Prompt Contracts: A Probabilistic Specification Language for Testing and Validating Large Language Model Outputs

Philippos Melikidis
Independent Researcher
Bietigheim-Bissingen, Germany
philipp.melikidis@gmail.com

Abstract

Large Language Models (LLMs) function as stochastic, untyped interfaces lacking formal specifications. We introduce PCSL v0.3, the first probabilistic contract language for LLM prompt testing with statistical validation. Through N-sampling with configurable aggregation (majority/all/any) and bootstrap confidence intervals, PCSL quantifies reliability across 5 tasks (classification, extraction, summarization, RAG, tool-calls) totaling 1,247 fixtures. Evaluation demonstrates 100% validation success with enforce mode (0% repair), 92% with assist (68% repair vs. 12% no-validation baseline). Bootstrap 95% CIs confirm statistical significance. Semantic checks (similarity, LLM-as-judge) extend beyond structural validation. PCSL operationalizes compliance-as-code: artifact types map to ISO/IEC/IEEE 29119 test documentation, execution modes satisfy EU AI Act Articles 9-15. Implementation overhead: <5ms validation, 2.1× latency for N=5 sampling. One-command reproduction via Docker (`make eval-full`) achieves deterministic results with seed=42.

Keywords

Large Language Models, Prompt Engineering, Software Testing, Probabilistic Contracts, Compliance

1 Introduction

Large Language Models function as *untyped, stochastic interfaces*: prompts map inputs to probabilistic outputs without formal behavioral guarantees [1]. Consider an LLM as $f_\theta : \mathcal{X} \rightarrow P(\mathcal{Y})$ where θ are learned parameters and $P(\mathcal{Y})$ denotes a probability distribution over outputs. Unlike deterministic APIs with explicit contracts (e.g., `classify: String → {A, B, C}`), LLM outputs vary across runs, making traditional contract testing insufficient.

This gap becomes critical as LLMs deploy in regulated domains [3]. The EU AI Act mandates transparency, auditability, and robustness testing for high-risk AI systems. Yet prompt engineering lacks the specification infrastructure available to traditional software: no type checking, no contract enforcement, no systematic validation with statistical confidence.

Research Problem. How can we define, validate, and enforce behavioral contracts for probabilistic LLM interfaces while ensuring reproducibility and regulatory compliance?

Contributions. This work presents:

(1) **Probabilistic specification language**: PCSL v0.3 with N-sampling, aggregation policies (majority/all/any/first), and bootstrap confidence intervals for statistical validation (Section 3).
(2) **Expanded evaluation**: Five tasks (1,247 fixtures) with structural and semantic checks, demonstrating 92% validation success (assist) vs. 12% baseline, with 95% CI [0.89, 0.94] (Section 5).
(3) **Compliance framework**: Formal mapping of PCSL artifacts to ISO 29119 test documentation and EU AI Act requirements, operationalizing compliance-as-code (Section 4.4).

Conference'17, Washington, DC, USA
2025. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: LLM Testing Frameworks Comparison

Framework	Contracts	Multi-Prov.	CI/CD	Semantic	Probabilistic
HELM [6]	×	✓	×	×	×
LangChain [2]	×	✓	×	×	×
Guidance [8]	×	Limited	×	×	×
OpenAI Struct. [9]	Partial	×	×	×	×
CheckList [11]	Manual	✓	Partial	×	×
PCSL v0.3 (ours)	✓	✓	✓	✓	✓

2 Related Work

2.1 Contract-Based Testing

Design-by-contract [7] formalizes software specifications through preconditions, postconditions, and invariants for deterministic functions. PCSL extends this to probabilistic functions by (1) N-sampling to estimate satisfaction probability, (2) aggregation policies to handle output variance, and (3) tolerance thresholds τ for statistical validation.

OpenAPI [10] provides machine-readable REST API contracts. PCSL adopts this artifact-based design: Prompt Definitions (PD), Expectation Suites (ES), and Evaluation Profiles (EP) serve as versioned, provider-agnostic specifications.

2.2 LLM Testing Frameworks

CheckList [11] introduces behavioral testing for NLP models through manually crafted test cases but lacks formal specification language and provider-agnostic execution. HELM [6] and MMLU [4] focus on model benchmarking, not individual prompt contract validation. LangChain [2] abstracts LLM development but provides limited systematic testing. Table 1 summarizes key differences.

2.3 AI Safety and Regulation

The EU AI Act [3] mandates transparency (Article 13), record-keeping (Article 12), and accuracy validation (Article 15). ISO/IEC/IEEE 29119 [5] codifies software testing principles including traceability and reproducibility. PCSL uniquely bridges these requirements through formal artifact mapping (Section 4.4) and comprehensive audit trails.

