patterns, and (3) declarative composition for short rules each of which a reviewer can read in isolation.

B. Validation analyses

A structured verdict is a tuple of attribute values, one per independent dimension of model trustworthiness. The domain-agnostic core produces such verdicts in two layers. Five *per-claim oracles* each formalise one established primitive and

emit an independent verdict attribute on every model element. Further *aggregation and meta-analyses* operate on the resulting verdict table rather than on the model elements directly. They aggregate the per-claim attributes into a routing decision, audit the apparatus's own discriminative power, and grade a complementary completeness question against the engineer's ground truth.

The structural oracles (*conformance* and *anchor*) check the model against the metamodel and against its traceability to source artefacts. The *migration*, *refinement*, and *roundtrip* checks draw on the transformation patterns [3].

The *conformance analysis* asks whether each model element is well-formed against the metamodel. It compares the element to the metamodel along multiple dimensions (typing, multiplicity, property domain and range, and disjointness) and emits per element a verdict of *conformant* or *violates*. A wrong type, a missing required property, a multiplicity breach, or a disjointness clash each counts as a violation.

The *anchor analysis* asks whether each model element traces back to a source artefact. It walks from the source-artefact registry (e.g., stakeholder requirements, manifest entries, knowledge-base assertions) along permitted model relations, and accepts an element as *anchored* on either a *direct* match (the element names an artefact already present) or a *transitive* one (a chain leading from an artefact that is itself anchored). Otherwise the element is reported *unanchored*, meaning an orphan claim the available evidence cannot back.

The *migration analysis* asks whether the validity conditions a model satisfied under one metamodel (the *old* schema, in force before a version change) still hold under a *new* one (the schema after the change). It runs the same set of model elements through both metamodels and surfaces drift in either direction. It highlights elements that satisfied the old metamodel and no longer satisfy the new one (e.g., silent breaks like a renamed property, a tightened multiplicity, or a removed subtype), as well as elements the new metamodel admits but the old one rejected. The verdict separates a *conservative* migration, one that preserves all earlier verdicts, from a *breaking* one.

The *refinement analysis* asks whether each abstract element has a concrete realisation that preserves its properties. It compares an abstract element against a candidate concrete realisation at the lower layer, and checks that the abstract element's properties carry forward into the realisation. For example, a "triple-modular-redundant" requirement must be realised by three independent voting channels, not by a single channel with an internal self-check. The analysis covers two complementary patterns. (1) A *relational* one, in which an abstract relation between two abstract elements must be backed by a chain of concrete relations between their realisations, and (2) an *existential* one, in which an abstract element must have at least one concrete witness. A realisation that loses some abstract properties is reported *partially realised*. An abstract element with no realisation at all is *not realised*. An element with a property-preserving realisation is *realised*.

The *roundtrip analysis* asks whether the concrete realisation,

when abstracted back, yields the abstract element it was meant to realise (e.g., the structural counterpart of inspecting a deployed redundancy architecture to confirm that its three claimed channels really are independent voting channels, not a re-labelled single channel).

It compares the original abstract element against the abstraction of its concrete realisation. A failed roundtrip reports *level-mixing*: the concrete realisation implements a different abstract concept than the one it claims to realise. The absence of a roundtrip witness at all is itself a verdict, since the element cannot then be coherently placed at either layer.

C. Aggregation and meta-analyses

Three meta-analyses operate on the verdict table the per-claim oracles emit: a partition that aggregates the per-element tuples, an audit of the apparatus's own discriminative power, and a completeness check that closes the structural gap claim-side oracles leave open.

The *partition analysis* is not itself an oracle but the verdict-formation operator that aggregates the preceding per-element verdict tuples into a three-valued routing decision against one or more engineer-declared target classes. It applies the rough-set construction directly on the verdict tuples. Model elements with identical tuples are indiscernible on the available evidence and must share their compliance label. Each engineer-declared target class then assigns every indiscernibility class to structured acceptance (the whole class lies in the target), structured review (the class straddles the target), or structured rejection (the class is disjoint from the target).

The construction operates in either of two regimes. When the target class is itself a function of the condition attributes the indiscernibility classes are by construction homogeneous with respect to the target and the boundary is empty. In this case, the operator delivers the three-valued routing under indiscernibility. When the target is independent of the condition attributes (e.g. an external expert verdict over the same objects), the boundary becomes non-empty and its width measures the adequacy of the attribute set to reproduce the target.

