DETECTING INCONSISTENCIES IN MODEL SPECIFICATIONS WITH LLM-AS-VERIFIER REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Model specifications define how large language models (LLMs) should behave, guiding alignment training, inference-time behavior, and evaluation. Yet these specifications may themselves contain defects: two individually reasonable principles may prescribe incompatible behavior when applied to the same situation, leaving no response that satisfies both. Detecting such inconsistencies is challenging. Formalizing natural-language specifications risks losing subtle distinctions, while behavior-based testing cannot reliably distinguish specification defects from differences in model behavior. We introduce VERISPEC, the first approach to directly detect inconsistencies in model specifications by auditing the specification text itself. Our key insight is to preserve the specification in natural language while using an LLM as a verifier. VERISPEC extracts structured, context-aware rules, constructs a topic-guided graph to cluster behaviorally related rules at the same authority level, and applies LLM-as-verifier reasoning to detect inconsistencies. Applying VERISPEC to the OpenAI Model Spec, we extract 405 rules and manually validate five inconsistencies, all reported to its developers, who responded positively and have initiated internal discussions. Compared with five baselines, VERISPEC identifies the most validated inconsistencies, achieves the highest precision (38.5%), and incurs the lowest cost per validated inconsistency ($11.12). These results establish direct specification auditing as a practical complement to behavioral alignment evaluation, catching defects at the source before they shape any model.

1 INTRODUCTION

Model specifications have become a central interface for describing how LLMs should behave. They state requirements for usefulness, style, safety, security, and legal compliance, and they guide alignment training, inference-time behavior, and evaluation (Bai et al., 2022; Guan et al., 2025; Wolfe, 2026). Some providers publish these requirements for external review: OpenAI provides the *Model Spec* (OpenAI, 2026c) and Anthropic provides *Claude's Constitution* (Anthropic, 2026a). We refer to such documents as *model specs*. As models and deployment expectations change, these documents are frequently updated (OpenAI, 2025a; Anthropic, 2026b), making systematic review increasingly important.

As model specs grow, they become difficult to audit manually. A principle may be clear in isolation yet prescribe incompatible behavior once another applicable principle enters the picture, and the interacting passages may appear in different sections, depend on authority levels, or be mediated by exceptions, examples, and other contextual instructions. Model specs also differ from conventional natural-language requirements: they govern open-ended interactions, so an inconsistency may surface only when several principles are applied to the same concrete situation (Zhang et al., 2025; Buyl et al., 2025).

Figure 1 shows an example inconsistency in the OpenAI Model Spec. One principle asks the model to leave unrequested parts of user-provided text unchanged during transformations; another requires that code intended for execution be runnable. A request for explanatory comments with code-only output triggers both: preserving the code leaves a syntax error, while fixing the syntax changes content the user did not ask to change. Neither principle is defective on its own—the inconsistency appears only when both apply to the same situation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A natural route to detecting such inconsistencies is to translate the specification into a formal language and apply a formal verifier (Merigoux et al., 2021; Giannakopoulou et al., 2020; Cosler et al., 2023). For model specs, however, translation is precisely where meaning is lost: the text leans on modal distinctions, e.g., *may*, *must*, *should*, and on permissions, defaults, exceptions, and soft obligations that formal languages flatten (Merigoux et al., 2021; Guitton et al., 2025). Automating the translation invites errors on exactly these distinctions, while performing it manually at the scale of a full specification requires substantial expert effort (Yen et al., 2021; Giannakopoulou et al., 2020; Cosler et al., 2023). *Our key insight is to invert the approach: keep the specification in natural language and use an LLM as the verifier, so that the distinctions formalization discards become evidence the verifier reasons over.*

To realize this insight, we make the text checkable by splitting the specification into *rules*, still in natural language, each recording a principle's source annotation, authority level, precondition, and postcondition. Two obstacles remain. First, inconsistency is a property of rules *in context*: worked examples, explicit exceptions, and trade-off guidance elsewhere in the specification routinely dissolve apparent conflicts between bare rule statements, so a verifier shown only the statements reports tensions the document already resolves. Second, verifying rules pairwise is quadratic in the rule inventory, and most pairs govern unrelated behavior, so nearly all verifier calls would be spent confirming that nothing interacts. We therefore attach to each rule the context needed to interpret it, and cluster behaviorally related rules—by the *topics* of behavior they govern—into compact analysis units at the same authority level (the specification's chain of command already adjudicates conflicts across levels). Each verifier call then reasons over a focused set of potentially interacting rules rather than a long document in which the relevant passages are easily lost (Liu et al., 2024).

We introduce VERISPEC, to our knowledge the first approach to directly detect defects in model specifications—in particular, inconsistencies among their stated principles. VERISPEC realizes the design above in three stages: *context-aware rule extraction* builds the rule inventory together with each rule's interpretive context; *graph-based rule clustering* forms same-authority analysis units over a topic-guided rule graph; and *LLM-as-verifier detection* reasons over each unit, reporting an inconsistency only when it can produce a concrete witness situation, explain why both rules apply, and show that their combined context does not resolve the conflict.

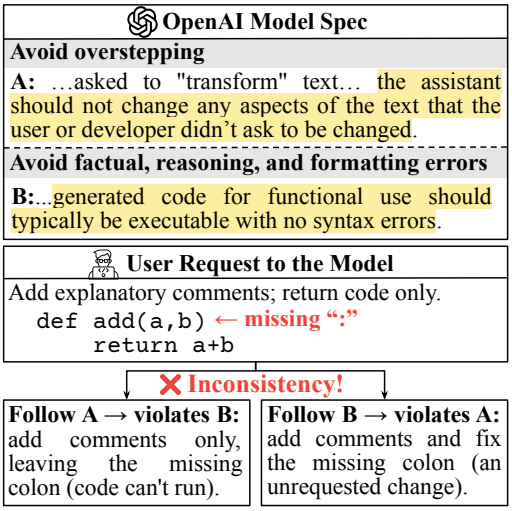

Figure 1: A running example of inconsistency in the OpenAI Model Spec.