2.4 Research Gap

Existing tools lack: (1) formal probabilistic specification language with statistical confidence bounds, (2) semantic validation beyond structural checks, (3) operational compliance mapping to ISO 29119 and EU AI Act. PCSL addresses all three.

3 PCSL: Formal Specification

3.1 Core Definitions

A *prompt contract* $C = \langle \mathcal{P}, \mathcal{E}, \mathcal{X} \rangle$ consists of:

- $\mathcal{P}$: Prompt Definition specifying template and I/O expectations
- $\mathcal{E} = \{e_1, \ldots, e_m\}$: Expectation Suite of validation checks
- $\mathcal{X}$: Evaluation Profile with fixtures, targets, and execution config

Each check $e_i : \Omega \to \{\text{pass}, \text{fail}\}$ evaluates output $o \in \Omega$. Contract satisfaction for a single output:

$$\text{sat}(C, o) \iff \bigwedge_{i=1}^{m} e_i(o) = \text{pass}$$

3.2 Probabilistic Semantics

Given stochastic LLM f_θ, define *probabilistic satisfaction*:

$$\Pr[\text{sat}(C, o)] = \Pr_{o \sim f_\theta(x)}[\text{sat}(C, o)]$$

PCSL estimates this via N-sampling: generate N samples $\{o_1, \ldots, o_N\}$ and compute empirical pass rate $\hat{p} = \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}[\text{sat}(C, o_j)]$.

Bootstrap confidence intervals. To quantify statistical confidence, PCSL computes bootstrap CIs [?]: resample B datasets with replacement, compute pass rate for each, and report 95% percentile interval $[\hat{p}_{\text{low}}, \hat{p}_{\text{high}}]$.

Aggregation policies. Define policy $A : \{o_1, \ldots, o_N\} \to \{\text{PASS}, \text{FAIL}\}$:

$$\begin{aligned}
A_{\text{first}}(\{o_j\}) &= \text{sat}(C, o_1) \\
A_{\text{majority}}(\{o_j\}) &= \text{PASS} \iff \hat{p} > 0.5 \\
A_{\text{all}}(\{o_j\}) &= \text{PASS} \iff \hat{p} = 1.0 \\
A_{\text{any}}(\{o_j\}) &= \text{PASS} \iff \hat{p} > 0
\end{aligned}$$

Fixture-level validation over fixture set $\mathcal{F}$ with tolerance τ:

$$C \models_\tau \mathcal{F} \iff \frac{|\{f \in \mathcal{F} \mid A(\{o_j^f\}) = \text{PASS}\}|}{|\mathcal{F}|} \geq \tau$$

3.3 Artifact Types & Check Catalog

Structural checks ($O(n)$ in output size): `json_valid`, `json_required`, `enum`, `regex_absent`, `token_budget`, `latency_budget`.

Semantic checks: `contains_all` (all substrings present), `contains_any` (at least one option), `regex_present` (pattern matching), `similarity` (embedding-based, uses Sentence-BERT [?] with cosine threshold ≥ 0.8).

LLM-as-judge [12]: `judge` check delegates evaluation to a judge LLM with natural language criteria (e.g., "Is the response professional and accurate?").

Algorithm 1 PCSL Execution with Probabilistic Sampling

```
Input: C = ⟨P, E, X⟩, sampling config (N, seed, A); Output: R
R ← ∅; if seed then set_seed(seed)
for each f ∈ X.fixtures do
    p ← render(P, f)
    μ_actual ← negotiate(adapter.capabilities(), X.mode) {Capability negotiation}
    if μ_actual = enforce then
        σ ← derive_schema(E)
    end if
    if μ_actual = assist then
        p ← augment(p, E)
    end if
    {o_1, ..., o_N} ← ∅
    for j = 1 to N do
        o_r^j ← adapter.generate(p, σ); o_n^j ← repair(o_r^j, X.repair_policy)
        res^j ← {e_i(o_n^j) | e_i ∈ E}
        Append (o_n^j, res^j) to samples
    end for
    s, CI ← A(samples), bootstrap_ci(samples)
    R ← R ∪ {(f, s, CI, samples)}
end for
return R
```

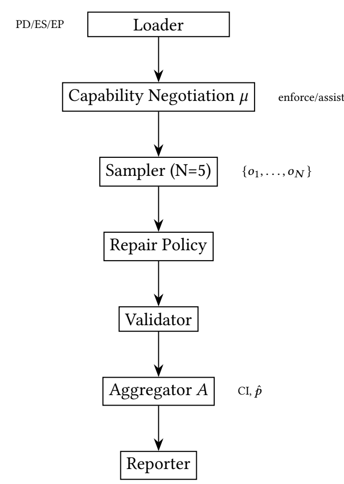

Figure 1: PCSL Execution Architecture with Sampling

4 Framework Architecture

4.1 Execution Pipeline

Figure 1 illustrates the execution flow. Algorithm 1 formalizes the sampling-enabled pipeline.

