A Neurosymbolic Approach to Natural Language Formalization and Verification

Chenyang An[1], Sam Bayless[1], Stefano Buliani[1], Darion Cassel[1], Byron Cook[1,2], Duncan Clough[1], Rémi Delmas[1], Nafi Diallo[1], Ferhat Erata[1], Nick Feng[1], Dimitra Giannakopoulou[1], Aman Goel[1], Aditya Gokhale[1], Joe Hendrix[1], Victor Heorhiadi[1], Marc Hudak[1], Dejan Jovanović[1], Andrew M. Kent[1], Benjamin Kiesl-Reiter[1](✉), Jeffrey J. Kuna[1], Nadia Labai[1], Joseph Lilien[1], Divya Raghunathan[1], Zvonimir Rakamarić[1], Niloofar Razavi[1], Michael Tautschnig[1,4], Ali Torkamani[1], Nathaniel Weir[1], Michael W. Whalen[1], Jianan Yao[3]

[1] Amazon Web Services [2] University College London [3] University of Toronto
[4] Queen Mary University of London

Abstract. Large Language Models perform well at natural language interpretation and reasoning, but their lack of formal correctness guarantees limits their adoption in regulated industries like finance and healthcare that operate under strict policies. To address this limitation, we launched Automated Reasoning checks (ARc): a public service that (1) uses LLMs with optional human guidance to formalize natural language policies, allowing fine-grained control of the formalization process, and (2) uses inference-time autoformalization to validate logical correctness of natural language statements against those policies. ARc performs multiple redundant formalization steps at inference time, checking the formalizations for semantic equivalence. Our benchmarks show that ARc exceeds 99% soundness and achieves a near-zero false positive rate in identifying logical validity. Our approach produces auditable artifacts that substantiate the verification outcomes and can be used to improve the original text. ARc is the first commercial offering from a major cloud provider to integrate automated reasoning into a generative AI guardrail.

1 Introduction

The capabilities of Large Language Models (LLMs) continue to advance rapidly, demonstrating unprecedented improvements in coherence and analytical accuracy [38,43,19]. Despite these advances, their tendency to generate plausible but incorrect information (hallucinations, cf. [41]) remains a barrier to widespread adoption in regulated sectors. Industries such as healthcare, financial services, and legal practices have legal and regulatory obligations for accuracy and auditability that current LLM technology has yet to meet [8].

Companies develop institutional policies to ensure compliance with laws and regulations. Such policies are typically captured in natural language (NL), defining rules, procedures, or guidelines. When deploying LLMs to answer questions

about these policies, a key challenge emerges: can we develop *guardrails* to ensure that the LLM outputs are correct? Consider an airline implementing a chatbot to assist customer service representatives in navigating refund policies: if the chatbot incorrectly claims that a customer is eligible for a refund, this could lead to legal exposure and loss of customer trust.

An effective guardrail would enable representatives to rely on chatbot responses by ensuring that when it reports an answer as valid, it actually is. Inspired by the concept of soundness in logic, we define *soundness* in our context as $(1-r)$, where r is the rate of incorrect validity claims across all decisions. High soundness thus means that across all requests, incorrect approvals are rare. Following established practices in safety-critical systems, where reliability is often measured in "nines" (e.g., 99% = "two nines," 99.9% = "three nines"), we target soundness levels of at least 99%, and secondarily focus on recall to maximize the probability of accepting valid content. We also pursue actionable feedback that steers LLMs toward content that a conservative guardrail can accept.

A natural candidate for robust guardrails are symbolic reasoning systems, as they use formal logic to generate independently verifiable guarantees [29]. This aligns well with policy documents, which rely on logical, rule-like statements (e.g., "if a flight is canceled or ..., then passengers are entitled to a refund"). But symbolic methods struggle with interpreting natural language, triggering the development of neurosymbolic approaches that combine the NL processing capabilities of neural networks with the mathematical rigor of symbolic systems [11].

This paper presents Automated Reasoning checks (ARc), a neurosymbolic approach exceeding 99% soundness on datasets it was not trained on. High soundness is also reflected in metrics such as false positive rate and precision. In addition, ARc delivers explainable verdicts and provides actionable feedback for revising LLM outputs. ARc is the first commercial offering from a major cloud provider to integrate automated reasoning into a generative AI guardrail.

ARc operates through two complementary components. The first, called Policy Model Creator (PMC), combines LLMs with symbolic reasoning to translate NL policies into formal *policy models* expressed in logic. It begins with an autoformalization phase that generates an initial policy model. This is followed by an optional vetting phase where domain experts refine the policy model with assistance from the system. Vetting enables domain experts to resolve ambiguities and inconsistencies in the original documents, or to correct potential imprecisions from autoformalization. Policy model creation occurs offline; its computation cost will be amortized across subsequent verification tasks.

The second component, called Answer Verifier (AV), verifies NL content against policy models. The AV uses LLMs to translate NL content into individual logical claims in the scope of the policy model. Each claim is analyzed separately and assigned a verification result, together with detailed logical explanations and corrective guidance. To increase reliability, the AV uses multiple LLMs to simultaneously formalize the same NL content, then uses symbolic reasoning to compare formalizations and assign confidence scores. The AV delivers auditable logical artifacts that substantiate the verification outcomes.

2 Related Work

Recent approaches use LLMs as judges to evaluate factual accuracy [15], though these rely on the same LLMs that are subject to inaccuracies to evaluate LLM-generated content. MiniCheck [34] provides efficient fact-checking by decomposing claims. RefChecker [12] introduces knowledge-centric verification against structured knowledge bases. SelfCheckGPT [21] leverages consistency across responses to detect hallucinations. FactCheck-GPT [37] provides comprehensive evaluation with fine-grained error categorization. While promising, these methods cannot provide formal guarantees. Our neurosymbolic framework verifies logical validity against formalized policies, achieving near-zero false positives.

Neurosymbolic systems combine LLMs (sometimes enhanced with Chain-of-Thought prompting [40,20,9]) with symbolic reasoning. They typically translate NL to formal representations that are then solved by external reasoners [26,25,4,30]. LINC [25] uses first-order logic with Prover9 [22]. Verus-LM [4] provides a multi-paradigm framework with IDP-Z3 [5]. SAT-LM [44] employs declarative prompting with SMT [24]. Logic-LM [26] supports multiple formalisms with self-refinement. Other approaches use custom DSLs [7] or Answer Set Programming [14,42,3].

These neurosymbolic systems formalize both the background knowledge and the query in a single step for each problem instance (or, in the case of [42], rely on hand-written background knowledge). In contrast, ARC autoformalizes both but separates them: the policy model can be vetted independently (through linting, symbolic test case generation, and human review) before being reused across many verifications. Regarding translation correctness, most of these systems offer only syntactic error detection and basic consistency checks (e.g., satisfiability of the formalized theory). LINC goes further by sampling multiple translations from a single model and applying majority voting over solver outcomes (Valid, Invalid, Unknown) rather than comparing translations at the logic level. ARC cross-checks per-query translations across diverse models using a notion of symbolic equivalence, with confidence thresholds indicating semantic agreement.

Autoformalization has been studied extensively in mathematics [36,32,39,16]. In contrast to ARC, these efforts typically assume that the background knowledge (the mathematical theory) has been pre-formalized, often by humans, and only the problem to be solved must be formalized by an LLM.