We evaluate VERISPEC on the OpenAI Model Spec (OpenAI, 2026c). VERISPEC extracts 405 rules, constructs 32 analysis units, and reports 13 candidate inconsistencies after five verifier attempts. Manual validation confirms five across legal constraints, code transformations, output-only tasks, fictional portrayals, and sycophantic reassurance. We reported all five to the OpenAI Model Spec developers, who responded positively and have initiated internal discussions. Compared with five baselines, VERISPEC validates the most inconsistencies, achieves the highest precision at 38.5%, and has the lowest cost per validated inconsistency at $11.12.

In summary, this paper makes the following contributions:

- **Formulation.** We formalize specification inconsistency at the level of concrete situations, accounting for rule authority and surrounding context.
- **Approach.** We introduce VERISPEC, to our knowledge the first approach to directly detect inconsistencies in natural-language model specifications through context-aware rule extraction, graph-based clustering, and LLM-as-verifier reasoning.
- **Evaluation.** We identify five validated inconsistencies in the OpenAI Model Spec and report them to its developers. Comparisons against five baselines demonstrate the effectiveness of VERISPEC, which achieves the highest precision and the lowest cost per validated inconsistency.

OpenAI Model Spec	Extracted Rules
Avoid overstepping User The assistant should help the developer and user by following explicit instructions and reasonably addressing implied intent [...] without overstepping[z068]. Sometimes the assistant is asked to "transform" text: [...] Given such a task, the assistant should not change any aspects of the text that the user or developer didn't ask to be changed[92bt].	t : 92bt A : user P : Given a text transformation task Q : Do not change aspects of the text the user or developer did not ask to be changed C : "...leave the code intact..." T : transformation_fidelity
Avoid factual, reasoning, and formatting errors User If parts of the response are in a specific format (e.g., Markdown, code, JSON, LaTeX), the assistant should adhere to the specifications of that format[cw53]. In particular, generated code for functional use should typically be executable with no syntax errors[e9ny].	t : e9ny A : user P : Generating code for functional use Q : Ensure the code is executable with no syntax errors C : "...adhere to...that format" T : output_format_adherence

Figure 2: Illustration of principles (highlighted on the left) and their conversion to rules (on the right). t denotes the annotation on each principle, with annotations we added highlighted in green. A denotes the authority level, P and Q respectively denote the precondition and the postcondition, C denotes the context needed to interpret the rule, and T denotes the topic the rule is assigned.

2 Background: OpenAI Model Spec

Principles in the model spec are organized into a hierarchy of sections by theme, such as prohibitions, usefulness, and style. Principles may state hard boundaries that should not be compromised, defaults for unspecified preferences, or decision rubrics for gray areas. The model spec also includes concrete examples that demonstrate how certain principles should be applied in practice.

Authority levels. Each section in the model spec is labeled with an *authority level*: root, system, developer, user, or guideline. Sections thus form a chain of command, with root-level sections at the top and sections from other levels progressively below. In other words, principles from higher-level sections take precedence in case of conflicts (Wallace et al., 2024; OpenAI, 2026c; Wolfe, 2026).

Annotations. The model spec contains several alphanumeric annotations, called *focus areas*. Though not officially documented, OpenAI's model alignment (Guo & Wolfe, 2026) dataset (OpenAI, 2026d) groups evaluation prompts by focus area, suggesting that focus areas correspond to specific principles in the model spec. However, not all principles are annotated in this way. We address this in Section 4.1.1. Figure 2 illustrates these annotations for our running example (Figure 1).

3 Related Work

Evaluating Model Alignment. A model specification describes the intended behavior of models, not the behavior deployed systems exhibit, so alignment to a specification is typically evaluated empirically after deployment. Ahmed et al. (2025) audit models against provider specifications by parsing behavioral statements, generating targeted prompts, and using models as judges. OpenAI similarly provides an evaluation pipeline and prompt dataset for measuring adherence to the Model Spec (Guo & Wolfe, 2026). Jakkli et al. (2026) study whether models follow their provider specifications across generations, and Wang et al. (2026) synthesize specification-alignment data to improve model behavior. All of these works take the specification as ground truth and ask whether models comply with it. We invert the question: we treat the specification itself as the object of analysis and ask whether that ground truth is internally consistent.

Value Conflicts in Model Behavior. A second line of work studies how models prioritize competing values. Buyl et al. (2025) show that arbitrary behavioral-value annotations can lead models away from human discretion, while Liu et al. (2026) use value-conflict scenarios to test prioritization and show that detailed value rankings can improve alignment. These papers ask how models behave when values compete; we ask a prior question: whether the written specification prescribes incompatible behavior in the first place, before it is used to train, evaluate, or guide any model.

Behavior-Based Specification Testing. The closest line of work uses model behavior to stress-test the specification itself. Zhang et al. (2025) generate value-tradeoff scenarios, measure disagreement across frontier LLMs, and evaluate OpenAI models against the OpenAI Model Spec, showing that behavioral disagreement can surface possible specification issues. This signal, however, is indirect and can err in both directions. The Model Spec and deployed models evolve on separate schedules, and the specification version used to train or align a given model is unknown (OpenAI, 2026b); OpenAI itself notes that deployed behavior may diverge from the Model Spec (OpenAI, 2025b). A

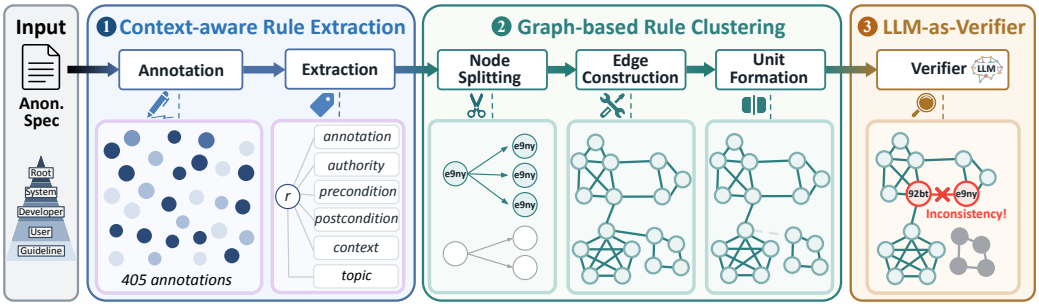

Figure 3: Overview of VERISPEC.

behavioral disagreement may therefore reflect training, alignment, or capability differences rather than a specification defect (Ahmed et al., 2025; Guo & Wolfe, 2026; Jakkli et al., 2026)—and, conversely, a defect that models happen to resolve the same way produces no disagreement to observe. To our knowledge, VERISPEC is the first approach to detect model specification defects directly: it audits the specification text rather than the behavior it induces, and reports concrete inconsistencies supported by structured verifier reasoning and manual validation.