The same reasoning core that produces the partition also lets the framework audit its own discriminative power. A reduct enumeration over the attribute set identifies the minimal subsets of oracle attributes that preserve the partition, and equivalently answers the dual question of how few requirements suffice to make the response unambiguous against the knowledge base. The audit produces three operational signals: (1) Per element, it exposes *verdict sensitivity*: a model element whose partition cell moves when an attribute is dropped is one whose verdict rests on that attribute in particular, which is the information a reviewer needs to defend or contest the verdict downstream. (2) Per corpus, it produces a *coverage diagnostic*: a partition preserved by a small reduct indicates that the corpus did not stress the remaining oracles, so the result reads as a comment on the corpus's discriminative variety rather than as a permanent ranking of the apparatus. (3) When the target is independent of the condition attributes, the same enumeration further yields an *attribute-adequacy diagnosis*: an attribute

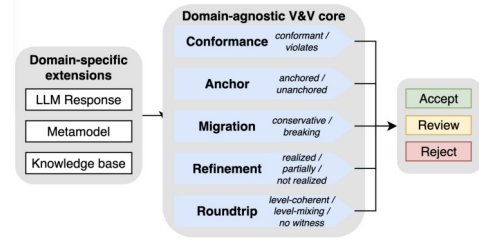


Fig. 2. Analysis pipeline

that uniquely splits an indiscernibility class along the target's verdict line is promoted to indispensable, naming what the present condition set is missing to reproduce the target.

The per-claim oracles all interrogate the structural *support* of claims the LLM has made. A target element the LLM should have enumerated but did not produces no claim, and therefore no verdict. The completeness analysis closes this gap by quantifying over targets rather than claims. The partition is parametrised by a target-class declaration and a knowledge-base-derivable membership relation, and the same two ingredients support a one-rule completeness analysis that reports as *missed* any target individual the LLM did not claim. Together with the anchor analysis this forms a bidirectional evidence binding. Anchor flags claims with no KB support, completeness flags KB targets with no claim witness. The analysis engages whenever the prompt contributes a structured target predicate (e.g., via a prompt template where the target rides on the schema, on the same footing as the output schema; or via the validation harness when the prompt is not itself machine-readable). The target predicate is KB-derivable, so the analysis grades completeness only against the engineer's own ground truth, not against real-world targets the KB does not record. An open-ended prompt with no enumerable target produces no missed-target verdicts, which is the correct behaviour since completeness is undefined where the request does not name a closed target.

V. EXAMPLE

We present the validation approach on a single end-to-end scenario by a typical MBSE process. An LLM decomposes one high-level safety requirement of a process-industry Emergency Shutdown System into eight implementable sub-requirements SR-01...SR-08. The requirement is HLR-001 ("shut the process down within the safe-state interval whenever an unsafe condition is detected"), allocated Safety Integrity Level SIL-3 on the IEC 61508 functional-safety scale.

The knowledge base around HLR-001 holds a hazard corpus, the hardware architecture (sensors and actuators), the SIL allocation, a parent-concern relation linking each hazard to its framing high-level requirement, and a legacy obligation registry inherited from earlier releases. A slice (HLR-001, the in-scope hazards, and the available hardware) is passed to the LLM as prompt context. The same knowledge base feeds the validation pipeline through a thin per-domain bridge. The LLM produces one structured claim per sub-requirement,

naming the hazard, the realising hardware, and the SIL. Two representative claims:

```
SR-01: { hazard: high-temp,
        realised_by: [cool-valve],
        sil: SIL-3,
        parent: HLR-001 }
```

```
SR-08: { hazard: overpressure,
        realised_by: [sensor:S7],
        sil: SIL-4,
        parent: HLR-001 }
```

Throughout the example the LLM only proposes; nothing is accepted on its word. The decomposition shown is one recorded input, and it deliberately keeps the claims the LLM got wrong — a hallucinated hazard, an under-redundant channel, a mis-scoped requirement with a conflicting SIL — to show the validator catching them.

The pipeline runs every sub-requirement through five per-claim oracles (*conformance*, *anchor*, *migration*, *refinement*, *roundtrip*) and a routing layer that aggregates the verdicts. Table I reports the per-stage verdicts. One column per LLM-claimed sub-requirement, one row per stage, with the last row giving the three-valued routing label (lower: route to acceptance; review: structured review; outside-upper: hard rejection). Cells use ✓ (clean), △ (non-fatal flag), × (hard rejection), — (no verdict). Four sub-requirements (SR-01, SR-02, SR-03, SR-07) pass every stage; two (SR-04, SR-06) land in review for structurally distinct reasons; two (SR-05, SR-08) are rejected from scope.

