PolicyLLM: Neuro-Symbolic Policy Extraction and Enforcement for Runtime AI Governance

Syed Ali Haider [1 *] **Benjamin Huh** [1 *] **Hak Hyun Kim** [1 *] **Goutham Nalagatla** [1 *] **Carlos Guerrero Alvarez** [1] **Yash Raj** [1] **Soroush Vosoughi** [1]

Abstract

AI governance frameworks increasingly demand auditable, verifiable evidence that deployed LLMs comply with the policies they operate under, yet the prevailing compliance approaches, fine-tuning, prompt engineering, and retrieval-augmented generation, are opaque, brittle under policy change, and produce no such evidence. We present POLICYLLM, a neuro-symbolic framework that compiles natural-language policy documents into symbolically validated decision graphs and enforces them as a runtime governance layer over LLM outputs. The system pairs neural extraction, SMT-based validation, and hybrid runtime enforcement with a hash-chained audit log designed for regulatory inspection. Our contribution is the enforcement infrastructure: policymakers, regulators, and compliance teams write the policies; PolicyLLM compiles and enforces them. On a policy compliance benchmark, PolicyLLM achieves the highest policy recall (0.72), precision (0.77), and condition F1 (0.33) among ten methods, with zero ungrounded claims observed. Code: `https://anonymous.4open.science/r/PolicyLLM-C03B/`.

1. Introduction

The deployment of large language models within institutions governed by binding policy and regulatory regimes has created a structural mismatch. Organizations and regulators express obligations through deterministic, condition–action rules with explicit exceptions, priorities, and effective dates. LLMs, in contrast, produce probabilistic outputs whose adherence to those obligations is statistical at best and unverifiable in practice. Air Canada was held liable after its chatbot fabricated a bereavement fare policy the company was then required to honor (Civil Resolution Tribunal of British Columbia, 2024). Under the EU AI Act, deployers of high-risk AI systems must implement risk management measures, maintain operational logs, and ensure outputs are interpretable for oversight (European Parliament and Council, 2024). State-level frameworks such as California's SB-53 (California Legislature, 2025) and New York's RAISE Act (New York State Legislature, 2025) signal an emerging norm of standardized safety frameworks, transparency reports, and incident disclosure, a norm that downstream deployer regimes are likely to mirror. Each of these frameworks presupposes technical mechanisms for monitoring, verification, and enforcement that current LLM deployment practice does not reliably provide (Reuel et al., 2025; Brundage et al., 2020).

The literature on AI accountability and auditing makes the gap explicit. Brundage et al. (Brundage et al., 2020) argue that responsible AI development requires *verifiable claims* to which developers can be held accountable; Raji et al. (Raji et al., 2020) introduce an end-to-end internal auditing framework but note that, once a system is deployed, emergent issues become difficult or impossible to trace back to their source; Mökander & Floridi (Mökander & Floridi, 2021) position auditing as a continuous process rather than a one-time check. NIST's AI Risk Management Framework (AI, 2023) similarly emphasizes that AI governance is lifecycle-grounded, not deploy-and-forget. Yet for LLM outputs specifically, the technical primitives that would make any of this auditable at runtime, a per-output record of which policies applied, what facts were checked, and what decision was reached, do not exist as standard infrastructure.

Our position. We see the policy/technical gap as splittable into two distinct problems: *what* the rules should be, and *how* those rules are enforced and evidenced at runtime. The first is a question for policymakers, regulators, legal teams, and domain experts; the second is a question of infrastructure. This paper contributes a concrete answer to the second: **a runtime governance layer that compiles natural-language policy text into a verifiable enforcement artifact, runs that artifact alongside the LLM at**

[*]Equal contribution [1]Department of Computer Science, Dartmouth College, Hanover, NH, USA. Correspondence to: Syed Ali Haider, Hak Hyun Kim <syed.ali.haider.gr@dartmouth.edu, hak.hyun.kim.gr@dartmouth.edu>.

Preprint. May 17, 2026.

inference time, and produces an audit log suitable as regulatory evidence. The intent is for organizations, regulators, and standards bodies to author the policies and pass them to a system like PolicyLLM rather than having to translate each new rule into prompt fragments, fine-tuning data, or bespoke filters.

System. PolicyLLM has three stages: a *neural extraction* module that identifies policy-relevant claims, entities, and conditions from heterogeneous documents; a *symbolic validation* module that compiles these into a logically consistent decision graph using the Z3 SMT solver (de Moura & Bjørner, 2008), with conflicts resolved via an explicit priority lattice; and a *hybrid enforcement* module that constrains generation through scaffold injection and verifies outputs through parallel symbolic and neural checks, escalating uncertain cases to human review. Every enforcement decision is recorded in an append-only, SHA-256-chained log.

Scope. PolicyLLM enforces obligations expressible as condition–action rules with typed variables: regulatory compliance, contractual obligations, privacy constraints, and operational policies. Policies requiring open-ended reasoning or contested interpretation fall outside the formalization; the system flags these for human review by design. We make no claim to solving alignment.

Contributions. (i) A pipeline that compiles natural-language policy into runtime-enforceable decision graphs with provenance and audit trails. (ii) An evaluation against nine baselines: PolicyLLM achieves the highest policy recall (0.72), precision (0.77), and condition F1 (0.33), with zero ungrounded claims observed. (iii) Validation of the full pipeline on a fully open-source stack runnable on a single 24GB GPU, demonstrating the contribution is architectural rather than dependent on proprietary infrastructure.