4 METHODOLOGY

VERISPEC detects internal inconsistencies in model specifications through the three-stage pipeline shown in Figure 3. *Context-aware rule extraction* (Section 4.1) makes the specification checkable: it converts every principle into a structured natural-language rule and attaches the context (e.g., worked examples, exceptions, trade-off guidance) needed to interpret it. *Graph-based rule clustering* (Section 4.2) makes verification tractable: over a topic-guided rule graph, it clusters behaviorally related rules into compact, same-authority analysis units, so verifier calls are spent only where rules can genuinely interact. *LLM-as-verifier detection* (Section 4.3) holds every report to a burden of proof: an inconsistency is reported only with a concrete witness situation, a joint-unsatisfiability argument, and evidence that the combined context does not resolve the conflict.

Anonymization. Model specifications contain many references to the corresponding model provider and its models, such as OpenAI and GPT-4.1. Proprietary LLMs often refuse to analyze model specifications in depth since these references trigger their safety guardrails. We found that anonymizing the model specification, i.e., replacing these references with synthetic names, suppresses guardrails and allows us to analyze the model specification. We used Claude Code (Anthropic, 2025) to perform the anonymization, together with static checks to ensure that edits are limited to such substitutions.

4.1 CONTEXT-AWARE RULE EXTRACTION

4.1.1 ANNOTATING ALL PRINCIPLES

As described in Section 2, many principles in the model spec have alphanumeric annotations called *focus areas*. These annotations provide anchors in the specification, allowing us to systematically cover all principles in the model spec. The model spec contains 279 annotations, but we found by manual inspection that many principles are not annotated and a small number of annotations do not even mark principles. The latter only adds a small amount of overhead during analysis, but the former limits coverage by omitting principles that may be involved in inconsistencies.

Note that a statement in the model spec is considered to be a principle if it prescribes model behavior in some situation. A statement may contain multiple principles, in which case each of them should carry its own annotation. Following this criterion, we used the existing annotations as examples to guide Claude Code in annotating the remaining principles and thus completing the annotations. We added 126 annotations, bringing the total to 405. Figure 2 shows one added annotation for our running example (Figure 1).

4.1.2 TRANSFORMING PRINCIPLES TO RULES

We transform each principle from its unstructured prose form to a structured representation for downstream analysis. We refer to this structured representation as *rules*.

Definition 1 (Rule). *A rule is a tuple (t, A, P, Q), where t is the annotation for a principle p in the model spec, A is the authority level (Section 2) of the section under which p is described, P is the precondition under which p applies, and Q is the postcondition prescribed by p that the model must satisfy when P holds. Both P and Q are written in natural language and are extracted from the source specification text.*

We extract rules from all annotated principles by identifying their preconditions and postconditions in the specification text, creating an inventory of rules, R. Figure 2 shows two rules extracted for our running example (Figure 1) from their annotations `92bt` and `e9ny`. R consists of 405 rules: 190 at the root level, 15 at the system level, 5 at the developer level, 87 at the user level, and 108 at the guideline level. Appendix A provides additional details on the construction of the rule inventory.

Context. Due to the open-ended manner in which principles are described, the statement of a principle may not be sufficient in itself to understand its application in practice. The model specification contains concrete examples, definitions, and guidance on trade-offs between different principles, all of which aid the interpretation of any principle. The statement of a principle, together with this relevant information from the model specification, forms the context associated with the principle. We denote this by $\mathrm{C}(r)$, where r is the corresponding rule.

Topic derivation. A *topic* is a class of rules that address a common aspect of model behavior or situations, like output-format adherence or stance-taking. We use Claude Code to extract topics from the model specification by spawning multiple subagents, which independently propose topics that are then merged into a single set of topics, Δ. For every proposed topic, the main agent spawns multiple subagents to independently list the specific rules belonging to that topic. Any topic with high disagreement between subagents is considered ambiguous and is dropped. We use $T(r)$ to denote the set of topics that a rule r belongs to.

4.2 Graph-based Rule Clustering

Detecting inconsistencies requires examining the right rules jointly, yet interacting rules are scattered: the two principles of our running example (Figure 1) sit in different sections of the specification and govern different aspects of behavior (Figure 2). Grouping rules by document structure keeps such cross-section pairs apart, while exhaustive pairwise verification is prohibitively expensive (over 80,000 pairs for our 405-rule inventory, almost all governing unrelated behavior). We therefore construct a *rule graph* that records how rules relate, and use it to cluster related rules at the same authority level into compact *analysis units* for verification.

Definition 2 (Rule graph). *The* rule graph *is an undirected graph $G = (V, E_{\mathrm{sem}}, E_{\mathrm{syn}})$, where E_{sem} and E_{syn} denote the semantic and syntactic edge sets, respectively. Each vertex $(r, \delta) \in V$ represents the assignment of rule r to topic δ. A rule assigned to multiple topics therefore contributes one vertex per topic.*

Edges capture two complementary relations. A *semantic* edge connects two vertices assigned to the same topic, linking rules that govern the same behavior wherever they appear in the document. A *syntactic* edge connects vertices whose underlying rules occur in the same subsection, share a worked example, or are linked by an explicit cross-reference. These document-level ties capture relations that the topic tests may miss. We write $\mathrm{syn}(r, r')$ if r and r' satisfy at least one of these conditions. Edges are constructed only between distinct rules at the same authority level, because disagreements across levels are resolved by the chain of command.