SR-04 (single cooling actuator) passes every other stage and is flagged only by *refinement*. At SIL-3 the route requires two redundant channels and the LLM supplies one. SR-06 (24-hour diagnostic self-test) is clean across every stage except *migration*. A legacy obligation refers to a diagnostic-timeout property name that the metamodel rename has retired, so the obligation was satisfied under the old name but is silently unsatisfiable under the new one, a silent regulated-domain drift that schema-only validators cannot phrase.

SR-05 is rejected on a different basis than SR-08. It names a hazard the corpus does not recognise, so *conformance* raises a non-fatal no-hazard flag and *anchor* finds no source artefact the claim traces back to (a hard *unanchored* verdict). With no anchored element to realise, *refinement* and *roundtrip* return no verdict, and the routing layer places the claim outside-upper. SR-08 is convicted by four well-formed analyses; SR-05 has no evidentiary footing at all — the hallucination case, where the LLM introduces an entity the knowledge base does not contain.

SR-08 (out-of-scope pressure trip with conflicting SIL) is the richest failure. Four of the five per-claim oracles converge on rejection, *migration* alone staying clean. *Conformance* detects a SIL clash (the LLM claims SIL-4 while HLR-001 is allocated SIL-3, and the metamodel makes the two disjoint). *Anchor* reports a wrong-hazard binding (the overpressure hazard is registered but not in HLR-001's scope). *Refinement* reports incomplete sensor redundancy, and the *roundtrip* returns HLR-002 (the parent that actually frames overpressure

in the KB) rather than the claimed HLR-001. The matrix preserves the four sources and so the engineering signal about what *kind* of repair the response needs.

We exercised the framework on decompositions produced by ChatGPT, Claude, and Gemini for the same prompt (Table II). All three pass conformance and anchor cleanly — no SIL clashes, no unanchored claims, no out-of-scope hazards — and the framework flags the same architectural defects in every run: the single-channel cooling actuator trips *refinement* against the SIL-3 redundancy requirement, and the diagnostic-timeout migration drift fires on every diagnostic sub-requirement. At the verdict level Claude and Gemini are bit-identical; their differences (response-time budgets, coverage percentages, prose) sit below the typed-graph abstraction the framework reasons over. ChatGPT's eighth sub-requirement is an integrated wrapper listing every realised component; it trips the current whole-response redundancy aggregation, an encoding limitation a per-role refinement variant resolves within the same formalism.

The audit takes the per-claim verdict table as an attribute-value information system and tests which attribute subsets preserve the indiscernibility partition. The four condition attributes are the per-claim verdicts of four of the five framework analyses. SIL-match (*conformance*, comparing the LLM's claimed SIL against HLR-001's SIL allocation), hazard-in-scope (*anchor*, checking the claimed hazard against the parent-concern relation), hardware-sufficient (*refinement*, checking the realising hardware cardinality against the requirement), and roundtrip-ok (*roundtrip*). The fifth analysis, *migration*, is held out here as an informational drift flag consumed only by the routing layer.

On this corpus the audit finds two minimal reducts of size two: {hazard-in-scope, hardware-sufficient} and {SIL-match, hardware-sufficient}, each preserving the indiscernibility partition exactly. The audit yields two operational signals. (1) Per object, a sensitivity reading (the review placement of SR-04 and the rejection of SR-08 both survive every minimal reduct). (2) Per corpus, a coverage diagnostic (hazard-in-scope and SIL-match are empirically interchangeable on this decomposition, a statement about the discriminative variety of the corpus, not about the structural value of the oracles). A different target class would not admit the SIL-match-free reduct, so the audit is re-run on each new decomposition rather than treated as a licence to retire oracles.

In the matrix above the target class is the conjunction of the four condition attributes, so every indiscernibility class is homogeneous with respect to the target by construction and the Pawlak boundary is empty; the routing labels in the last row come from the operator's routing layer, not from a non-trivial rough-set approximation. The apparatus shifts into a diagnostic regime when the target is lifted off the condition attributes. Attach an external reviewer's accept/reject verdict to the same eight sub-requirements and re-bind the partition's target to "accepted"; suppose the reviewer accepts {SR-01, SR-02, SR-03, SR-04, SR-07} and rejects {SR-05, SR-06, SR-08}, with SR-06 rejected because the obligation lost under

TABLE I
THE DIAGNOSTIC MATRIX PRODUCED BY THE PIPELINE ON THE LLM-DECOMPOSED SAFETY REQUIREMENT.