2. Related Work

Retrieval-Augmented Generation. RAG conditions outputs on retrieved policy documents (Lewis et al., 2020), allowing updates without retraining. However, RAG grounds only the input; the model's synthesis remains probabilistic and may drop constraints, resolve ambiguities incorrectly, or hallucinate qualifications. When retrieved passages contain conflicting conditions, the model lacks any mechanism for logical resolution.

Prompt Engineering and Prompt Injection. Prompt-based compliance relies on the model's capacity to follow natural-language directives, which is neither guaranteed nor verifiable. Adversarial inputs can override system-level instructions (Perez & Ribeiro, 2022), and indirect injection through retrieved content introduces further attack surfaces (Greshake et al., 2023).

Fine-Tuning and RLHF. RLHF and supervised fine-tuning (Ouyang et al., 2022; Bai et al., 2022a) achieve fundamentally statistical alignment: models learn to produce outputs *likely* to satisfy constraints, not outputs *guaranteed* to do so. Reward misspecification can lead to reward hacking (Gao et al., 2023), and neither method supports rapid policy update without retraining.

Guardrails and Constitutional AI. Systems such as NeMo Guardrails (Rebedea et al., 2023) offer determinism through pre- and post-processing filters, but enterprise and regulatory policies involving conditional logic, temporal dependencies, and entity-specific exceptions cannot be captured by flat rules without combinatorial explosion. Constitutional AI (Bai et al., 2022b) has the model revise its own outputs against stated principles, but the same model generates and evaluates. LLM-as-a-judge architectures (Zheng et al., 2023) add a secondary neural check, but one probabilistic system verifying another does not yield deterministic guarantees and does not produce the formal artifact governance regimes require.

Neuro-Symbolic Reasoning and Policy-as-Code. Neural semantic parsing translates natural language into formal representations (Liang, 2016), and SMT solvers such as Z3 (de Moura & Bjørner, 2008) provide constraint verification with formal soundness guarantees. The policy-as-code paradigm, exemplified by Open Policy Agent (Vijayaraghavan, 2025), encodes policies as executable specifications. However, these symbolic approaches see limited adoption in LLM governance because model outputs are unstructured natural language rather than well-typed program states. PolicyLLM bridges this *compilation gap* by translating ambiguous policy language into formal representations that are both faithful and maintainable.

AI Auditing and Accountability. A growing body of work positions auditing as the central mechanism through which AI systems are held accountable (Mökander & Floridi, 2021; Mökander et al., 2021; Raji et al., 2020). Anderljung et al. (Anderljung et al., 2023) extend this argument to frontier models, calling for pre-deployment risk assessment and continuous post-deployment monitoring. PolicyLLM contributes a concrete substrate for the runtime portion of these proposals: the per-output enforcement record is precisely the kind of evidence such auditing regimes require but rarely have access to in current LLM deployments.

3. System Description

PolicyLLM operates as a three-module pipeline (Figure 1). An offline phase transforms raw policy documents into a

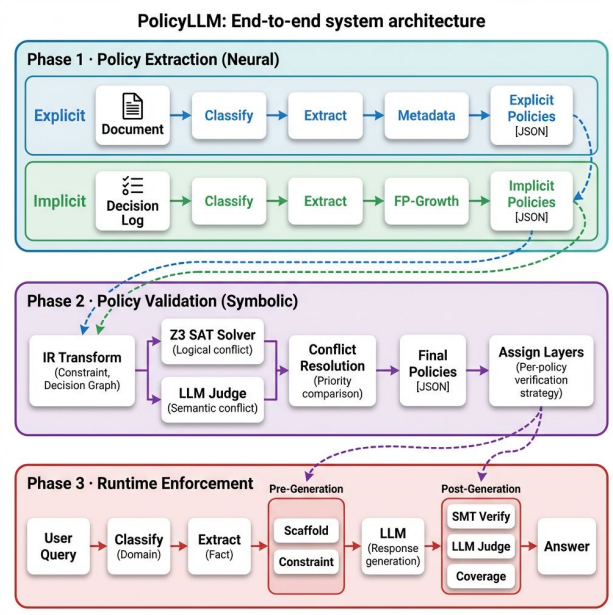

Figure 1. PolicyLLM pipeline. Offline, raw policy documents are extracted and compiled into a verified decision graph. At inference time, the enforcement module constrains generation through scaffold injection, verifies outputs through parallel checks, and writes every decision to a hash-chained audit log.

compiled bundle through normalization (§3.1) and SMT-based validation (§3.2). At inference time, the enforcement module (§3.3) consumes the bundle alongside each query, guides generation through scaffold injection, and verifies outputs through parallel symbolic and neural checks. Policy updates require only recompilation of the bundle, not retraining of the LLM.

3.1. Data Extraction

The data extraction module transforms heterogeneous policy documents into structured, schema-valid policy records suitable for downstream formalization and enforcement. Organizations typically maintain policies across disparate formats, so the first objective is to normalize inputs while preserving structure and provenance.