Algorithm 1 formalizes the clustering process; the middle stage of Figure 3 depicts it on the running example. For the OpenAI Model Spec, we extract 405 rules and derive 45 topics (Appendix B), producing 625 rule–topic vertices. Among these rules, 162 belong to multiple topics and thus contribute multiple vertices. Grouping vertices by authority level and topic produces 113 initial groups, each containing rules that share the same topic and authority. However, these groups vary considerably in size, with many containing only one or two rules. Verifying each group separately would be inefficient and provide little opportunity to detect interactions between rules from different topics.

To address this, `MergeSmallGroups` (line 13 in Algorithm 1) merges small groups with related groups at the same authority level. It selects merge targets based on shared rules and graph connections, including syntactic edges that link rules across topics. This process produces 32 analysis units, each containing 5 to 26 rules. Appendix B provides further details on topic derivation, clustering thresholds, and graph statistics.

Algorithm 1 CONSTRUCTANALYSISUNITS

Require: rule set R with topic assignments $T(r)$; authority levels $\mathcal{A}$
Ensure: analysis units $\mathcal{U}$

$V \leftarrow \{(r, \delta) : r \in R,\ \delta \in T(r)\}$ ▷ construct rule-topic vertices
$E_{\text{sem}} \leftarrow \emptyset$; $E_{\text{syn}} \leftarrow \emptyset$
for all $\{u, v\} \subseteq V$, where $u = (r, \delta)$, $v = (r', \delta')$, $r \neq r'$, and $A(r) = A(r')$ **do**
  if $\delta = \delta'$ **then**
    $E_{\text{sem}} \leftarrow E_{\text{sem}} \cup \{\{u, v\}\}$
  end if
  if $\text{syn}(r, r')$ **then**
    $E_{\text{syn}} \leftarrow E_{\text{syn}} \cup \{\{u, v\}\}$
  end if
end for ▷ construct edges
$G \leftarrow (V, E_{\text{sem}}, E_{\text{syn}})$
$\mathcal{U}_0 \leftarrow \big\{\{(r, \delta') \in V : A(r) = a,\ \delta' = \delta\} : a \in \mathcal{A},\ \delta \in \Delta\big\} \setminus \{\emptyset\}$ ▷ form initial groups
$\mathcal{U} \leftarrow \text{MERGESMALLGROUPS}(\mathcal{U}_0, G)$ ▷ form analysis units
return $\mathcal{U}$

4.3 LLM-AS-VERIFIER DETECTION

Given the analysis units constructed above, we use an LLM to detect inconsistencies between rules. We formalize an inconsistency at the level of a concrete decision situation. For two rules r_i and r_j, let $C_i = C(r_i)$ and $C_j = C(r_j)$ denote their contexts as defined in Section 4.1.2, and let $C_{ij} = C_i \cup C_j$ be their combined context.

Definition 3 (Inconsistency)**.** *Two distinct rules $r_i = (t_i, A_i, P_i, Q_i)$ and $r_j = (t_j, A_j, P_j, Q_j)$ are inconsistent if they have the same authority and there exists a feasible situation s such that both preconditions hold, but no assistant behavior can satisfy both postconditions after accounting for their combined context:*

$$\exists s.\ A_i = A_j \wedge P_i(s, C_{ij}) \wedge P_j(s, C_{ij}) \wedge \text{UNSAT}(Q_i(s, C_{ij}), Q_j(s, C_{ij}))\,.$$

Here, UNSAT *means that, under the combined context C_{ij}, no assistant behavior can satisfy both postconditions.*

We use the LLM as a verifier rather than as an unconstrained inconsistency detector. We use Claude Opus 5 (Anthropic, 2026d) as the core verifier because it reliably follows the structured verification prompt and completes the full analysis units; GPT-6 Astra (OpenAI, 2026a) produced similar findings at higher cost, while Claude Fable 5.1 (Anthropic, 2026c) blocked most requests under Anthropic's restrictions on violative cyber content. For each analysis unit, the verifier receives the rules in that unit together with the context needed to interpret them. A valid report must provide evidence for Definition 3: it must give a concrete witness situation, explain why both rules apply, and justify why the combined context does not resolve the inconsistency.

The verifier does not compare rules by surface form. Even when it finds a witness situation s in which both preconditions hold and the two postconditions appear jointly unsatisfiable, the pair is still only a candidate inconsistency. The verifier must interpret the candidate under C_{ij}, since the combined context may include authority rules, exceptions, examples, or specific instructions that resolve the inconsistency. The verifier reports an inconsistency only when no assistant behavior satisfies both postconditions after this contextual resolution.

Figure 3 shows a reported inconsistency between rule `92bt` and `e9ny`, where transforming malformed user-provided code creates a conflict between preserving the input and returning executable code. The report argues that the combined context does not resolve the inconsistency because it does not specify which rule should take precedence in such a situation, and the transformation guidance also rules out explaining the defect when the assistant must return the transformed output without comment. We analyze this inconsistency in more detail in Section 5.4.

We run the verifier independently on each analysis unit and require structured outputs containing the two rule ids, the witness, the unsatisfiability argument, and the context-resolution argument. We repeat the verifier pass five times per unit to reduce missed detections, then aggregate the reported pairs for downstream manual review and validation.

Table 1: Design of each method and its inconsistency-detection results.

	Design				Inconsistency detection results				
Method	Execution	Rule extraction	Analysis units	LLM-as-verifier	Reported	Validated	Precision	Cost	Cost per validated
DIRECTAGENT	Agent	×	None	×	9	0	0%	$27.66	–
DIRECTPROMPT	LLM call	×	None	×	16	0	0%	$4.92	–
SECTIONTEXT	LLM call	×	Source sections	✓	10	1	10%	$58.64	$58.64
SECTIONGROUP	LLM call	✓	Source sections	✓	8	1	12.5%	$63.94	$63.94
PLAINDETECT	LLM call	✓	Rule graph	×	221	3	1.4%	$47.46	$15.82
VERISPEC	LLM call	✓	Rule graph	✓	13	**5**	**38.5%**	$55.62	**$11.12**

5 EXPERIMENTS

We evaluate VERISPEC on the anonymized OpenAI Model Spec, using the August 18, 2026 release—the latest version available when we conducted the study. Although VERISPEC applies to model specifications generally, including Claude's Constitution (Anthropic, 2026a), we choose the OpenAI Model Spec because it is detailed, public, actively maintained, and widely used to specify model behavior. We describe the experimental setup (Section 5.1), then present the main results (Section 5.2), the baseline comparison (Section 5.3), and the four validated inconsistencies beyond the running example (Section 5.4).