Stage	SR-01	SR-02	SR-03	SR-04	SR-05	SR-06	SR-07	SR-08
01 conformance	✓	✓	✓	✓	△ no-hazard	✓	✓	× SIL clash
02 anchor	✓	✓	✓	✓	× unanchored	✓	✓	△ wrong-hazard
03 migration	✓	✓	✓	✓	✓	△ drift	✓	✓
04 refine	✓	✓	✓	△ 1-of-2	—	✓	✓	△ 1-of-2
05 roundtrip	✓	✓	✓	✓	—	✓	✓	× wrong-parent
06 routing	lower	lower	lower	review	outside-upper	review	lower	outside-upper

TABLE II
CROSS-LLM DECOMPOSITION OF HLR-001 UNDER THE SAME PROMPT.

LLM	#SR	conf./anchor	refine	migration
ChatGPT	8	clean	1-of-2	diag. drift
Claude	7	clean	1-of-2	diag. drift
Gemini	7	clean	1-of-2	diag. drift

migration is material to the safety argument. The all-“yes” indiscernibility class now contains four accepted and one rejected element, so the Pawlak boundary becomes non-empty. Adding the migration verdict as a fifth condition attribute is exactly what splits the problematic class along the verdict line and restores a deterministic partition. The same partition that delivered a routing label in the descriptive regime here delivers an attribute-adequacy diagnosis: the present condition set is missing exactly the information the migration stage already carries. No new analysis is authored; the same formalism re-binds to a new target and reports the diagnosis.

VI. FUTURE WORK

The framework is a work in progress. The generic validation patterns formulated in the ASP metaprogramming style, together with an experimental domain binding, indicate that the approach is promising and admits clean extension without breaking the core. However, several directions remain worth investigating.

The completeness gap is closed by reusing the existing target-class machinery. Support-dominated prompts, where the request does not declare a closed target, leave this analysis silent. The next instantiation is a domain whose prompt itself carries a KB-enumerable target (a coverage, checklist, or enumeration task in which the engineer’s ground truth names what the LLM should have produced) where the resulting per-target verdict slots into the same diagnostic matrix the support analyses populate.

The framework treats the LLM prompt as an opaque boundary condition. It controls which KB facts enter the LLM’s context but does not itself enter the apparatus. Lifting the prompt into the shared vocabulary on the same footing as the response and the metamodel makes it a third instantiation of the same modelling discipline, recording which KB elements were injected as context, which validation obligations the prompt declares, and which target classes scope the response. This way, the validation becomes *prompt-declared* as the

analyses applied are those the prompt itself authorises, on the model of machine-readable tool schemas. From the completeness view, it grades the response against the KB subset the prompt addressed, so out-of-scope KB elements stop firing spurious violations. And the framework’s indiscernibility criterion turns outward into a prompt-design discipline. A prompt is well-formed for verification when its declared requirements uniquely characterise the correct answer against the KB, with asymmetric tests on response functions (cardinality, structural invariants).

A more general framing unifies the apparatus. The relation between the ideal answer and the actual one ranges over a five-valued domain (exact match, under-coverage, over-coverage, partial overlap, and disjointness), and every pipeline stage contributes a propagation rule over it: the LLM’s intrinsic error distribution, the prompt’s reproduction or distortion of the intended question, the response’s inheritance of both, KB validation as filter, and fault-tolerant orchestration patterns (such as N-version redundancy with common and independent inputs), each under its own composition rules. The analyses read as projections of this propagation approach, and the broader catalogue of error-propagation analysis carries over from classical dependable-systems engineering. The framework’s rough-set apparatus already pre-figures the algebra. Under a target-independent binding (as in the retrospective extension of the case study), the per-element *lower/boundary/outside-upper* approximation is the natural carrier for the under-coverage, partial-overlap, and disjointness distinctions on the same vocabulary the per-claim oracles emit, and reduct enumeration answers the dual question of how few requirements suffice for unambiguous discrimination against the knowledge base.

Two breadth directions remain open. Across domains, the approach is worth trying on additional safety and assurance domains (e.g., ISO 26262 (automotive), DO-178C (avionics), or IEC 62304 (medical software)), all of which carry decomposition, allocation, and refinement obligations the analyses can encode, only the per-domain bridge is re-authored, and the core analyses and formalism carry across. Within the demonstrated domain, other question types over the same metamodel and knowledge base are also worth exercising to broaden the empirical evidence of the analyses’ coverage.