There is also a substantial body of work on dedicated formal languages for capturing policies and legislation, see, e.g., the Programming Languages and the Law (ProLaLa) workshops at POPL. Catala [23] is a notable example: a domain-specific language designed for systematic translation of statutory law into executable code, using defeasible reasoning to handle the general-case/exceptions logic common in legal texts. Such approaches capture laws directly in the dedicated language, whereas ARC autoformalizes existing NL policy documents into logic. Casadio et al. [6] propose a methodology for certifying the robustness of NLP models, showing how verification errors can stem from the NL representation layer. Their work on robustness guarantees for NL models is complementary to our redundant translation approach for mitigating translation errors.

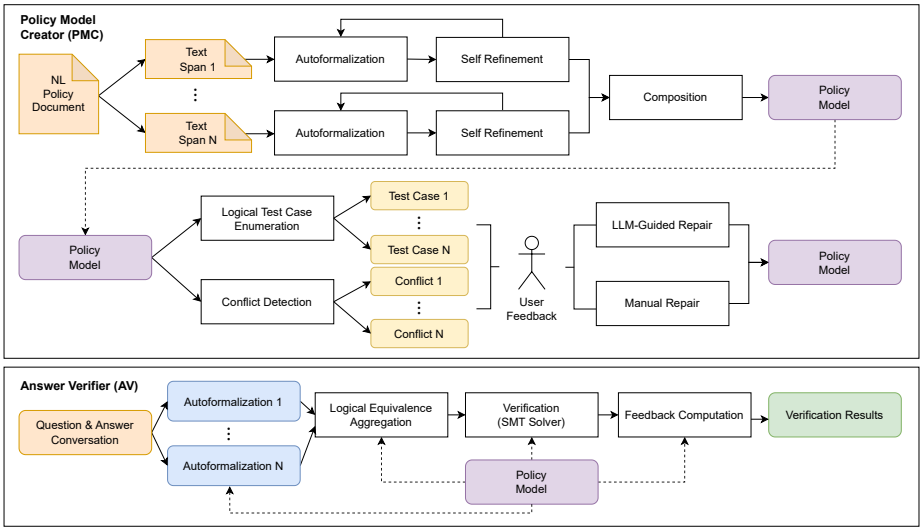

Fig. 1: End-to-end architecture of ARc

3 Methodology

Fig. 1 shows ARc's two main components: the PMC (§3.1) and the AV (§3.2). We illustrate our approach with an NL policy about park admission fees:

> **General admission**: The regular admission to the park is $50. The admission fee in the low season is 75% of the regular admission fee.
>
> **Discount**: Seniors (age greater than 65) qualify for a 40% discount. Whenever a discount applies, there will be a $10 flat discount processing fee.
>
> **Credit**: You can use credit for up to 50% of your final admission (1 credit for 1 dollar). However, if credits are used, then the discount rate is capped at 25%. You can purchase credit at a rate of $0.60 per credit. You can only purchase credit in increments of 5 (cost $3).
>
> **Tax**: A federal tax of 10% applies to the final expense.

Suppose a user asks "I am a senior and want to visit the park in the low season, and I have a budget of $35.40. Can I visit the park?", and we want to verify a chatbot's answer of "No, $35.40 is not enough."

ARc tackles the verification problem in two stages. In the first stage, the PMC autoformalizes the policy into a *policy model*: a set of logic rules, expressed in SMT-LIB [2], together with a *schema* that defines variables with their types and NL descriptions. SMT-LIB is a standardized logic language that uses prefix notation, where operators precede their arguments; e.g., "if x, then y" is written as `(=> x y)`. Fig. 2 shows snippets of the policy model. In the second stage, the AV autoformalizes the statement under validation into logic formulas (over the policy model schema), then uses an SMT solver to verify those formulas against

Variable	Type	Description
isLowSeason	Bool	Whether the admission day is in the low season
feeAfterDiscount	$\mathbb{R}$	Admission fee after discounts are applied but before tax

Rule 1: `(=> isLowSeason (= admissionFee (* 0.75 baseFee)))`

Rule 2: `(<= (* 2.0 customerCredits) feeAfterDiscount)`

Fig. 2: Snippets of policy model (top: variable schema; bottom: rules)

Translation (Confidence: 1.0)
P: `(and (= ageClass SENIOR) isLowSeason (= totalAdmissionFund 35.4))`
C: `(not isEntryAllowed)`
Result: *Satisfiable* (not *Valid*)
Counter-Example (shows C can be false):
creditUnit=3, customerCredits=15.0, creditDollarValue=9.0,
cashAmount=23.181, totalPaymentAvailable=38.181,
finalAdmissionFee=38.125, isEntryAllowed=true, ...
Satisfying Assignment (shows C can be true):
creditUnit=0, finalAdmissionFee=35.75, isEntryAllowed=false, ...

Fig. 3: Snippet of validation feedback

the policy model. The result includes the logic translation (with a confidence score between 0 and 1), the validation result, and further feedback (see Fig. 3).

In our example, the result is *Satisfiable*: the claim (cannot visit the park) is consistent with the premises (the person is a senior, it is low season, and they have a fund of $35.40 available) but doesn't necessarily follow from them. AV provides two assignments as feedback: one showing a counter-example where admission is possible, and a case where the person cannot enter the park. The key difference is in the use of credits ($creditUnit = 3$ vs. $creditUnit = 0$).

Explanation. The person needs to use $15 worth of credits for admission. The admission fee in the low season is $37.5 (75% of $50). After applying the 25% senior discount (capped at 25% because of credit use) and adding the $10 discount processing fee, the actual admission fee becomes $38.125. This fee can be paid by combining $15 of credit (cost: $9) with a $23.125 cash payment, for an expense of $32.125. After adding a federal tax of 10%, the final expense becomes $35.3375, which is within the budget of $35.40. Both Claude Sonnet 3.7 and Opus 4.1 (with reasoning mode) incorrectly classified the answer as valid, providing plausible but flawed reasoning (see Appendix Figs 10 and 11).

3.1 Policy Model Creator (PMC)

The PMC takes a policy document written in natural language and autoformalizes it into a policy model. It also provides an array of utilities to support users in policy vetting, described in detail in the *Policy Vetting* paragraph below.

Autoformalizing Policies To handle the size and complexity of real-world policy documents in the face of known LLM reasoning limitations around context size and distractors [28,18], the PMC takes a divide-and-conquer approach to autoformalize documents into logic (see Fig. 1).

The PMC first splits the input document into a set of text spans. These are processed using an incremental, refinement-guided autoformalization procedure: A language model processes each span and identifies statements that express coherent, formalizable meaning. For each statement, the LLM translates the semantic content into a list of SMT-LIB datatypes, variables, and logical constraints (*rules*). The LLM's context maintains existing declarations within a span to avoid duplicated or conflicting declarations. The complete formalization of a span is what we call a *policy unit*. If this process introduces an error (e.g., malformed syntax), we provide the invalid declarations and their failure causes to the LLM for repair in a refinement loop.