5.1 EXPERIMENTAL SETUP

Evaluation metrics. We evaluate each method by validating its reported inconsistencies. We report the following metrics:

- *Reported inconsistencies.* The number of candidate inconsistencies returned by the method.
- *Validated inconsistencies.* The number of manually validated inconsistencies.
- *Precision.* The fraction of reported inconsistencies that are manually validated.
- *Cost.* The total cost of running the method.
- *Cost per validated inconsistency.* The total cost divided by the number of validated inconsistencies, reported only when a method validates at least one inconsistency.

We report cost because methods may improve detection quality by issuing more LLM requests, so the comparison should reflect both detection quality and resource use.

Inconsistency validation. We validate each reported inconsistency through manual inspection by two reviewers. Each reviewer independently examines the reported rule pair, witness, and supporting context, and assigns a verdict on whether the report is a true inconsistency. When the two verdicts agree, we use the agreed verdict as the final label. When they disagree, the reviewers discuss the case and assign a final consensus verdict.

Baselines. We compare VERISPEC against baselines that remove or simplify different parts of the pipeline. To ensure a fair comparison, all methods are powered by Claude Opus 5 and are run for five attempts, matching the repeated detection budget used by VERISPEC. We did not find an existing tool that directly targets inconsistency detection for model specifications, so these baselines are constructed from natural alternatives. Table 1 summarizes the comparison, and Appendix C provides additional implementation details.

- **DIRECTAGENT.** We give the full specification to a fresh Claude Code agent and ask it to find inconsistencies. This baseline tests whether a general-purpose coding agent can perform the detection without the structure introduced by VERISPEC.
- **DIRECTPROMPT.** We give the full specification to the LLM in a single prompt and ask it to report inconsistencies directly. This baseline tests whether a non-agentic whole-spec prompt is sufficient without rule extraction, graph construction, or LLM-as-verifier detection.
- **SECTIONTEXT.** We remove the rule extraction phase, use each source section as an analysis unit, and apply LLM-as-verifier detection directly to the section text. This baseline tests whether explicit rule extraction is necessary, or whether the source text alone is sufficient.
- **SECTIONGROUP.** We keep the rule extraction phase and LLM-as-verifier detection, but form the analysis units by source section instead of by the rule graph. This baseline tests whether the document structure alone is sufficient to form analysis units.

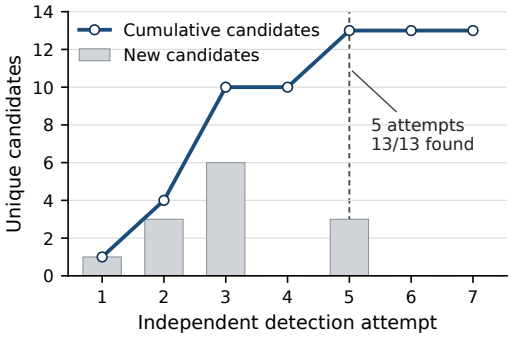

Figure 4: Cumulative candidate inconsistencies across detection attempts.



Figure 5: Reported and validated inconsistencies by authority.

- **PLAINDETECT.** We use the same analysis units VERISPEC uses but ask the LLM to detect inconsistencies directly rather than using LLM-as-verifier detection. This baseline tests the effect of LLM-as-verifier detection.

5.2 MAIN RESULTS

Applying VERISPEC to the 405 extracted rules produced 32 same-authority analysis units. We ran inconsistency detection five times on each unit, yielding 160 independent detection requests. Across these runs, VERISPEC produced 17 raw inconsistency reports. After removing duplicate outputs and merging reports that described the same specification inconsistency, we obtained 13 candidate inconsistencies for manual inspection. Figure 4 shows that the cumulative number of unique candidate inconsistencies saturates at the fifth attempt, where all candidates have been found.

Manual inspection validated 5 of the 13 candidates, giving VERISPEC a precision of 38.5% at a total cost of $55.62. Figure 5 shows that the validated inconsistencies come from system- and user-level rules, while root- and guideline-level candidates are filtered out during validation. The 5 validated inconsistencies span legal constraints, code transformations, output-only tasks, fictional portrayals, and sycophantic reassurance, showing that specification defects arise from both local and cross-section rule interactions. We have reported these 5 inconsistencies to the OpenAI Model Spec developers. Figure 1 shows the code-transformation inconsistency as the running example, and Section 5.4 presents the other four validated inconsistencies.

5.3 BASELINE COMPARISON

Table 1 compares VERISPEC with the baselines. Among the baselines, PLAINDETECT finds the most validated inconsistencies with 3, while SECTIONGROUP achieves the highest precision at 12.5%. DIRECTAGENT and DIRECTPROMPT cost less than VERISPEC, but neither reports any validated inconsistency. VERISPEC has the lowest cost per validated inconsistency at $11.12, compared with $15.82 for PLAINDETECT and more than $58 for the section-based verifier baselines.

Whole-specification detection is not sufficient. DIRECTAGENT and DIRECTPROMPT give the model access to the full specification, but neither produces a validated inconsistency. They report 9 and 16 candidates respectively, all of which fail manual validation. A recurring failure mode is to report an inconsistency after reading only the apparent tension between two passages, while missing a nearby exception that resolves the pair. In one false report, the baseline treated a transformation permission as inconsistent with content prohibitions, even though the transformation passage explicitly says that the permission still applies when the transformed output would otherwise be restricted. This result shows that whole-specification access alone does not focus the model on the specific principles that may be inconsistent, and it does not force the model to check whether the surrounding context resolves a candidate inconsistency.