The pipeline begins with document regularization. The system identifies each input format and routes it to the appropriate parser: text-based PDFs are read directly, scanned PDFs are processed through OCR when configured, and DOCX, HTML, Markdown, and plain text files are parsed with format-specific readers that preserve headings, paragraphs, and tables. The output of this stage is a canonical document object annotated with section boundaries, page references, and character spans, ensuring traceability from any extracted policy content back to its exact source location.

Each section then passes through a multi-pass extraction flow. The first pass applies a relevance filter to distinguish substantive policy content from supporting material such as introductions or definitions. The second pass performs structured component extraction, producing schema-valid JSON representations of scope, conditions, actions, and exceptions. The third pass enriches each candidate with entity-level signals including dates, monetary values, percentages, and temporal expressions, using regex-based extraction for speed and consistency with model-based fallback for ambiguous text. By this stage, each candidate policy carries a well-defined rule structure grounded in source evidence.

Following section-level extraction, a document-level consolidation step merges overlapping policies into deduplicated rule-level records, combining source spans, provenance trails, and evidence counts rather than discarding them. The pipeline then enriches each merged policy with governance metadata such as owner, effective date, domain classification, and regulatory linkage. Pattern-based extraction handles the majority of metadata fields, with language model inference reserved for cases where document language is insufficiently structured. When source documents are incomplete, domain and tenant defaults fill gaps, and a final validation step checks consistency, raises review flags for high-risk records, and stores confidence signals for downstream use.

The output is stored in JSONL format with one policy per line, accompanied by an index file reporting policy totals, domain distribution, and the proportion of flagged items. The approach combines deterministic structure with targeted model reasoning: outputs remain stable through schema validation while still capturing policy language that purely rule-based methods would miss.

3.2. Policy Validation

The validation module transforms extracted policy representations into an enforcement-ready compiled bundle through four stages combining symbolic reasoning via the Z3 SMT solver (de Moura & Bjørner, 2008) with semantic analysis using LLM embeddings.

Stage 1: Policy IR Construction. The first stage converts extracted policy JSON into an intermediate representation (IR) that supports downstream symbolic reasoning. Each policy condition is mapped to a typed variable through deterministic rewrite rules. For example, a condition of type `time_window` maps to an integer variable `days_since_purchase`, while a `boolean_flag` maps to a boolean variable `has_receipt`. These typed representations establish the variable vocabulary and priority signals used in subsequent stages.

Stage 2: Decision Graph Construction. The second stage constructs an ordered Directed Acyclic Graph (DAG) over the IR-annotated policies, where each policy corresponds to an ordered path through the graph. Variable ordering within the DAG follows a priority criterion: type priority (boolean, then enumerated, then numeric), frequency of occurrence across policies, and alphabetical order as a tiebreaker. This ordering ensures that the most discriminative conditions are evaluated earliest, producing a compact decision structure amenable to efficient conflict analysis.

Stage 3: Z3-Based Conflict Detection. The third stage identifies logical contradictions between compiled policy paths using satisfiability checking through the Z3 theorem prover. Each pair of potentially conflicting paths is encoded as a conjunction of constraints and tested for satisfiability; a satisfiable result indicates an input configuration under which two policies yield contradictory outcomes. Unlike LLM-based conflict detection, which can miss violations and lacks formal explainability, symbolic satisfiability checking provides deterministic guarantees: every reported conflict is a provably valid logical contradiction, and the solver returns a witness assignment that serves as an interpretable explanation.

Stage 4: Priority Resolution and Bundle Compilation. The final stage resolves detected conflicts using the priority ordering established in Stage 2, determining which policy takes precedence under each conflicting input configuration. Once all conflicts are resolved, the pipeline merges the validated decision graph, priority assignments, conflict resolution records, and variable metadata into a single compiled policy bundle. This bundle serves as the enforcement-ready artifact consumed by the runtime module, providing a complete and logically consistent specification against which LLM outputs can be verified.

3.3. Policy Enforcement

Given a compiled policy bundle produced by the upstream validation stage and an incoming user query, the enforcement module determines whether the LLM-generated response complies with all applicable organizational policies. The module produces a composite compliance score, a routing action (pass, auto-correct, regenerate, or escalate), and a set of identified violations. Enforcement proceeds through three temporally distinct phases followed by scoring and action routing.

Pre-Generation. The system constructs an enforcement context by classifying the user query into a domain and intent via an LLM-based semantic classifier, then retrieving applicable rules, decision-graph paths, and constraints from a pre-built index over the compiled bundle. We use neural classification rather than keyword matching to ensure robustness to paraphrasing and synonymy. Temporal filtering excludes rules whose effective dates have not yet been reached. When retrieved rules conflict, a two-tier dominance resolution procedure applies: explicit override entries in a dominance table are consulted first, followed by a priority lattice with default ordering regulatory, core values, company, department, and situational, where earlier classes take precedence.

Scaffold Injection. Rather than relying solely on post-hoc verification, the system injects policy-aware structure directly into the generation prompt. An invariant block translates each constraint into a natural-language directive: negation constraints render as `NEVER` directives, affirmative constraints as `ALWAYS` directives. A reasoning scaffold converts the compiled decision-graph paths into a sequence of mandatory reasoning steps ordered by the topological sort encoded in the bundle. For each variable, the scaffold emits a typed instruction: boolean variables require explicit verification, enum variables enumerate the permissible values, and numeric variables require a value check. Conditional branches extracted from the paths are appended with provenance metadata citing the source document, effective date, and policy identifier. This structured injection enforces chain-of-thought reasoning aligned with the policy decision graph.