After the PMC has formalized all text spans, it composes the policy units into a single policy model. The PMC generates textual embeddings of variables and clusters them using cosine similarity. Variables within a cluster are unified, while variables that share the same name but are not clustered are renamed. Consistent replacement of original variables with unified variables is performed for the rules of each policy unit, then the rules are aggregated, dropping syntactic duplicates. The resulting policy model is a structured representation of the document, consisting of three fields: datatypes, variables, and rules. Each variable is associated with an NL description that explains its meaning in terms of the source document, as shown in Fig. 2. This initial policy model is then vetted, as described in §3.1. We measure the relationship between document size and formalized policy size in §A.1.

PMC Policy Vetting The initial policy model might contain errors and omissions. Additionally, as shown in §4.2, NL policies often contain ambiguities that only subject matter experts can resolve. We therefore provide users with several methods for vetting of their policy models: linting, inspection, and testing (both manual and automatic). We also develop automated repair approaches around these vetting methods.

Linting. A linter for our policy models that checks integrity and consistency properties beyond the simple malformedness errors caught during autoformalization. We perform a mix of syntax-based and semantic checks: we detect unused variables and types, contradictory rules, and disjoint rule sets. The list of warnings is shown to the user, who can address them directly (e.g., by deleting an unused variable) or through more detailed policy inspection and repair.

Inspection. Manual inspection allows users to review the generated policy model, similar to code review in software development. We assist users by providing two views of the rules for inspection: SMT-LIB for experts and structured English for non-experts. Structured English is generated using templates (like "if ... then ...") to avoid potential hallucinations from LLMs. Users also can provide NL feedback on the policy, which triggers an automatic LLM-based policy repair step that adjusts the policy model. Manual inspection provides strong correctness guarantees when all rules are carefully reviewed, but it can be challenging with large numbers of complex rules that have intricate interactions. The PMC therefore also provides testing as an additional policy vetting methodology.
Testing. Testing provides a systematic way to validate policy models through examples. Similar to unit tests, test cases in the PMC are either NL question-answer pairs or simple NL statements with their expected outcomes provided by the user (e.g., valid, invalid). Test cases can either be provided manually by users or generated automatically. For manually provided test cases, the PMC "executes" them by running the AV to compute the findings; a mismatch indicates an error in the policy model or in the AV translation. The PMC also offers automatic, symbolic test-case generation that leverages an SMT solver to systematically explore the state space of the policy model. Since such test cases are generated symbolically, each comes with its provably-correct actual finding, so a mismatch isolates the error to the policy model. Either way, the supplied information (e.g., rules justifying the result) helps users diagnose and repair.

3.2 Answer Verifier (AV)

The AV uses LLMs to translate natural language (i.e., question-answer pairs) into a list of premise-conclusion pairs, where premises (P) and conclusions (C) are expressed in the policy model's schema. For example, "Since you have at least $50, you can enter the park" translates to premise ($\geq$ *totalAdmissionFund* 50) and conclusion *isEntryAllowed* which represent the contextual facts and logical consequences, respectively.

To increase translation confidence, the AV *redundantly translates* the NL statement using k LLMs (Alg. 1), then compares the resulting premise-conclusion pairs semantically using an SMT solver to estimate a confidence score for each pair. Intuitively, the confidence score of a premise-conclusion pair $\langle P, C \rangle$ is the proportion of the k translations that non-vacuously entail the implication $P \Rightarrow C$ (i.e., entail it without rendering P contradictory). For example, consider the text from §3 and the policy model in Fig. 2. Redundant translation with three LLMs produces three identical premise-conclusion pairs, thus producing the results in Fig. 3 with confidence score 3/3 (1.0). If one LLM instead produced a pair with a different conclusion (`isEntryAllowed`), AV would return two distinct pairs: the original from Fig. 3 with confidence 2/3, and the alternative with confidence 1/3.
Validation Feedback. After translation, the AV uses an SMT solver to validate each claim $\langle P, C, \mathit{conf} \rangle$ against $\mathcal{M}$ while producing logically grounded feedback given the following precedence order:

Algorithm 1 AV Redundant Translation

procedure RedundantTranslation(msg, $policy$, $LLMs$)
  $F \leftarrow \emptyset$; $Ts \leftarrow [Translate(msg, policy, llm) \mid llm \in LLMs]$
  for each $\langle P, C \rangle$ in every $T \in Ts$ **do**
    F.add($\langle P, C, cf \rangle$) where $cf = |\{T' \in Ts \mid T' \models (P \Rightarrow C) \wedge T' \not\models \neg P\}|/|Ts|$
  return F

Finding	**Condition**
TooComplex	Text or SMT-LIB translation exceeds token limits
TranslationAmbiguous	Confidence below threshold (default: 3/3)
NoTranslations	LLM cannot translate text to policy model vocabulary
Impossible	$\mathcal{M} \models \neg P$ (premises contradict policy model)
Invalid	$\mathcal{M} \wedge P \models \neg C$ (conclusion is inconsistent)
Valid	$\mathcal{M} \wedge P \models C$ (conclusion is entailed)
Satisfiable	$\mathcal{M} \wedge P \not\models C$ and $\mathcal{M} \wedge P \not\models \neg C$ (consistent, not entailed)

For *Impossible*, *Invalid*, and *Valid* findings, the feedback includes relevant rules from the policy model extracted from the SMT solver, providing sufficient information for independent verification via theorem prover. For *Satisfiable* findings, the feedback returns assignments ("scenarios") demonstrating how the answer can be correct or wrong, thus providing hints on how the premises could be extended to make the conclusion valid (see Fig. 3, for example, where the use of credits is a differentiator between the scenarios). For *TranslationAmbiguous* findings, the feedback presents two differing translations, together with an assignment that is satisfiable in one translation but not in the other. *NoTranslations* findings return the untranslatable text segments. Logic warnings are surfaced if premises or conclusions are always true or false irrespective of policy rules.

4 Empirical Evaluation

In order to understand ARc's effectiveness as a guardrail, and the impact of our design choices, we evaluate ARc, and more specifically the AV, around the following research questions (RQs):

RQ1 (Reliability of Validating Logical Accuracy): How reliably does ARc validate logical accuracy compared to alternative baselines?

RQ2 (Impact of Redundant Translation): How does redundant translation (§3.2) impact ARc's performance?

RQ3 (Effectiveness of ARc's feedback): Is ARc's feedback effective in driving improvement of LLM outputs?

Metrics. We frame logical accuracy detection as a binary classification problem: decide whether NL statements are *Valid* or not.

To target high-stakes applications, our primary objective is to eliminate false positives across the entire pipeline. In this context, rejecting borderline

Table 1: Comparison of logical accuracy detection performance on ConditionalQA-logic [31]. The columns show soundness (S), false-positive rate (FPR), precision (Pr), recall (Re), F1 score (F1), accuracy (Ac), counts of true/false positives/negatives (TP/FP/TN/FN), and error count (Error#). Approaches meeting soundness threshold are highlighted, as well as best and worst values for other metrics.