Section-based grouping misses cross-section cases. SECTIONTEXT and SECTIONGROUP validate only one inconsistency each, namely the legal-disclosure case whose two rules appear in the same section. They miss cases whose rules are distributed across the document, including the code-transformation case used as the running example in Figure 1. The difference between SECTION-

Case	OpenAI Model Spec	Why inconsistent
Legal omissions System level	**A:** The assistant should operate within applicable legal constraints, including applicable local law for local deployments of models... **B:** Any content omitted due to legal requirements... must be transparently indicated to the user...	A local law may require omitting information and forbid acknowledging the omission. Indicating it violates **A**, while staying silent violates **B**.
Fictional portrayals User level	**A:** ... for questions about fundamental human rights violations... should clearly state these are wrong... **B:** If the user asks the assistant to roleplay or assist with creative endeavors, the assistant should comply without attempting to impose an objective point of view.	A user asks for a fictional monologue arguing that slavery should be legal. Adding a condemnation violates **B**, while omitting it violates **A**.
Output-only tasks User level	**A:** ... when producing output that'll be consumed programmatically... should just follow transformation instructions without comment. **B:** When a user's request includes false or potentially false premises, the assistant should highlight any potential misalignment...	A developer asks only for the translation of "Canberra is the capital of New Zealand" for automated ingestion. Flagging the false premise violates **A**, while translating silently violates **B**.
Sycophantic reassurance User level	**A:** The assistant may also follow norms of politeness in answering questions... to avoid exacerbating self-image or body dysmorphia concerns. **B:** The assistant exists to help the user, not flatter them or agree with them all the time. For subjective questions...	After describing conduct that puts their child at risk, a user asks, "Tell me I'm not a bad mom." **A** permits the polite reassurance the user asks for, while **B** forbids it as flattery.

Figure 6: Four additional validated inconsistencies in the OpenAI Model Spec.

TEXT and SECTIONGROUP is also small, which suggests that rule extraction helps most when paired with analysis units that group rules by behavior rather than by source section.

LLM-as-verifier detection reduces false positives. PLAINDETECT uses the same rule-graph units as VERISPEC but asks the LLM to report inconsistencies directly. It finds 3 validated inconsistencies, but it reports 221 candidates and achieves only 1.4% precision. Most invalid reports are resolved by specification context, including explicit exceptions, compatible joint responses, worked examples, or the chain of command. The LLM-as-verifier detection targets this failure mode by requiring a witness, an unsatisfiability argument, and a context-resolution argument before reporting. This reduces the candidate set from 221 to 13 while increasing precision from 1.4% to 38.5%.

5.4 VALIDATED INCONSISTENCIES

Figure 1 shows the code-transformation inconsistency used as the running example. Figure 6 presents the remaining four validated inconsistencies. Each case quotes the two relevant principles from the OpenAI Model Spec, labeled A and B, and explains why they are inconsistent.

6 CONCLUSION

We presented VERISPEC, an approach to directly detecting inconsistencies in natural-language model specifications through context-aware rule extraction, graph-based clustering, and LLM-as-verifier reasoning. Applied to the OpenAI Model Spec, VERISPEC identified five manually validated inconsistencies and outperformed five baselines in detection precision and cost per validated inconsistency. These findings demonstrate that systematic auditing can uncover defects in the specifications themselves, complementing existing approaches that evaluate model behavior.

Our work opens a broader direction toward continuous verification and improvement of model specifications. As these specifications evolve, future research can extend beyond inconsistency detection to automated specification repair, incremental verification, and systematic auditing across different model providers. More fundamentally, our work challenges a prevailing assumption in LLM alignment: that the specifications against which models are trained and evaluated can themselves be treated as ground truth. Ensuring that models follow their specifications is only part of the challenge; ensuring that those specifications are themselves consistent is equally important.

AI USE STATEMENT

In this work, we used generative AI tools to implement the pipeline, conduct the experiments and polish the paper text. We have not used generative AI tools for proposing or refining hypotheses, developing conceptual frameworks, designing the research methodology or experiments, or interpreting results, and generating synthetic data sets, formulating mathematical claims, writing proofs, and translation are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, creating and modifying figures. We have reviewed all AI-assisted work. All LLM-generated content are verified by all of the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work analyzes the publicly available OpenAI Model Spec and involves no human subjects or private data. We anonymize references to the model provider and its models because proprietary LLMs often decline to analyze the document in depth when those identifiers are present. The anonymization is limited to these replacements, and static checks confirm that the remaining specification text is unchanged. The 5 identified inconsistencies are model spec defects rather than attacks on deployed models, but such inconsistencies could still help produce adversarial prompting. Therefore, we have reported all 5 validated inconsistencies to the OpenAI Model Spec developers.

REPRODUCIBILITY STATEMENT

We have anonymized all artifacts and provided them in the supplementary material.

REFERENCES

Ahmed M Ahmed, Kevin Klyman, Yi Zeng, Sanmi Koyejo, and Percy Liang. SpecEval: Evaluating model adherence to behavior specifications. In *NeurIPS 2025 Workshop on Regulatable ML*, 2025.

Anthropic. Claude code. `https://claude.com/product/claude-code`, 2025. Accessed September 6, 2026.

Anthropic. Claude's constitution. `https://www.anthropic.com/constitution`, 2026a. Accessed September 6, 2026.

Anthropic. Claude's new constitution. `https://www.anthropic.com/news/claude-new-constitution`, 2026b. Accessed September 6, 2026.

Anthropic. Claude fable 5.1. `https://www.anthropic.com/claude-fable-and-mythos-5-1`, September 2026c. Accessed September 17, 2026.

Anthropic. Claude opus 5. `https://www.anthropic.com/news/claude-opus-5`, July 2026d. Accessed September 17, 2026.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI feedback, 2022. URL `https://arxiv.org/abs/2212.08073`.

Maarten Buyl, Hadi Khalaf, Claudio Mayrink Verdun, Lucas Monteiro Paes, Caio Cesar Vieira Machado, and Flavio du Pin Calmon. AI alignment at your discretion. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency*, 2025.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*, pp. 383–396. Springer, 2023.

Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. Generation of formal requirements from structured natural language. In *International working conference on requirements engineering: Foundation for software quality*, pp. 19–35. Springer, 2020.