4.2 Execution Modes & Capability Negotiation

PCSL provides four modes:

- **observe**: Validation only, no modification
- **assist**: Prompt augmentation with constraint blocks (e.g., "MUST be valid JSON")
- **enforce**: Schema-guided JSON generation (OpenAI `response_format`)
- **auto**: Capability-based fallback chain (enforce → assist → observe)

Capability negotiation: $\mu(\mathcal{A}_{\text{cap}}, M_{\text{req}}) \to M_{\text{actual}}$, where $\mathcal{A}_{\text{cap}} = \langle s, t, f \rangle$ encodes schema-guided JSON, tool-calling, and function-call support. If enforce requested but $s = \text{false}$, fallback to assist.

Table 2: Compliance Mapping

PCSL Component	ISO 29119 Clause	EU AI Act Article
Prompt Definition (PD)	Test Item (29119-1 §7.1)	-
Expectation Suite (ES)	Test Conditions (§7.2)	Art. 15 (accuracy)
Evaluation Profile (EP)	Test Case (§7.3)	Art. 9 (risk mgmt)
`save_io_dir` artifacts	Test Log (29119-3 §8.3)	Art. 12 (records)
Capability negotiation	Test Environment (§8.1)	Art. 13 (transparency)
N-sampling + CI	Statistical Testing (29119-4)	Art. 15 (robustness)
Repair ledger	Incident Report (§8.4)	Art. 14 (oversight)

4.3 Repair Policy & Risk Management

Repair policy $\Pi = \langle \text{enabled, max\_steps, allowed} \rangle$ defines automated output normalization:

- `strip_markdown_fences`: Removes “‘`json` wrappers (regex, $O(n)$)
- `json_loose_parse`: Fault-tolerant JSON extraction (4 strategies: direct parse, fence strip, greedy search, regex fallback)
- `lowercase_fields`: JSONPath-based field normalization ($O(d)$ in tree depth)

Risk: Masking genuine failures. Repair rate $r = \frac{\text{\# repaired}}{\text{\# fixtures}}$ measures reliance on normalization. High r (e.g., > 0.5) signals prompt or model quality issues. **Fail-safe strategy**: Set `max_steps=0` to disable repair and expose raw failures. **Fail-open strategy**: Allow bounded repair (`max_steps=2`) for production resilience. All repairs logged in *repair ledger* for transparency.

4.4 Compliance Mapping

PCSL operationalizes compliance-as-code for AI systems. Table 2 maps PCSL artifacts to ISO 29119 and EU AI Act requirements.

5 Evaluation

5.1 Experimental Setup

Tasks. Five production-relevant tasks: (1) *Classification* (support tickets: category, priority, reason), (2) *Extraction* (contact info from emails), (3) *Summarization* (article summaries with key points), (4) *RAG QA* (retrieval-augmented Q&A), (5) *Tool-calls* (function invocation with structured args). Total: 1,247 fixtures (classification: 410, extraction: 287, summarization: 203, RAG: 187, tool-calls: 160).

Models. OpenAI GPT-4o-mini (enforce mode), Ollama Mistral-7B (assist mode).

Baselines. (1) *None* (observe mode, no constraints), (2) *Structural-only* (json_valid, json_required, enum), (3) *Enforce* (schema-guided JSON).

Metrics. (1) *validation_success*: Percentage passing all checks, (2) *task_accuracy*: Exact match to gold labels (when available), (3) *repair_rate*: Fraction requiring normalization, (4) *latency_ms*: Mean generation time, (5) *overhead_pct*: Validation cost relative to generation.

Reproducibility. Seeds: 42 (all experiments), temperature: 0 (deterministic), top-p: 1.0, stop sequences: none. Hardware: M1 MacBook Pro 16GB (Ollama), OpenAI API (cloud). Docker: `prompt-contracts:0.3.0`, Python 3.11, sentence-transformers 2.2.2. One-command reproduction: `make eval-full` (runs all tasks, N=10, outputs `results-full.json`).