Post-Generation Verification. After the LLM produces a candidate response, four verification signals are computed in parallel. A *regex safety gate* checks for forbidden patterns including PII formats and unconditional commitment language such as "I guarantee" or "we will definitely." The regex gate operates as a hard gate: any match triggers immediate escalation regardless of all other scores, encoding the design principle that safety constraints are inviolable.

An *SMT verification* pass extracts factual assertions from the response using a hybrid strategy. The primary pass is symbolic: type-specific regular expressions identify boolean assertions, integer and float values, and enum memberships against the schema. If the symbolic pass covers fewer than half of the schema variables, a neural fallback invokes the LLM to extract remaining facts, merging the results while giving precedence to symbolically extracted values. The extracted facts are then encoded as Z3 constraints and checked for satisfiability against each applicable rule; constraint violations and uncovered decision paths are recorded (Algorithm 1).

A *coverage analysis* measures the fraction of required decision variables explicitly addressed in the response. Let N_{req} denote the set of variables required by the applicable paths

and N_{cov} the subset mentioned in the response:

$$s_C = \begin{cases} \frac{|N_{\text{cov}}|}{|N_{\text{req}}|} & \text{if } N_{\text{cov}} = N_{\text{req}}, \\ 0.8 \cdot \frac{|N_{\text{cov}}|}{|N_{\text{req}}|} & \text{otherwise,} \end{cases} \tag{1}$$

applying a 20% penalty for incomplete coverage to incentivize thoroughness.

A *judge LLM* evaluates the response for semantic compliance along five axes: factual accuracy with respect to the policy rules, action compliance, constraint adherence, tone and implication, and completeness of decision-step coverage. The judge returns a score $s_L \in [0, 1]$ at temperature 0.0 for deterministic evaluation. If the judge LLM is unavailable, the system defaults to a neutral score of 0.5.

Scoring and Action Routing. The verification signals are aggregated into a composite compliance score:

$$\hat{s} = w_Z \cdot s_Z + w_L \cdot s_L + w_C \cdot s_C, \tag{2}$$

with weights $w_Z = 0.60$, $w_L = 0.30$, $w_C = 0.10$. The regex gate is deliberately excluded from the weighted sum and instead functions as a hard override: if any forbidden pattern is matched, the action is unconditionally set to escalate, ensuring that safety-critical violations such as PII leakage cannot be overridden by high scores on other dimensions.

For responses that pass the regex gate, the routing action is determined by threshold:

$$\alpha = \begin{cases} \text{PASS} & \text{if } \hat{s} \geq 0.95, \\ \text{AUTO\_CORRECT} & \text{if } 0.85 \leq \hat{s} < 0.95, \\ \text{REGENERATE} & \text{if } 0.70 \leq \hat{s} < 0.85, \\ \text{ESCALATE} & \text{if } \hat{s} < 0.70. \end{cases} \tag{3}$$

When the initial action is auto-correct or regenerate, the system enters a bounded retry loop with escalating strictness. For auto-correction, violation descriptions are appended to the prompt as explicit correction hints and the response is regenerated once. For regeneration, the scaffold is tightened by injecting explicit prohibitions derived from the identified violations, and the response is regenerated up to two additional times. If all retries are exhausted without reaching a passing score, the system escalates to human review, bounding the total number of LLM calls per query to four. Every enforcement decision is recorded in an append-only JSONL log secured by a SHA-256 hash chain, providing a verifiable audit trail suitable for regulatory compliance.

4. Experiments and Results

We evaluate PolicyLLM against nine baselines spanning purely neural, purely symbolic, and hybrid approaches. Robustness experiments addressing model dependence, embedding choice, and external benchmark transfer are summarized briefly here and detailed in Appendix B.

4.1. Experimental Setup

Benchmark. We evaluate on three tasks derived from real policy documents spanning refund (Zara T&C), privacy (HIPAA), HR compliance (NICE), and finance (YC SAFE). The extraction benchmark contains four documents with 27–29 expert-annotated reference policies each (≈80 unique policies). The enforcement benchmark comprises 24 scenarios (20 general, 2 PII, 2 guarantee-language). The hallucination benchmark contains 30 manually authored scenarios with known grounded and hallucinated claims. All evaluation is test-only; no model training is performed.

Models and Parameters. Main results use GPT-4o-mini at temperature 0.0 and max tokens $2{,}048$ for generation, with the same model serving as judge. RAG baselines retrieve via cosine similarity over `all-MiniLM-L6-v2` embeddings (top-k=10 extraction, top-k=5 enforcement). Few-shot baselines use two fixed exemplars. Alternative backbones (Qwen3-30B-A3B), judges (Mixtral-8x7B-Instruct), and embedders (BGE-large-en-v1.5, text-embedding-3-small) are reported in Appendix B.

Metrics. Policy recall and precision are computed via greedy best-match with Jaccard similarity ($\geq$0.10) over tokenized policy fields. Condition F1 is micro-averaged at the condition level after canonicalizing condition keys to type and operator. Compliance is binary per scenario: correct if the system's routing action matches the expected outcome (pass or escalate). Hallucination rate is the fraction of responses flagged by both a token-overlap heuristic and an LLM-as-judge assessment. All neural methods use temperature 0.0 for reproducibility; extraction and enforcement results are single-run, while hallucination metrics include bootstrap 95% confidence intervals over 1,000 resamples.