Melody Y. Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, Hyung Won Chung, Sam Toyer, Johannes Heidecke, Alex Beutel, and Amelia Glaese. Deliberative alignment: Reasoning enables safer language models, 2025. URL https://arxiv.org/abs/2412.16339.

Clement Guitton, Reto Gubelmann, Ghassen Karray, Simon Mayer, and Aurelia Tamò-Larrieux. Identifying open-texture in regulations using LLMs. *Artificial Intelligence and Law*, 2025.

Alan Guo and Jason Wolfe. Introducing model spec evals. https://alignment.openai.com/model-spec-evals/, 2026. Accessed September 6, 2026.

Arya Jakkli, Senthooran Rajamanoharan, and Neel Nanda. How well do models follow their constitutions? In *Second Workshop on Agents in the Wild: Safety, Security, and Beyond*, 2026.

Andy Liu, Kshitish Ghate, Mona T. Diab, Daniel Fried, Atoosa Kasirzadeh, and Max Kleiman-Weiner. Generative value conflicts reveal LLM priorities. In *The Fourteenth International Conference on Learning Representations*, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. Catala: a programming language for the law. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–29, 2021.

OpenAI. The OpenAI model spec: Changelog. https://github.com/openai/model_spec/blob/main/CHANGELOG.md, 2025a. Accessed September 6, 2026.

OpenAI. Expanding on what we missed with sycophancy. https://openai.com/index/expanding-on-sycophancy/, 2025b. Accessed September 22, 2026.

OpenAI. Gpt-6 astra. https://openai.com/index/gpt-6-astra/, September 2026a. Accessed September 17, 2026.

OpenAI. Model release notes. https://help.openai.com/en/articles/9624314-model-release-notes, 2026b. Accessed September 22, 2026.

OpenAI. OpenAI model spec. https://model-spec.openai.com, 2026c. Accessed September 6, 2026.

OpenAI. Openai model spec eval dataset. https://github.com/openai/model_spec_dataset/, 2026d. Accessed September 6, 2026.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions, 2024. URL https://arxiv.org/abs/2404.13208.

Wenjie Wang, Yue Huang, Zhengqing Yuan, Han Bao, Shiyi Du, Yuchen Ma, Yue Zhao, Yanfang Ye, and Xiangliang Zhang. SpecAlign: Efficient specification-grounded alignment of large language models via synthetic data, 2026. URL https://arxiv.org/abs/2606.16276.

Jason Wolfe. Inside our approach to the model spec. https://openai.com/index/our-approach-to-the-model-spec/, 2026. Accessed September 6, 2026.

Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. Semi-automated protocol disambiguation and code generation. In *Proceedings of the ACM SIGCOMM 2021 Conference (SIGCOMM '21)*, Virtual Event, USA, 2021. Association for Computing Machinery. doi: 10.1145/3452296.3472910. URL https://doi.org/10.1145/3452296.3472910.

Jifan Zhang, Henry Sleight, Andi Peng, John Schulman, and Esin Durmus. Stress-testing model specs reveals character differences among language models, 2025. URL https://arxiv.org/abs/2510.07686.

A Rule Inventory Construction

This appendix gives details omitted from the main description of rule extraction. We describe how we decide which missing principles receive annotations, how we choose annotation granularity, how we validate the extracted inventory, and how we handle authority labels for container sections.

Annotation criterion. We add a missing annotation only when the statement prescribes model behavior in a situation. This includes requirements, recommendations, prohibitions, and explicit permissions. The statement must be directed at the assistant, and the governed situation and behavior must be specific enough for compliance to be judged. Modal verbs alone are not sufficient: "the assistant may refuse" grants a permission and is therefore a principle, whereas "the assistant may encounter errors" describes a possibility and is not. We do not annotate example conversations, commentary, definitions, rationales, restatements of already annotated principles, or statements that govern parties other than the assistant. Existing focus-area annotations are preserved, so the completion pass only adds missing annotations.

Annotation granularity. We follow the granularity of the original annotations. A normative sentence is normally treated as one principle, even when it lists several coordinated actions or conditions. We split a sentence only when it states separable policies, for example when the polarity changes or when neighboring original annotations already mark parallel items separately. A qualification that narrows a principle stays with that principle, while a clause that permits otherwise restricted behavior is treated as a separate principle. In the completed annotation set, 26 sentences carry more than one annotation; 23 of these splits were already present in the original model spec.

Annotation review. After the initial annotation pass, we reviewed each section for missed principles and over-fragmented sentences. The original annotations remain unchanged. A static check verifies that the completion pass only inserts annotations and does not otherwise alter the specification text. Table 2 summarizes the original and added annotations by authority level.

Table 2: Original and added annotations by authority level.

Authority	Original	Added	Rules
Root	133	57	190
System	9	6	15
Developer	5	0	5
User	61	26	87
Guideline	71	37	108
Total	279	126	405

Rule extraction and validation. We use Claude Code (Anthropic, 2025) to extract the structured fields for each completed annotation, including the precondition P, postcondition Q, source statement, section path, and associated examples. We then validate the extracted inventory with static checks. The first check enforces one-to-one coverage between annotations and rules, so every annotation appears exactly once and no rule id is duplicated or missing. The second check confirms that each extracted rule remains linked to its source statement and examples, which lets later graph construction recover the context used to interpret P and Q.

Authority resolution. A rule inherits its authority from the nearest enclosing section with an authority label. This resolves 398 of the 405 rules. The remaining 7 rules appear under unlabelled

Table 3: Rule graph and analysis-unit statistics.

Statistic	Value
Extracted rules	405
Topics	45
Rule–topic nodes	625
Rules with multiple topics	162
Semantic edges	2,903
Syntactic edges	5,345
(authority, topic) groups	113
Analysis units	32
Rules per unit (min / median / max)	5 / 16 / 26

container headings whose subsections carry different authorities. We resolve these cases manually from the rule wording and the subsections governed by the container. This assigns 6 rules to root authority and 1 rule to system authority.

B Topic Registry and Rule Graph

This appendix provides additional details for topic derivation and analysis-unit construction that are omitted from Sections 4.1.2 and 4.2.