Method	S ↑	FPR ↓	Pr ↑	Re ↑	F1 ↑	Ac ↑	TP ↑	FP ↓	TN ↑	FN ↓	Error# ↓
ARc (#3-ensemble, threshold=3/3)	99.4	1.8	94.4	14.9	25.8	42.4	169	10	548	962	5
ARc (#3-ensemble, threshold=2/3)	99.2	2.5	93.2	16.9	28.6	43.5	191	14	544	940	9
ARc (without redundant translation)	98.4	4.8	92.2	28.0	43.0	50.2	317	27	531	814	1
LLMaJ (#3-ensemble, threshold=3/3)	98.2	5.4	92.4	32.4	47.9	52.9	366	30	528	765	-
LLMaJ (#3-ensemble, threshold=2/3)	97.9	6.3	92.1	36.3	52.0	55.2	410	35	523	721	-
LLMaJ (1x Sonnet4.5)	97.9	6.3	92.0	35.4	51.1	54.6	400	35	523	731	-
LLMaJ (1x Sonnet4.5 w/ extended thinking)	97.1	8.8	92.7	55.0	69.0	67.0	622	49	509	509	-
FG Implicit span-level [15]	95.0	15.2	90.4	70.6	79.2	75.3	798	85	473	333	-
FG JSON [15]	94.6	16.5	89.7	70.5	78.9	74.8	797	92	466	334	-
FG Response-level [15]	83.8	49.1	78.3	**87.6**	82.7	75.5	**991**	274	284	**140**	-
MiniCheck [17]	88.5	34.9	83.0	84.4	83.7	78.0	954	195	363	177	-
RefChecker [12]	91.9	24.6	86.3	76.0	80.8	75.8	860	137	421	271	-
SelfCheckGPT [21]	93.0	21.1	89.2	86.3	**87.7**	**83.8**	976	118	440	155	-
Logic-LM [26]	98.0	5.9	86.4	18.6	30.6	43.5	210	33	525	921	1149
Proof of Thought [7]	88.8	33.9	81.2	72.2	76.4	70.2	817	189	369	314	**0**
LINC [25]	**99.9**	**0.2**	**98.7**	6.6	12.4	37.4	75	**1**	**557**	1056	696

cases (*not-valid*) is favorable: such outputs can either be refined by the answer-generating LLM using ARc feedback, or escalated to human experts. We thus define *soundness* as $1 - \frac{\#\text{False Positives}}{\#\text{Samples}}$, measuring the rate at which incorrect approvals occur across all decisions.

A natural alternative to soundness would be precision. The two metrics differ in perspective: precision estimates a posterior ("Given a request was classified as *Valid*, how likely is that verdict to be correct?"), while soundness estimates a prior ("How likely is a request to receive an incorrect *Valid* verdict?"). Based on customer feedback, we prioritized the prior perspective: ARc's customers are service operators deploying chatbots across thousands of daily interactions, where the relevant operational question is how likely any given request is to be incorrectly approved. Like any soundness notion, our soundness rewards abstention and is incomplete on its own, so we use recall to complement it. As mentioned in §1, we target soundness levels of at least 99%.

In addition to soundness, we use standard classification metrics (precision, recall, F1, accuracy), treating *Valid* as the positive class and all others as negative. When comparing alternative methods, *Valid* recall is used as a tie-breaker under the requirement of maintaining high soundness.

4.1 Evaluation of Logical Accuracy Validation

Dataset. Several reasoning benchmarks (e.g., FOLIO [10], ProofWriter [33], and LogicNLI [35]) test logical inference, but our focus is on validating whether NL answers comply with formalized policy documents. We therefore focus on the

ConditionalQA dataset [31] because it is well-aligned with our task, featuring: 1) questions that require compositional logical reasoning, 2) variety of questions (yes/no, multiple answers, not-answerable), and 3) human annotated answers.

We enrich the ConditionalQA dev dataset (391 labeled QAs over 59 source documents) with several types of "not valid" examples beyond its original "valid" / "not_answerable" classification. The resulting set includes the following categories: *Valid* (logically correct), *Invalid* (incorrect due to wrong conditions), *Satisfiable* (missing necessary conditions), *Impossible* (contradictory conditions), and *NoTranslations* (content that cannot be formalized, originally classified as not-answerable). These categories were created by systematically manipulating the conditional structure of original answers: removing conditions (*Valid* $\rightarrow$ *Satisfiable*), negating the claim (*Valid* $\rightarrow$ *Invalid*), or merging contradictory conditions (*Valid* $\rightarrow$ *Impossible*). The extended dataset (ConditionalQA-logic) contains 377 *Valid* examples and 186 examples that are not *Valid* (112 *Invalid*, 56 *Satisfiable*, 4 *Impossible*, and 14 *NoTranslations*).

Our goal in this section is to evaluate the AV's soundness in an end-to-end setting with automatically generated, unvetted policy models. The policy models used for Table 1 were autoformalized by the PMC and not manually refined. Section 4.2 provides initial evidence that human vetting can further improve both soundness and recall.

RQ1: Reliability of Validating Logical Accuracy. Table 1 reports on the comparison of ARc against alternative methods: LLM-as-Judge (LLMaJ) with different prompting strategies, FACTS Grounding (FG) [15], fine-grained hallucination detection methods [34,17,21,12], and other neurosymbolic approaches [26,7,25]. This evaluation focuses on the ability of each approach to predict validation labels for QA pairs about given NL policy documents. We evaluate each of the 563 benchmark instances three times to account for nondeterminism in LLM outputs, yielding 1689 total decisions.

We first examine the approaches that meet our soundness threshold, which are only ARc and LINC [25]. Between the two, ARc has a higher recall (14.9% vs 6.6%) and significantly fewer errors (5 vs 696 out of 1689 instances), where errors are instances in which the system was unable to formalize the QA contents and thus could not produce a verdict. Second, we compare ARc to others that did not achieve the required soundness threshold of 99%. ARc's reliability comes with lower recall (14.9% for the configuration with soundness over 99%), but the tradeoff is intentional: in safety-critical domains, false approvals are far more costly than false rejections. Among the methods below the soundness threshold, LLMaJ (#3-ensemble, threshold=3/3) comes closest at 98.2% soundness with precision similar to ARc's (92.4% vs 94.4%), but produces 3 times as many false positives (30 vs 10, or 5.4% vs 1.8% FPR). In contrast, FG Response-level has the highest recall of 87.6%, but this comes at the cost of soundness dropping to just 83.8%, the second-lowest of all methods.

RQ2: Impact of Redundant Translation. ARc uses redundant translation (Alg. 1) to increase confidence in NL-to-logic translations. Comparing the first 3 rows of Table 1: soundness rises from 98.4% to 99.4% and FPR drops from 4.8% to 1.8%,

at the cost of reduced recall (28.0% to 14.9%). Lowering the confidence threshold from 3/3 to 2/3 recovers recall to 16.9%, with soundness dropping to 99.2%.

ARC's recall is low compared to the other techniques in Table 1, meaning that in many cases it returns results other than *Valid* for content that should be *Valid*. ARC is intended to be deployed as a guardrail for chatbots in regulated industries, where an incorrect approval (e.g., 'you qualify at 0% interest') causes real harm at scale. In this setting, a non-*Valid* verdict is not a failure: the system can revise the answer using ARC's feedback (see RQ3) or route to a customer support agent. Low recall thus means more deferrals, not more failures, and at 99.4% soundness, *Valid* verdicts carry strong enough assurance to reduce the review burden. In Section 4.2, we demonstrate that three rounds of revision with ARC's feedback can drive the rate of *Valid* answers from 9% to 46% on a real-world policy.