4.2. Baselines

We compare three categories of methods. *Neural-only*: vanilla LLM (no policy context), system prompt injection, and few-shot prompting. *Retrieval-augmented*: RAG retrieval over the policy corpus and a RAG + Z3 hybrid. *Symbolic and rule-based*: keyword overlap with rules, semantic similarity with rules, SMT-only (Z3 without neural extraction), and LLM zero-shot conflict detection.

4.3. Main Results

PolicyLLM achieves the highest policy recall (0.72), precision (0.77), and condition F1 (0.33), with gains over the strongest baseline (RAG + Z3) of +0.25 recall, +0.04 preci-

Method	P-Rec	P-Prec	C-F1	Comp.	Lat.
Vanilla LLM	0.06	0.20	0.24	83.3	0.58
Sys. prompt inj.	0.53	0.75	0.25	**100.0**	0.77
Few-shot	0.25	0.55	0.32	95.8	0.68
RAG only	0.46	0.62	0.27	95.8	0.80
Keyword + rules	0.12	0.16	0.13	**100.0**	**0.03**
Sem. sim. + rules	0.12	0.16	0.13	**100.0**	0.06
SMT-only (Z3)	0.12	0.16	0.13	83.3	0.09
LLM zero-shot	0.45	0.58	0.27	54.2	1.21
RAG + Z3 hybrid	0.47	0.73	0.27	95.8	0.70
PolicyLLM (Ours)	**0.72**	**0.77**	**0.33**	91.7	2.44

Table 1. Policy compliance evaluation. P-Rec: policy recall; P-Prec: policy precision; C-F1: condition-level F1; Comp.: compliance rate (%); Lat.: mean latency (s). Best in bold.

Method	Hallucination Rate
Vanilla LLM	0.549
LLM zero-shot	0.542
Sys. prompt inj.	0.535
RAG only	0.444
RAG + Z3 hybrid	0.382
Few-shot	0.333
SMT-only (Z3)	0.201
Sem. sim. + rules	0.201
Keyword + rules	0.201
PolicyLLM (Ours)	**0.000**

Table 2. Hallucination rate (fraction of responses containing at least one ungrounded claim).

sion, and +0.06 condition F1 (Table 1). Condition F1 of 0.33 is both the headline result on this metric and the primary current weakness; the absolute number reflects the difficulty of fine-grained extraction from heterogeneous policy text. The architecture is designed so that extraction errors do not propagate into unsafe outputs: symbolic validation rejects logically inconsistent rules pre-deployment, and runtime verification escalates outputs whose facts cannot be discharged against the bundle. Empirically this fail-safe property holds: despite F1 of 0.33, PolicyLLM achieves zero ungrounded claims and the highest precision. PolicyLLM's compliance rate of 91.7% falls below baselines achieving 95.8–100%; the system surfaces all applicable conditions and routes uncertain cases to human review rather than silently omitting policy-sensitive content. Latency (2.44s) exceeds baselines but remains consistent with interactive deployment; the dominant cost is LLM calls.

4.4. Hallucination Analysis

Hallucination rates decrease with the degree of structural grounding (Table 2). Neural-only methods exhibit the highest rates (0.53–0.55); retrieval reduces but does not eliminate hallucination (RAG: 0.44; RAG + Z3: 0.38); rule-based methods cluster at 0.20. PolicyLLM achieves zero hallucination on this benchmark: scaffold injection constrains generation to the extracted policy IR, and post-generation verification blocks any ungrounded assertion. We note this result is benchmark-specific and does not constitute a deployment guarantee, but it demonstrates that neuro-symbolic verification substantially reduces ungrounded claims relative to all alternatives.

4.5. Robustness

We summarize Appendix B. Replacing the judge with Mixtral-8x7B-Instruct and the backbone with Qwen3-30B-A3B preserves method rankings: PolicyLLM retains the highest precision and remains the only method with non-zero conflict detection. Substituting embedders (BGE-large-en-v1.5, text-embedding-3-small) does not close the gap, indicating gains derive from structured extraction and symbolic verification rather than embedding quality. The full pipeline runs end-to-end on Qwen3-30B-A3B locally on a single 24GB GPU at int8, with no API calls. On external benchmarks, PolicyLLM transfers strongly to structured policy tasks (up to +0.14 on privacy entailment, $\approx 2\times$ precision on CUAD-style clause extraction) and weakly on open-ended legal reasoning (ContractNLI, Unfair-ToS), consistent with the scope statement in §1.

5. Policy Implications: Runtime Governance Infrastructure

The above results matter not only for the LLM compliance literature but for the broader project of operationalizing AI governance (Reuel et al., 2025). The remainder of this section articulates how PolicyLLM connects to specific functions that recent governance frameworks demand.

From principles to runtime artifacts. A persistent finding in the AI governance literature is that high-level principles, fairness, transparency, accountability, have not translated into deployable mechanisms. Mökander & Floridi (Mökander & Floridi, 2021) call for ethics-based auditing as a continuous process rather than a one-time check; Mökander et al. (Mökander et al., 2021) stress that auditing must intervene throughout the lifecycle, not only after deployment. PolicyLLM operationalizes these prescriptions for the LLM setting: the compiled policy bundle is a machine-checkable specification, the runtime verification pass is the continuous audit, and the hash-chained log is the lifecycle artifact. The design supports the central premise of the policy-as-code movement (Vijayaraghavan, 2025) extended to unstructured-output systems.