The same indiscernibility-partition machinery that produces the compliance verdict can compare verdicts across LLMs on the same prompt. Two responses lift into the shared vocabulary

as two columns of one decision table, and the partition emits *lower* (both LLMs agree), *boundary* (only one supports the claim), or *outside* (LLM-specific) on each claim. The target-class parameter already hosts this comparison without modification.

The current analyses are hand-authored rules over the shared vocabulary. The same formalism admits their inductive synthesis. Inductive learning of ASP programs learns rules from labelled examples within an engineer-bounded hypothesis space. Because the learner emits rules over the same vocabulary, an induced analysis composes with the hand-authored ones without integration glue. Folding this synthesis loop into the framework lets the apparatus grow with a labelled corpus of reviewed responses rather than only with hand-authored rules.

GENERATIVE AI DISCLOSURE STATEMENT

During the preparation of this work, the authors used Claude to assist with drafting and refining manuscript text. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

REFERENCES

- [1] V. Lifschitz, *Answer Set Programming*. Berlin, Heidelberg: Springer-Verlag, 2019.
- [2] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, “Multi-shot ASP solving with clingo,” *Theory and Practice of Logic Programming*, vol. 19, no. 1, pp. 27–82, 2019. [Online]. Available: <https://doi.org/10.1017/S1471068418000054>
- [3] D. Varró and A. Pataricza, “Generic and meta-transformations for model transformation engineering,” in *UML 2004: The Unified Modeling Language*, ser. LNCS, vol. 3273. Springer, 2004, pp. 290–304.
- [4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, and Applications*, 2nd ed. Cambridge University Press, 2007.
- [5] H. Knublauch and D. Kontokostas, “Shapes constraint language (SHACL),” W3C Recommendation, 2017. [Online]. Available: <https://www.w3.org/TR/shacl/>
- [6] Z. Pawlak, “Rough sets,” *International journal of computer & information sciences*, vol. 11, no. 5, pp. 341–356, 1982.
- [7] Y. Yao, “Three-way decisions with probabilistic rough sets,” *Information Sciences*, vol. 180, no. 3, pp. 341–353, 2010.
- [8] A. Campagner and D. Ciucci, “Three-way learnability: A learning theoretic perspective on three-way decision,” in *Proceedings of the 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*, ser. Annals of Computer Science and Information Systems, vol. 30, 2022, pp. 243–246.
- [9] A. Skowron and C. Rauszer, “The discernibility matrices and functions in information systems,” in *Intelligent Decision Support*, R. Słowiński, Ed. Kluwer, 1992, pp. 331–362.
- [10] S. Emanuilov and R. Ackermann, “Stemming hallucination in language models using a licensing oracle,” *arXiv preprint arXiv:2511.06073*, 2025. [Online]. Available: <https://arxiv.org/abs/2511.06073>
- [11] N. Fathallah, A. Das, S. De Giorgis, A. Poltronieri, P. Haase, and L. Kovriguina, “NeOn-GPT: A large language model-powered pipeline for ontology learning,” in *The Semantic Web: ESWC 2024 Satellite Events*, ser. LNCS, vol. 15344. Springer, 2024, pp. 36–50. [Online]. Available: https://doi.org/10.1007/978-3-031-78952-6_4
- [12] D. Adam and T. Kliegr, “Traceable LLM-based validation of statements in knowledge graphs,” *Information Processing & Management*, 2025, arXiv:2409.07507. [Online]. Available: <https://arxiv.org/abs/2409.07507>
- [13] S. Miya, “Eidoku: A neuro-symbolic verification gate for LLM reasoning via structural constraint satisfaction,” *arXiv preprint arXiv:2512.20664*, 2025. [Online]. Available: <https://arxiv.org/abs/2512.20664>
- [14] A. G. Regino and J. C. dos Reis, “Can LLMs be knowledge graph curators for validating triple insertions?” in *Workshop on Knowledge Graph Curation*, 2025.
- [15] S. Alotaibi, A. Lisitsa, A. McCabe, and J. MacSween, “Leveraging large language models for automated export control screening: Evaluating llms framework,” in *Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS)*, ser. Annals of Computer Science and Information Systems, vol. 43, 2025, pp. 441–449.
- [16] Object Management Group, “Systems Modeling Language (SysML®) version 2.0 — final specification,” 2025. [Online]. Available: <https://www.omg.org/spec/SysML/>