4.2 Refining Real-World Policy Models and Answers

To understand the applicability of ARC to real-world policies, we collected customer-facing policy documents from six different businesses, refined with human-in-the-loop vetting (§3.1). We use these policies to evaluate iterated self-refinement of LLM answers using ARC feedback (*RQ3*); in Appendix A.1, we additionally share one policy as a case study of the manual refinement process itself (soundness: 96.8%→100%, recall: 25%→45.5%).

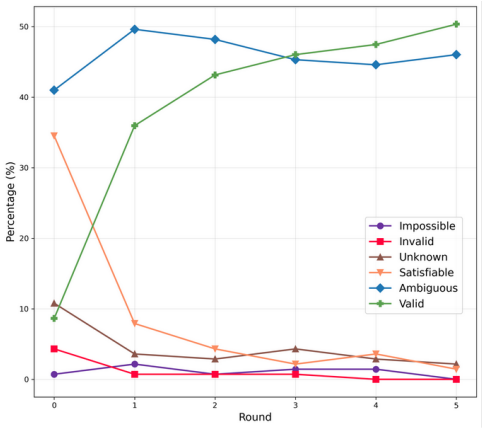

Fig. 4: ARC validation finding distribution after k iterations of answer revision using ARC feedback. At $k = 0$, we plot the finding distribution before any revisions.

RQ3: Effectiveness of ARC's Feedback. In a real-world setting, users may deploy LLM-based chatbots to answer questions about policies, requiring assurance of answer correctness. The formally-grounded feedback that ARC provides (Section 3.2) can be used for automated answer revision: given a non-*Valid* verdict, an LLM iteratively revises its answer using ARC's feedback. Fig. 4 shows how the relative percentages of each finding type evolve as an LLM iteratively revises AV-judged non-*Valid* answers using ARC feedback (#3-ensemble, threshold=3/3). After just three iterations, the LLM goes from 9% to 46% *Valid* answers. Primarily, this comes from a sharp reduction in AV-judged *Satisfiable* answers (where the answer could be true or false depending on context not provided in the question or answer). By providing logically-derived scenarios

showing when the answer is true and when it is false, ARC enables the LLM to effectively revise these into *Valid* answers. ARC's feedback is less effective in revising *TranslationAmbiguous* and *NoTranslations* answers. Our analysis shows that in these cases revising the policy model is more effective; e.g., the policy model could be missing variables, leading to a failure to formalize the answer, or the policy model could have variables that overlap in meaning, leading to a failure to generate a consistent formalization of the answer. Average iterations to *Valid* and success rate per finding type can be seen in Appendix Table 5.

5 Conclusions, Limitations, and Future Work

We presented ARC, a neurosymbolic guardrail that exceeds 99% soundness when validating LLM answers against NL policies. This soundness comes at the cost of recall, a tradeoff we believe appropriate for regulated industries. Since its launch, ARC has been adopted across industries, including responsible AI in education [27], financial AI agents [13], and logistics operations [1]. Soundness of ARC heavily depends on the quality of the policy model that it uses for validation. For this reason, ARC enables human oversight. As we have shown, ARC provides meaningful feedback to aid automated answer revision, resulting in an increase of *Valid* answers. Despite ARC's high soundness, there are limitations:

- *Document types*: Policies with numerical tables, cross-references, or implicit assumptions can be challenging to formalize without human vetting.
- *Autoformalization challenges*: Subtle issues like ambiguous pronouns or implicit temporal scoping can lead to incorrect formalizations.
- *Computational cost*: Redundant translation requires multiple (3) LLM calls, resulting in average 5-15 second latency and increased API cost per Q/A validation with our current implementation.
- *Human effort*: The investment for human vetting of policies, while amortized over time, remains a significant upfront cost, especially for large documents.
- *Dataset representativeness*: The mechanically mutated evaluation examples (removing conditions, negating claims, merging contradictory conditions) may not fully capture the kinds of errors real LLMs produce in practice. Evaluating on naturally occurring LLM failures is a direction for future work.

Future work includes exploring automatic and confidence-aware focused vetting, fine-tuned translation models for improving accuracy and latency/costs, and improved logical formalisms to address current limitations. Our approach directly benefits from advances in LLMs and generative AI techniques: as models improve, their ability to formalize natural language to logic will too. We are confident ARC will inherit these improvements while maintaining the mathematical guarantees provided by symbolic reasoning.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Data-Availability Statement. ARc is publicly available as part of Amazon Bedrock Guardrails at `https://aws.amazon.com/bedrock/guardrails/`. The evaluation uses a non-public augmentation of ConditionalQA [31].

A Appendix

A.1 Additional Experiments

Impact of Human Policy Vetting We designed a case study to evaluate the support that ARC provides for policy refinement and its impact on ARC's performance as a guardrail. In our study, we create and compare two formalizations of an airline's refund policy: one using the PMC to create a policy model without additional vetting, and one that is revised using a human-in-the-loop (as described in §3.1). To evaluate these two formalizations, we created a test suite balanced across three sources: 1) verbatim statements from the original policy; 2) Q/A pairs generated by an LLM; 3) Q/A pairs generated by three different individuals. The expected classification labels were determined manually.

Table 2: Effect of human vetting on logical accuracy detection for RyanAir's customer service policy.

Method	S ↑	FPR ↓	Pr ↑	Re ↑	F1 ↑	Ac ↑	TP ↑	FP ↓	TN ↑	FN ↓
ARC with human vetting	**100.0**	**0.0**	**100.0**	**45.5**	**62.5**	**61.3**	**20**	**0**	**18**	**24**
ARC without any human vetting	96.8	8.7	84.6	25.0	38.6	43.6	11	2	16	33

We found that clarification of ambiguities is a task that warrants human input. There were several ambiguities and edge cases in the study document, such as what happens when a death occurs *on* (as opposed to prior to) the day of travel. The most notable ambiguity is where the policy states: "you may be entitled for a refund if your scheduled time of departure is delayed by at least 5 hours". It is unclear whether one is entitled to such a refund only if they did not travel (a prerequisite the document clarifies in other cases).

The detailed finding categories were helpful in guiding policy revisions. *TranslationAmbiguous* or *NoTranslations* findings typically indicated that variables needed to be added or revised. *Impossible* findings, on the other hand, consistently indicated subtle rule inconsistencies and triggered rule revisions. A recurring pattern in this category is one where autoformalization fails to recognize valid exceptions to the rules that may appear in a different section of the document. For example, the document begins with a broad statement that "If your flight operated and you didn't travel, you're not entitled to a refund," but later on lists several special circumstances under which passengers may indeed qualify for a refund even if their flight operated. As shown below, ARC correctly flags the displayed question-answer pair as *Impossible* and identifies the problematic rules in the policy model.

Question-answer pair:
Q: My flight operated but I did not travel because I was denied boarding. Am I eligible for a refund?
A: Yes, if you were denied boarding you are eligible for a refund.

ARC interprets the question-answer as follows:

```
Premise: (and didFlightOperate
    (not didPassengerTravel)
    (= flightDisruptionReason DENIED_BOARDING))
Conclusion: isRefundEligible
```

ARC judgment: *Impossible*

ARC returns two rules to explain this finding:

```
1: (=> (and didFlightOperate
        (not didPassengerTravel))
    (not isRefundEligible))

2: (=> (= flightDisruptionReason DENIED_BOARDING)
       isRefundEligible)
```

The revision process was a non-trivial effort of several person-hours. As illustrated in Table 2, refinement played a central role in creating a policy that is effective as a guardrail. In fact, human vetting increased soundness to 100% and the recall to 45.5%. Note that, even though some tests were run during human vetting, about a third of the tests were held-out. Validation inaccuracies were evenly spread across all classes. Since vetted policies can be reused across future validation tasks, the cost is amortized over time, making human-in-the-loop vetting a practical and effective complement to automated formalization.