Verifiable claims, made on every output. Brundage et al. (Brundage et al., 2020) argue that responsible AI development requires *verifiable claims* to which deployers can be held accountable. The ten mechanisms surveyed in that report span institutions, software, and hardware; PolicyLLM contributes one concrete software-level mechanism

for the deployment-time portion of this agenda. For each LLM output, the audit log records which policies applied, which facts were extracted from the response, and which constraints were satisfied or violated. A regulator, third-party auditor, or internal compliance officer can interrogate any past output and reconstruct the compliance reasoning, a capability that statistical alignment does not provide.

Closing the deployment-time accountability gap. Raji et al. (Raji et al., 2020) introduce SMACTR for internal algorithmic auditing but observe that, once a system is deployed, emergent issues become difficult or impossible to trace back to their source. PolicyLLM is designed precisely for that traceability: the per-output record makes the chain from query, to applicable policies, to extracted facts, to verification outcome, to routing decision explicit and replayable. The system thus complements pre-deployment auditing frameworks by providing the runtime logging substrate they typically presuppose but do not implement.

Conformity assessment under risk-based regulation. The EU AI Act (European Parliament and Council, 2024) requires that high-risk systems undergo conformity assessment before deployment, including verification that the system complies with applicable harmonized standards. The NIST AI Risk Management Framework (AI, 2023) structures risk management around four functions, govern, map, measure, manage, with the latter three requiring measurable evidence. PolicyLLM's symbolic validation produces precisely this evidence at the policy level: every conflict reported by the SMT solver is accompanied by a witness assignment, an interpretable input under which two policies yield contradictory outcomes. This is the form of explainable, auditable evidence conformity assessors require, and that purely neural methods cannot provide.

Deployer obligations and the policy/technical interface. The framing of governance obligations as falling on *deployers* (EU AI Act Article 26 onward) and on *frontier developers* (SB-53, RAISE Act) places different but related demands on technical infrastructure. Anderljung et al. (Anderljung et al., 2023) identify pre-deployment risk assessment, external scrutiny of behavior, deployment decisions informed by risk assessment, and post-deployment monitoring as core regulatory levers. PolicyLLM addresses the runtime portion of this stack and intentionally does not address the others. Our position is that the technical AI governance ecosystem benefits from a clear division of labor: regulators and policymakers specify what counts as compliance, evaluation researchers measure it pre-deployment, and runtime systems like PolicyLLM enforce and log it at inference time. The audit log connects these layers, it is the artifact a third-party auditor, supreme audit institution (AI, 2023), or regulator can examine to bridge from policy to behavior.

What policymakers can rely on. The most consequential implication is design-prescriptive rather than descriptive. If a regulatory regime can specify obligations as condition–action rules with typed variables and explicit priorities, as is true of most procedural regulation, contractual obligations, and operational policies, then a system like PolicyLLM can compile, enforce, and log compliance against them automatically. We therefore offer the framework as a general-purpose substrate: *write your policy in natural language; PolicyLLM will compile it into an enforcement artifact, run it alongside the LLM, and produce an evidentiary record for every output.* This division of labor invites policymakers, standards bodies, and compliance teams to author the rules at whatever level of granularity their regime requires, with the assurance that downstream enforcement and audit are infrastructural rather than per-deployment engineering work.

Limits of the approach for governance use. We are explicit that PolicyLLM does not address subjective or open-ended normative judgments, and does not replace the broader pre-deployment evaluation work emphasized by the model-evaluation literature (Anderljung et al., 2023; Brundage et al., 2020). It is one runtime layer in a stack, not a complete governance solution. The audit log it produces is only as useful as the policies it compiles; a regime that fails to specify obligations clearly will not be rescued by better enforcement infrastructure.

6. Limitations

The primary benchmark is derived from a single curated corpus; transfer experiments on ContractNLI, LegalBench, and CUAD (Appendix B) provide partial generalization evidence but not full coverage of cross-jurisdictional or temporally evolving regimes. The zero-hallucination result is benchmark-specific and not a deployment guarantee. Condition F1 of 0.33 indicates fine-grained extraction remains the principal weakness; targeted improvements (operator-aware few-shot prompting, length-sensitive deduplication) are under investigation. Mean latency (2.44s) is dominated by LLM inference rather than symbolic components and admits standard amortization. PolicyLLM assumes condition–action formalizability; obligations requiring open-ended reasoning are routed to human review by design.

7. Conclusion

We presented PolicyLLM, a runtime governance layer that compiles natural-language policy documents into symbolically validated decision graphs and enforces them over LLM outputs with auditable evidence. PolicyLLM achieves

the highest policy recall (0.72), precision (0.77), and condition F1 (0.33) among ten methods, with zero ungrounded claims observed; the full pipeline runs on open-source models on a single 24GB GPU. Our position is that the technical AI governance gap is best closed not by a single monolithic compliance system but by a clear division of labor: policymakers specify obligations, evaluation researchers measure pre-deployment, and runtime systems like PolicyLLM enforce and log compliance at inference time. Code and data are available at `https://anonymous.4open.science/r/PolicyLLM-C03B/`.