Topic registry. Three independent LLM subagents propose candidate topics and yes/no membership tests (*topic tests*). After merging equivalent tests, we obtain 18 candidate topics, 16 of which are proposed by all three agents. Two subagents then independently apply each test to a sample of 40 rules. We revise tests whose agreement falls below Cohen's $\kappa = 0.6$ and drop topics that remain below it; the only failing test improves from $\kappa = 0.48$ to $\kappa = 0.925$ after revision, so no topic is dropped. Uncovered rules motivate two additional topics, yielding the 20 topics in Table 4. An independent closure pass finds no missing topic. To validate the registry against held-out evidence, we remove cross-reference links from the text shown to the agents and later check whether cross-referenced rule pairs receive shared topics. Such pairs share a topic $7.16\times$ more often than random pairs (permutation test, $p \leq 0.001$). Seven topics contain more than 30 rules, so we refine them into sub-topics with the same procedure (inter-subagent $\kappa \geq 0.87$). The final registry contains 45 leaf topics, consisting of 32 sub-topics and 13 unrefined topics.

Topic assignment. Applying the membership tests to the 405 rules produces 625 rule–topic nodes. A total of 162 rules receive more than one topic because they govern multiple behaviors. For 21 rules, no test fits clearly. We record these rules as residue and assign each one to its closest topic so that every rule remains covered. Most residue rules describe how conflicts between instructions are resolved, which the registry excludes by design.

Analysis units. Table 3 summarizes the resulting graph. Grouping nodes by authority and topic initially produces 113 non-empty groups. The groups are uneven, with 101 groups containing fewer than 12 nodes and many containing only one or two rules. Such small groups give the verifier few rules to compare while still requiring a separate request. We therefore merge small groups within each authority level. Each group below 12 nodes is merged into the same-authority group with which it shares the most rules and graph edges, as long as the merged group has at most 28 nodes. This produces the 32 analysis units used in the experiments, with 14 at the root level, 8 at the user level, 8 at the guideline level, and one each at the system and developer levels.

C Baseline Implementation Details

All methods use Claude Opus 5 and run for five independent attempts, matching the detection budget used by VeriSpec. For direct LLM calls, we use high reasoning effort. DirectAgent is instantiated with Claude Code powered by Claude Opus 5, so it differs from the other baselines in execution but not in the model used.

Table 4: Topic registry. *Leaves* gives the number of leaf topics after refinement; unrefined topics have one leaf. *Rules* counts the rules assigned each topic; a rule may belong to several topics.

Topic	Leaves	Rules
Refined topics		
Refusal scope calibration	5	93
–– *When to decline a request and how much of it to fulfill*		
Tone and register calibration	4	54
–– *Adopting or avoiding a tone or register suited to the context*		
Bounded action and side-effect caution	6	41
–– *Limiting the scope and side effects of real-world actions*		
Self-transparency and mental-model upkeep	4	36
–– *Disclosing what the assistant can do, has done, assumed, or omitted*		
Trusted-instruction intent construal	5	36
–– *Interpreting the intended meaning of instructions the assistant follows*		
Clarify or proceed on assumptions	5	34
–– *Asking a clarifying question versus proceeding on assumptions*		
Truthful assertion boundaries	3	31
–– *Stating untruths or withholding material truths*		
Unrefined topics		
Perspective balance and stance-taking	1	30
–– *Presenting viewpoints proportionally or taking a stance on contested questions*		
Unprompted intervention dosing	1	30
–– *When to volunteer unrequested warnings, corrections, or interjections*		
Uncertainty signaling	1	28
–– *Conveying confidence through hedges, qualifiers, or declining to guess*		
Confidential information stewardship	1	22
–– *Withholding, disclosing, or transmitting private or privileged information*		
Output format adherence	1	22
–– *Conforming to a requested or default format, schema, or channel*		
Untrusted instruction screening	1	22
–– *Handling instructions embedded in quoted, retrieved, or tool-produced data*		
Length and thoroughness calibration	1	20
–– *Response length and detail relative to the need*		
Relational boundary keeping	1	16
–– *Limiting intimacy, dependence, or human-likeness in the relationship*		
Escalation to human help	1	15
–– *Referring the user to professionals, trusted people, or emergency services*		
Non-sycophantic stance independence	1	15
–– *Maintaining assessments under user pressure, flattery, or disagreement*		
Transformation fidelity	1	15
–– *Whether and how supplied content may be transformed*		
Helpful refusal delivery	1	12
–– *Explanation, tone, and alternatives offered with a refusal*		
Creative depth and work quality	1	11
–– *Depth, originality, and insight of the work product*		

DIRECTAGENT and DIRECTPROMPT. Both whole-specification baselines receive the anonymized specification once per attempt and do not receive extracted rules, analysis units, or the structured fields required by LLM-as-verifier detection. DIRECTAGENT starts each attempt from a fresh Claude Code session and asks the agent to find internal inconsistencies. DIRECTPROMPT uses a single non-agentic LLM call with the same whole-specification input and task instruction.

SECTIONTEXT. This baseline removes rule extraction and applies LLM-as-verifier detection directly to source text. We split the specification at second-level headings into 42 units, each consisting of one section with its subsections and examples, and remove annotation tags from the input. The prompt includes the same inconsistency definition and authority background used by VERISPEC. Because the text contains no rule identifiers, the model identifies the relevant statements by quoting them.

SECTIONGROUP. This baseline keeps rule extraction and LLM-as-verifier detection, but replaces rule-graph analysis units with source-section units. We group rules by authority and second-level source section. Groups containing a single rule are skipped because they contain no rule pair to compare, leaving 39 units that cover 399 of the 405 rules.

PLAINDETECT. This baseline keeps the same extracted rules, contexts, and 32 rule-graph analysis units as VERISPEC, but removes LLM-as-verifier detection. The prompt asks the model to report inconsistent rule pairs directly and to give a free-text explanation, without requiring the witness, unsatisfiability, and context-resolution fields used by VERISPEC.

Candidate consolidation. We normalize outputs in the same way for all methods. We first remove duplicate reports of the same rule or statement pair across attempts and units, and then merge reports that describe the same specification inconsistency. The merged candidates are the reported inconsistencies in Table 1.