Effectiveness of Feedback for Mitigating Logical Inaccuracies Table 3 sheds light on how effective is ARC's feedback compared to existing state-of-the-art methods for mitigating logical inaccuracies using the CONDITIONALQA-LOGIC dataset. We utilized a uniform experimental methodology for each method M: raw feedback from M is incorporated into an identical prompt for answer revision through an LLM for at most 10 revision iterations. In each revision iteration k ($0 \leq k \leq 10$), the answer from the previous iteration is evaluated by M, and if labeled as not *Valid* by M, a new revised answer is generated through an LLM (Claude Sonnet 3.7) by incorporating the raw evaluation feedback from M in the Fig. A.5 prompt. Columns 2-6 (marked under % *Valid*) report the percentage of answers classified as *Valid* by each method M after different revision iterations $k \in 0, 1, 3, 5, 10$. We further evaluated the final revised answers generated after at most 10 revision iterations separately with top 3 judges from Table 1 (in terms of soundness), and report soundness and recall with respect to the corresponding judge J. Additionally, the CONDITIONALQA-LOGIC dataset contains 14 questions labeled as not answerable from the source text by human annotators from [31], for which we report the count of final revised answers labeled as *Valid* by method M as false positives counts in the last column.

Examining the above results, we observe the following:

- ARC's feedback helps drive the count of *Valid* answers (as evaluated by ARC) from 55 tests initially to 123 tests after 10 iterations. ARC remains cautious and conservative when passing revised answers as *Valid*, and flags answers even in cases with minute/subtle errors or discrepancies.
- FG Implicit span-level, FG JSON, and LLMaJ (1x Sonnet3.7) are much more liberal, reaching $> 93\%$ *Valid* after 1 revision and $> 99\%$ after 10.
- When comparing final revised answers through different judges, ARC stands out with the highest overall soundness across all top 3 judges.

Table 3: Effectiveness of different methods in providing feedback to mitigate logical inaccuracies (as detected by the same method) on ConditionalQA-logic [31]. For each method, raw feedback is incorporated into an identical prompt for LLM-based refinement (Fig. A.5). We report the percentage of responses classified as *Valid* by the method after at most 10 answer refinement iterations. Columns S & Re report soundness and recall of final revised answers evaluated separately with top 3 judges from Table 1 (w.r.t. soundness) as ground truth. Column FP_{Human} reports false positives in the final revised answers for questions labeled as not answerable by human annotators.

Method M	% *Valid*					Judge J to evaluate final revised answers after 10 revision iterations						Not ans.
						ARc (#3-ensemble)		LLMaJ (#3-ensemble)		FG Implicit span-level		
	@0	@1	@3	@5	@10	S	Re	S	Re	S	Re	FP_{Human}
ARc (#3-ensemble)	10.5	15.5	19.2	21.3	23.6	—		**89.9**	33.3	**93.9**	72.2	**1**
LLMaJ (#3-ensemble)	23.8	68.2	73.4	75.9	77.6	**32.2**	30.9	—		91.4	95.5	9
FG Implicit span-level	38.7	93.1	99.2	99.6	**100.0**	9.6	**100.0**	54.2	**100.0**	—		14
FG JSON	53.6	93.1	99.2	99.6	99.6	11.3	96.6	46.6	99.2	78.2	99.5	14
LLMaJ (1x Sonnet3.7)	42.0	94.6	99.0	99.6	99.8	10.3	98.2	56.3	99.7	80.8	**99.8**	14

- As expected, FG Implicit span-level, FG JSON, and LLMaJ (1x Sonnet3.7) methods show near-perfect recall. However, the final revised answers from these methods show major soundness gaps when evaluated with ARc or LLMaJ (#3-ensemble) judges, making them unsuitable for high-stakes tasks.
- For the 14 questions that are annotated as not answerable, ARc showed significantly lower false positives compared to other methods.
- Overall, the soundness-recall tradeoff persists across methods: ARc demonstrates the highest soundness and agreements across judges, but at the cost of lower recall rates. ARc conservative judgments and attention to minute and subtle details are well aligned for safety-critical domains where false approvals are far more costly than false rejections.

Utilizing PMC Rules Beyond AV In this experiment, we used rules generated by PMC (§3.1) for ConditionalQA-logic as in-context information for LLMaJ, either instead of or in addition to the source document. Table 4 summarizes the key results for both configurations.

Table 4: Overall logical accuracy detection across types of in-context information for LLM baselines.

In-Context Information	S ↑	FPR ↓	Pr ↑	Re ↑	F1 ↑	Ac ↑	TP ↑	FP ↓	TN ↑	FN ↓
LLMaJ (#3-ensemble, threshold=3/3)	98.3	5.0	92.1	29.0	44.2	50.9	304	26	493	743
1) with PMC rules, without Doc	**98.9**	**3.5**	**93.5**	24.7	39.1	48.5	259	**18**	**501**	788
2) with PMC rules, with Doc	97.6	7.1	90.6	**34.2**	**49.7**	**53.6**	**358**	37	482	**689**

PMC Scaling In order to examine how PMC scales with respect to policy size, we run it over a large real-world document consisting of 274 pages of content. Each page consists of approximately 500 tokens. Figure 5 measures the number of datatypes, variables, rules with respect to document size.

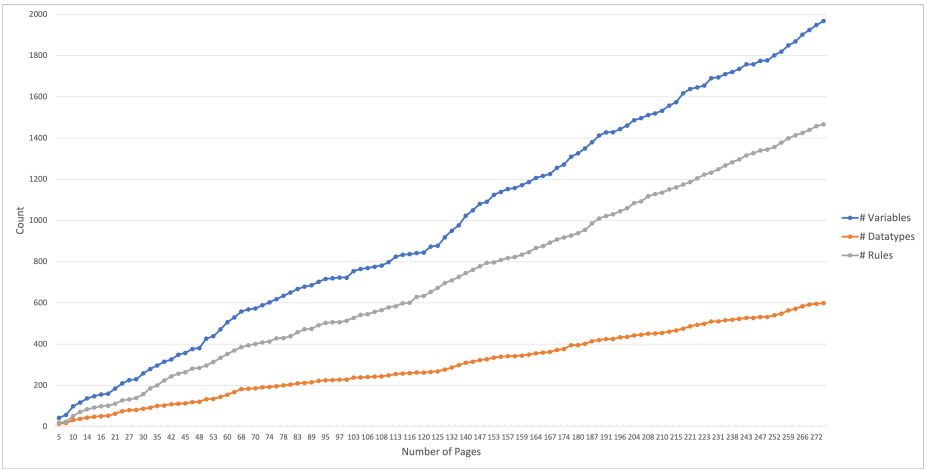

Fig. 5: Number of datatypes, variables, and rules with respect to number of unique pages of text formalized, where each page is approximately 500 tokens.