References

AI, N. Artificial intelligence risk management framework (ai rmf 1.0). *URL: https://nvlpubs. nist. gov/nistpubs/ai/nist. ai*, pp. 100–1, 2023.

Anderljung, M., Barnhart, J., Korinek, A., Leung, J., O'Keefe, C., Whittlestone, J., Avin, S., Brundage, M., Bullock, J., Cass-Beggs, D., et al. Frontier ai regulation: Managing emerging risks to public safety. *arXiv preprint arXiv:2307.03718*, 2023.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a. URL `https://doi.org/10.48550/arXiv.2204.05862`.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukosuite, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T., Hume, T., Bowman, S. R., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T., and Kaplan, J. Constitutional ai: Harmlessness from ai feedback, 2022b. URL `https://arxiv.org/abs/2212.08073`.

Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., Khlaaf, H., Yang, J., Toner, H., Fong, R., Maharaj, T., Koh, P. W., Hooker, S., Leung, J., Trask, A., Bluemke, E., Lebensold, J., O'Keefe, C., Koren, M., Ryffel, T., Rubinovitz, J., Besiroglu, T., Carugati, F., Clark, J., Eckersley, P., de Haas, S., Johnson, M., Laurie, B., Ingerman, A., Krawczuk, I., Askell, A., Cammarota, R., Lohn, A., Krueger, D., Stix, C., Henderson, P., Graham, L., Prunkl, C., Martin, B., Seger, E., Zilberman, N., hÉigeartaigh, S. , Kroeger, F., Sastry, G., Kagan, R., Weller, A., Tse, B., Barnes, E., Dafoe, A., Scharre, P., Herbert-Voss, A., Rasser, M., Sodhani, S., Flynn, C., Gilbert, T. K., Dyer, L., Khan, S., Bengio, Y., and Anderljung, M. Toward trustworthy ai development: Mechanisms for supporting verifiable claims, 2020. URL `https://arxiv.org/abs/2004.07213`.

California Legislature. Senate bill 53: Transparency in frontier artificial intelligence act, 2025. Signed into law on Sept. 29, 2025 by Gov. Gavin Newsom; authored by Sen. Scott Wiener.

Civil Resolution Tribunal of British Columbia. Moffatt v. air canada. `https://www.canlii.org/en/bc/bccrt/doc/2024/2024bccrt149/2024bccrt149.html`, 2024. 2024 BCCRT 149 (CanLII), Decision No. CRT 149/2024.

de Moura, L. and Bjørner, N. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

European Parliament and Council. Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence (artificial intelligence act). `https://eur-lex.europa.eu/eli/reg/2024/1689/oj`, 2024. OJ L 2024/1689, 12 July 2024.

Gao, L., Schulman, J., and Hilton, J. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023. URL `https://doi.org/10.48550/arXiv.2210.10760`.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pp. 79–90. ACM, 2023. URL `https://arxiv.org/abs/2302.12173`.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33,

pp. 9459–9474, 2020. URL https://arxiv.org/abs/2005.11401.

Liang, P. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76, 2016.

Mökander, J. and Floridi, L. Ethics-based auditing to develop trustworthy ai. *Minds and Machines*, 31(2):323–327, 2021.

Mökander, J., Morley, J., Taddeo, M., and Floridi, L. Ethics-based auditing of automated decision-making systems: Nature, scope, and limitations. *Science and Engineering Ethics*, 27(4):44, 2021.

New York State Legislature. Responsible ai safety and education (raise) act (s6953b/a6453b), 2025. Signed Dec. 19, 2025; chapter amendment signed Mar. 27, 2026; effective Jan. 1, 2027; establishes an oversight office within the New York Department of Financial Services.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022. URL https://arxiv.org/abs/2203.02155.

Perez, F. and Ribeiro, I. Ignore previous prompt: Attack techniques for language models, 2022. URL https://arxiv.org/abs/2211.09527.

Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., Smith-Loud, J., Theron, D., and Barnes, P. Closing the ai accountability gap: defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20, pp. 33–44, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369367. doi: 10.1145/3351095.3372873. URL https://doi.org/10.1145/3351095.3372873.

Rebedea, T., Dinu, R., Sreedhar, M. N., Parisien, C., and Cohen, J. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In Feng, Y. and Lefever, E. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 431–445, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.40. URL https://aclanthology.org/2023.emnlp-demo.40/.

Reuel, A., Bucknall, B., Casper, S., Fist, T., Soder, L., Aarne, O., Hammond, L., Ibrahim, L., Chan, A., Wills, P., Anderljung, M., Garfinkel, B., Heim, L., Trask, A., Mukobi, G., Schaeffer, R., Baker, M., Hooker, S., Solaiman, I., Luccioni, A. S., Rajkumar, N., Moës, N., Ladish, J., Bau, D., Bricman, P., Guha, N., Newman, J., Bengio, Y., South, T., Pentland, A., Koyejo, S., Kochenderfer, M. J., and Trager, R. Open problems in technical ai governance, 2025.