We can see that PMC-produced policy size (in terms of counts of datatypes, variables, and rules) scales smoothly with document size. The overall document amounts to 600 datatypes, 1968 variables, and 1467 rules.

A.2 Experiment Details

Table 5: Revision outcomes by starting finding type. Success rate is the percentage of items reaching VALID within 5 revision rounds; average iterations is computed only over those that reached VALID.

Finding Type	Success Rate	Avg. Iterations
Impossible	0.0%	–
Invalid	83.3%	2.40
NoTranslation	40.0%	2.33
Satisfiable	68.8%	1.45
Ambiguous	24.6%	1.71

Dataset Details (Section 4.1). The original ConditionalQA dataset provides binary labels (valid or not-answerable) for question-answer pairs, suited for the task of answering complex questions over long documents. However, the dataset's defining characteristic–that answers are only correct under specific stated conditions–presented a unique opportunity for more comprehensive evaluation suited for our logical accuracy detection task. We systematically leveraged this conditional structure to generate additional evaluation categories by using the relationship between answers and their associated conditions:

Satisfiable: Answers with non-empty conditions deliberately removed. Tests whether a method can identify when necessary conditions are missing, even though the core answer content remains logically satisfiable within the document context.

Invalid: Answers with incorrect alternative conditions applied. Evaluates a method's ability to detect when conditions directly contradict the stated answer based on the document (e.g., flipping yes/no responses while maintaining the original conditions).

Impossible: Contradictory yes/no answers with non-empty conditions merged. Tests detection of logical impossibilities by combining mutually exclusive responses (yes/no answers) under unified condition sets.

Valid: Original answers with their stated conditions intact. Represents the ground truth conditional answers as provided in the dataset.

NoTranslations: Questions originally marked as not-answerable in the dataset. Preserves the dataset's inherent cases for which an answer cannot be given based on the source text.

This systematic augmentation transformed the original dataset into a multi-dimensional evaluation dataset for logical accuracy detection that tests conditional reasoning capabilities across various logical relationships and edge cases.

Baseline Details (Section 4.1). For a fair comparison, we evaluated all methods under comparable configurations:

LLMaJ: For a comprehensive LLM-as-judge baseline that takes into account different validation output types comparable to ARc, we developed a customized prompt (Fig. 7) with explicit instructions and details about the logical accuracy validation task. When coupled with majority voting as an ensemble of 3 (i.e., LLMaJ (#-ensemble) in Table 1), we utilized a comparable ensemble configuration as utilized in ARc (#-ensemble) (i.e., ARc with redundant translation using 3 LLM calls).

FACTS Grounding: We utilized the exact same prompts as presented in [15] using Claude Sonnet 4.5 as the LLM.

MiniCheck: We evaluated the method in its recommended default configuration as presented in [17].

RefChecker: We used the accurate-context setting from [12] (document in the prompt) and evaluated joint claim checking with Claude Sonnet 4.5 as Extractor and Claude Sonnet 4.0 as Checker.

SelfCheckGPT: We ran [21] with Claude Sonnet 4.5 as sampler LLM (3 samples) and Claude Sonnet 4.0 as judge LLM.

Logic-LM: We adapted the method from [26] and configured with Prover9[1] as the solver.

A.3 Implementation Details

Fragment of SMT-LIB Utilized by ARc ARc supports the autoformalization of natural language policy documents into quantifier-free SMT-LIB with non-linear arithmetic (QF_NIRA) as shown in Fig. 6. This logical fragment allows us to express predicates over integers, real numbers, booleans, and *datatypes* (enumerated values). We restrict our approach to this logical fragment because regulatory policy documents are typically written for human consumption and thus lack complex quantification.

$$\tau := \texttt{Int} \mid \texttt{Real} \mid \texttt{Bool} \mid k$$
$$d := (\texttt{declare-datatype}\ k\ (v_1 \dots v_n)) \mid (\texttt{declare-const}\ x\ \tau)$$
$$op := + \mid - \mid / \mid * \mid = \mid > \mid < \mid \leq \mid \geq$$
$$c := \textit{integers} \mid \textit{reals} \mid \texttt{true} \mid \texttt{false}$$
$$e := x \mid v \mid c \mid (\texttt{and}\ e\ e) \mid (\texttt{or}\ e\ e) \mid (\texttt{not}\ e) \mid (\texttt{=>}\ e\ e) \mid (op\ e\ e)$$
$$s := (\texttt{assert}\ e)$$
$$p := d_1 \dots d_n; s_1 \dots s_n$$

Fig. 6: The fragment of SMT-LIB expressed by our formalization.

A.4 LLM-as-Judge System Prompt

> You are an expert document validator. Your task is to determine whether a given answer to a question is correct according to the provided policy document. When a test finishes, you're provided with a set of validation results to understand how your Automated Reasoning policy is performing. A test includes the following information:
> Query and Content: A question a user might ask your GenAI application and a possible response. You define these if you manually create the test. Automated Reasoning defines these if you generated test scenarios.
> Confidence threshold: The minimum confidence level for logic validation that you set for your test. This threshold determines how Automated Reasoning handles uncertainty in translating natural language to formal logic. Content that meets or exceeds the threshold is considered a high-confidence finding that can be validated with a definitive result (VALID or INVALID). Content that falls below the threshold is a low-confidence finding that's marked as

[1] https://www.cs.unm.edu/~mccune/prover9/

TRANSLATION_AMBIGUOUS, indicating the system detected ambiguity and chose not to provide a potentially incorrect validation result.
Validation results:
Expected result: The result you expect from running the test.
Actual result: The result from running the test.
Execution result: Indicates whether the test passed. If the expected and actual results align, the test passed. If not, the test failed.
Findings: The output from an Automated Reasoning policy test is a set of findings. Findings represent factual claims contained in your test question and answer. Use these to help you understand why a test passed or failed.
Type: Translations can include a combination of claims and premises.
Premises: Provides context, assumptions, or conditions that affect how a claim should be evaluated. In question-and-answer formats, the premise is often the question itself. Answers can also contain premises that establish constraints or conditions. For example, in the question, "What numbers are divisible by 2?" and answer, "Even numbers", the premise is "numbers divisible by 2". In the statement, "When the traffic light turns green, you must go," the premises is "traffic light is green".
Claims: Factual statements that Automated Reasoning evaluates for accuracy. In a question-and-answer format, the claim is typically the answer. In a standalone statement, the claim is the fact being asserted. For example, in the question, "What numbers are divisible by 2?" and answer, "Even numbers", the claim is "even numbers".
Result: Indicates how valid a finding's claims are. For more information, see Test validation results.
Confidence: The confidence score (ranging from 0.0 to 1.0) that Automated Reasoning has in the translation from natural language to formal logic, representing how certain the system is about correctly interpreting the input text. Higher scores indicate greater certainty in the translation. For example, if a translation has a confidence of "1.0", that indicates maximum certainty that the natural language was accurately converted to formal logic. Lower confidence scores suggest the system has some uncertainty about the translation that you may want to review.
Assignments: Variable assignments from your policy that prove the finding is valid or not. Translations have logic statements that show how the natural language was converted to formal logic. These can be more complex when there is nested logic. For example, hasDogHistoryOfAggression is false.
Rules: The extracted logic from your policy that supports the finding. A test provides you with enough relevant rules from your policy to help you understand the finding result.

The following list details possible validation results from an Automated Reasoning policy test:
VALID The claims in the model's response are logically consistent with your policy rules and can be mathematically proven correct. The response correctly follows all applicable logical constraints and the reasoning from premises to conclusions is sound.

Example: If your policy states "Employees with 1+ year of service get parental leave" and the model responds "You qualify for parental leave since you've worked here for 18 months," this would be VALID because 18 months exceeds the 1-year requirement.

INVALID The claims in the model's response contradict or violate your policy rules. The response contains statements that are mathematically provable as incorrect based on your policy's formal logic constraints.

Example: If your policy states "Employees with 1+ year of service get parental leave" and the model responds "You qualify for parental leave even though you've only worked here for 3 months," this would be INVALID because 3 months doesn't meet the 1-year requirement.

SATISFIABLE Given the information provided in the policy, whether the claims in the model's response are correct or in violation of policy rules depends on additional information that is not specified in the response. Without that additional information, the claims can neither be proven correct nor incorrect.

Example: If your policy states "Employees need 1+ year of service for parental leave AND must submit form HR-101" and the model responds "You qualify for parental leave since you've worked here for 2 years," this would be SATISFIABLE because the response correctly addresses the service requirement but doesn't mention the form requirement (without contradicting it).

IMPOSSIBLE Automated Reasoning can't make a statement about the claims. This can happen if the premises are logically incorrect, or if there is a conflict within the Automated Reasoning policy itself.

Example: If your policy contains contradictory rules like "All employees get vacation days" and "No employees get vacation days," or if the test question contains impossible premises like "What benefits do employees get if they work negative hours?", the result would be IMPOSSIBLE because the logical foundation is flawed.

TRANSLATION_AMBIGUOUS Detected an ambiguity in the translation meant it would be unsound to continue with validity checking. Additional context or follow-up questions might be needed to get translation to succeed.

Example: If your test question is "Can they take leave?" without specifying who "they" refers to, or if the model response uses ambiguous pronouns like "It depends on their situation" without clear referents, the result would be TRANSLATION_AMBIGUOUS because the system cannot reliably translate the vague language into formal logic.

TOO_COMPLEX The input contains too much information for Automated Reasoning to process within its latency limits.

Example: If your test includes an extremely long model response with hundreds of interconnected claims about employee benefits, vacation policies, health insurance, retirement plans, and performance reviews all in a single response, the result might be TOO_COMPLEX because the logical analysis would exceed processing time limits.

NO_TRANSLATIONS Identifies that some or all of the input prompt wasn't translated into logic. This can happen if the input isn't relevant to the Automated Reasoning policy, or if the policy doesn't have variables to model relevant input. If Automated Reasoning can't translate anything, you get a single

NO_TRANSLATIONS finding. You might also see a NO_TRANSLATIONS (along with other findings) if some part of the validation isn't translated.
Example: If your HR policy is designed to validate employee benefits but your test question asks "What's the weather like today?" or "How do I cook pasta?", the result would be NO_TRANSLATIONS because the content is completely unrelated to your policy's domain and variables.
POLICY DOCUMENT: {document}
QUESTION: {question}
ANSWER: {answer}
Based on the document above, classify this question-answer pair into exactly one of the QA validator aggregate results
Analyze the question and answer carefully against the document. Consider: - Does the answer accurately reflect what the document states? - Are there any conditions, exceptions, or edge cases the answer fails to mention? - Is the answer always true, sometimes true, or never true according to the document?
Provide your classification as a single word from the list above in the format <answer>[...]</answer> followed by a brief explanation.
CLASSIFICATION:

Fig. 7: Prompt utilized for LLM-as-Judge (LLMaJ) method.

A.5 Answer Refinement Prompt

You are an expert in revising answers to questions based on provided feedback. Given a domain, a question, an original answer, and an explanation of why the original answer is incorrect, your task is to revise the original answer based on the given explanation. Return only the revised answer without any prefix. Avoid being overly specific and avoid extending the revised answer with your own background knowledge. The revised answer should be consistent with the provided source text.
DOMAIN: {domain}
SOURCE TEXT: {source_text}
QUESTION: {question}
ORIGINAL ANSWER: {original_answer}
FEEDBACK: {feedback}
REVISED ANSWER:

Fig. 8: Prompt template utilized for LLM-based answer refinement.

A.6 LLM-as-Judge Prompt and Outputs for Running Example

For the running example in Sec. 3, we used the following simplified prompt:

System prompt:
You are a logician that validate answer to question based on a given policy
User prompt:
Please consider the following policy:
<policy>
General admission: The regular admission to the park is $50. The admission fee in the low season is 75% of the regular admission fee.
Discount: Seniors (age greater than 65) qualify for 40% discounts. Whenever a discount applies, there will be a $10 flat discount processing fee.
Credit:You can use credit for up to 50% of your final admission (1 credit for 1 dollar). However, if credits are used, then the discount rate is capped at 25%. You can purchase credit at a rate of $0.60 per credit. You can only purchase credit in increments of 5 (cost 3$).
Tax: A federal tax of 10% applies to the final expense.
</policy>
Please help validate the question and answer pair as valid or not.
<question> I am a senior citizen visiting park in the low season, I have $35.4 dollars as total fund, will I be able to visit the park
</question>
<answer> No, $35.4 is not enough </answer>

Fig. 9: LLM-as-Judge Prompt for the example in Sec. 3

Valid.

Reasoning:Let me calculate the cost for a senior citizen visiting the park in the low season:
Base admission price in the low season:
Regular admission: $50 low season rate: 75% of regular = $50 × 0.75 = $37.50
Senior discount:
Seniors get 40% discount = $37.50 × 0.60 = $22.50 Discount processing fee: +$10 Pre-tax amount: $32.50 Tax calculation:
10% federal tax = $32.50 × 0.10 = $3.25 Final cost: $32.50 + $3.25 = $35.75
The visitor has $35.40, which is $0.35 short of the required $35.75.
Using credits wouldn't help because:
With credits, the discount would be capped at 25% (worse than the 40% senior discount) There would be additional costs to purchase the credit.
The answer is valid. With $35.40, the senior citizen does not have enough money to visit the park in the low season.

Fig. 10: LLM-as-Judge Validation (Claude Sonnet 3.7)

Valid.

Reasoning: Looking at this problem, I need to calculate the minimum cost for a senior citizen visiting in the low season.
Let me work through the calculation:
Step 1: Low Season Admission
Regular admission: $50 low season admission: 75% × $50 = $37.50 Step 2: Senior Discount
Seniors get 40% discount Discounted admission: $37.50 × (1 - 0.40) = $37.50 × 0.60 = $22.50 Add discount processing fee: $22.50 + $10 = $32.50 Step 3: Consider Credits Using credits would cap the discount at 25% instead of 40%, making it more expensive, so the optimal choice is NOT to use credits.
Step 4: Apply Tax
Subtotal: $32.50 Federal tax (10%): $32.50 × 1.10 = $35.75 Final Cost: $35.75
Since the visitor has $35.40 and needs $35.75, they are short by $0.35.
Validation Result: VALID
The answer "No, $35.4 is not enough" is correct. The minimum amount needed is $35.75, which exceeds the available funds by $0.35.

Fig. 11: LLM-as-Judge Validation (Claude Opus 4.1)