Vijayaraghavan, S. K. J. Policy as code: A paradigm shift in infrastructure security and governance. In *International Conference of Global Innovations and Solutions*, pp. 182–193. Springer, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

A. Neuro-Symbolic SMT Verification

Algorithm 1 Neuro-Symbolic SMT Verification

Require: Response r, rules R, decision paths G, constraints K, schema variables V
Ensure: $\langle \textit{passed}, \textit{violations}, \textit{score} \rangle$
Extract facts F from r via type-specific pattern matching
if $|F| < 0.5 \cdot |V|$ **then**
    Augment F with LLM-based extraction {Neural fallback}
end if
for each rule $\rho \in R$ **do**
    Encode F and ρ.conditions as Z3 constraints; check satisfiability
end for
for each constraint $k \in K$ **do**
    If prohibited fact appears in F, record violation
end for
Verify F satisfies at least one complete path in G
return $\langle \textit{no violations}, \textit{violation list}, \textit{score} \rangle$ =0

B. Robustness Experiments

B.1. Cross-Family Models

We replaced the GPT-4o-mini judge with Mixtral-8x7B-Instruct-v0.1 and the backbone with Qwen3-30B-A3B, eliminating model-family overlap between generation and evaluation. PolicyLLM retains the highest precision (0.28, $\approx 2\times$ next-best) and remains the only evaluated method with non-zero conflict detection (F1 = 1.0 on conflict instances). Method rankings are preserved.

B.2. Embedding Sensitivity

Replacing `all-MiniLM-L6-v2` with BGE-large-en-v1.5 and text-embedding-3-small does not close the gap between baselines and PolicyLLM. The contribution derives from structured extraction and symbolic verification, not embedding quality.

B.3. Open-Source Deployment

We validated the full pipeline using Qwen3-30B-A3B as the sole language model (backbone, judge, neural fallback), running locally on a single 24GB GPU at int8 with no external API calls. All key system properties persist: highest extraction precision, unique non-zero conflict detection, and full end-to-end functionality including symbolic verification, scaffold injection, audit logging, and escalation.

B.4. External Benchmark Transfer

On ContractNLI, LegalBench, and CUAD, transfer is consistent with the scope statement in §1: gains of up to +0.14 accuracy on privacy entailment and $\approx 2\times$ precision on CUAD-style clause extraction; performance on open-ended reasoning subtasks (ContractNLI, Unfair-ToS) is comparable to baselines.

C. Synthetic Data Generation and Evaluation Pipeline

C.1. Ground-Truth Constitution and Documents

We generate a ground-truth constitution (10–50 policies configurable) using an LLM, spanning refund, privacy, security, shipping, and returns, each annotated with priority, owner, domain, effective date, and exceptions. Four document regimes are produced: **Stage 1 (Explicit)** formal clause-style statements; **Stage 2 (Conflicts)** Stage 1 plus K low-priority conflicting statements to exercise SMT detection; **Stage 3 (Implicit)** narrative and hedged language requiring inference; **Stage 4 (Mixed)** 50% explicit, 15% conflict-seeded, 25% implicit, 10% hybrid.

C.2. Queries and Enforcement Tests

Stage 4 adds synthetic queries with labels `valid_path`, `violation`, `uncovered`, `edge_case`, enabling end-to-end PASS/ESCALATE evaluation rather than extraction-only metrics.

C.3. Evaluation Metrics

Extraction is measured with policy-level recall/precision, condition F1, action accuracy, and exception accuracy. Conflict handling is scored by detection and priority correctness. Enforcement uses PASS/ESCALATE accuracy, FP/FN rates, and compliance score calibration.

C.4. Observed Synthetic Results

Table 3 shows perfect policy and action recovery across all four stages, with condition accuracy 71–83% and exception accuracy 50–150%. Conflict seeding in Stage 2 did not reduce extraction fidelity, suggesting the pipeline isolates extraction from downstream SMT checks. Table 4 highlights mixed-regime failure modes: domains collapsed to "unknown," equality/inequality operators were mis-tagged, and short policies were merged into medium-length ones.

C.5. Reproducibility

The synthetic suite is deterministic given seeds and model versions. Workflow: generate constitution $\rightarrow$ generate stage-specific documents $\rightarrow$ run extraction $\rightarrow$ compile bundle

Table 3. Synthetic extraction pipeline performance.

Metric	Explicit	Conflicts	Implicit	Mixed
Policies Encoded	4	4	4	4
Policies Extracted	4	4	4	4
Extraction Accuracy	100%	100%	100%	100%
Conditions Extracted/GT	6/5	7/5	6/5	6/5
Condition Accuracy	83%	71%	83%	83%
Actions Extracted/GT	4/4	4/4	4/4	4/4
Action Accuracy	100%	100%	100%	100%
Exceptions Extracted/GT	3/2	3/2	1/2	3/2

Table 4. Stage 4 mixed sensitivity analysis.

Dimension	Category	GT	Extracted	Accuracy
Policy Domain	Privacy	1	0	0%
	Returns	1	0	0%
	Security	1	0	0%
	Shipping	1	0	0%
	Unknown	0	4	100%
Complexity	Moderate	1	2	100%
	Simple	3	2	67%
Exceptions	No Exceptions	2	1	50%
	With Exceptions	2	3	100%
Operators	==	3	0	0%
	¿=	1	0	0%
	Unknown	0	4	100%

with SMT conflict checks → (Stage 4) run enforcement on labeled queries → score extraction, conflict resolution, compliance. All artifacts are saved per stage to allow ablations and replays.