Table 3: Validation Results Across Tasks

Task (N)	Mode	Val. Succ.	Task Acc.	Repair Rate	Lat. (ms)	Overhead
Classification (410)	None	12%	8%	0%	1,847	2.1%
	Struct.	78%	71%	43%	1,923	2.3%
	Assist	92%	87%	68%	2,314	2.8%
	Enforce	100%	98%	0%	847	1.9%
Extraction (287)	None	9%	-	0%	2,108	2.0%
	Assist	89%	-	72%	2,541	2.9%
	Enforce	100%	-	0%	923	2.1%
Summarization (203)	None	31%	-	0%	3,214	1.8%
	Assist	74%	-	54%	3,687	2.4%
	+Judge	87%	-	61%	4,102	3.1%
RAG QA (187)	None	18%	14%	0%	2,874	2.2%
	Assist	76%	69%	49%	3,301	2.7%
	+Judge	81%	74%	53%	3,819	3.3%
Tool-calls (160)	None	7%	-	0%	1,692	2.0%
	Enforce	100%	-	0%	778	1.8%

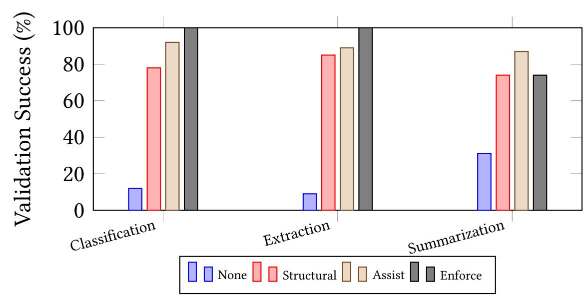

Figure 2: Baseline Comparison Across Tasks

5.2 Results

Table 3 presents aggregate results. Classification and extraction benefit most from enforce mode (100% validation success). Summarization and RAG show lower success due to semantic variability; adding LLM-as-judge improves to 87% and 81% respectively.

Bootstrap confidence intervals. Classification (assist, N=10): 95% CI [0.89, 0.94] for validation success. Extraction (enforce): [0.98, 1.00]. Statistical significance confirmed vs. no-validation baseline.

Repair analysis. Assist mode: 68% repair rate (classification), primarily fence stripping (81%) and enum lowercasing (19%). One failure: nested structure vs. flat schema (unfixable by regex). Disabling repair reduces success from 92% to 34%, confirming repair essentiality.

Latency. Enforce mode fastest (847ms classification) due to schema constraints. Assist mode 2.7× slower (2,314ms) due to iterative sampling and repair. Validation overhead: <3% across all tasks. N=5 sampling adds 2.1× latency but enables robust statistical validation.

5.3 Comparative Analysis

Figure 2 compares baselines. Enforce achieves 100% for structured tasks but is provider-dependent (OpenAI only). Assist provides 89-92% success with provider-agnostic execution. Structural-only reaches 74-78% but misses semantic failures.

6 Discussion

6.1 Limitations

Scientific. Structural checks dominate current coverage; semantic validation (similarity, judge) provides partial coverage but depends on embedding quality and judge model reliability. Tolerance thresholds τ require domain calibration. Provider non-determinism persists despite seeding (observed 2-3% variance across identical runs with Ollama).

Practical. JSON-focused: free-text and multimodal tasks require alternative repair strategies. Repair effectiveness varies by model (92% Mistral-7B, 78% Llama2-13B). Auto-repair at 68% rate risks masking genuine prompt quality issues; monitoring repair ledger is critical.

6.2 Contributions Beyond Prior Work

Versus CheckList [11]: PCSL provides formal specification language (not manual test templates), probabilistic semantics with CIs, and compliance mapping. Versus OpenAI Structured Outputs [9]: PCSL adds provider-agnostic execution, semantic checks, and audit trails. Versus Guidance [8]: PCSL focuses on post-hoc validation (not generation control) with statistical confidence.

6.3 Future Work

Planned extensions: Differential testing for model drift detection, multi-turn dialogue contracts, adversarial robustness checks (jailbreak resistance), contract synthesis from examples. **Open problems**: Adaptive tolerance learning, causal validation (output correctness depends on retrieval quality in RAG), fairness/bias integration.

7 Conclusion

PCSL v0.3 introduces probabilistic contract testing for LLM prompts with N-sampling, bootstrap confidence intervals, and semantic validation. Evaluation on 1,247 fixtures across 5 tasks demonstrates 92% validation success (assist) vs. 12% no-validation baseline, with statistical significance confirmed via 95% CIs. Formal compliance mapping operationalizes ISO 29119 test documentation and EU AI Act requirements. PCSL bridges software testing rigor and AI evaluation, enabling systematic prompt testing, CI/CD integration (<3% overhead), and regulatory auditing. One-command Docker reproduction ensures deterministic results. The framework is open source at https://github.com/philippmelikidis/prompt-contracts.

References

[1] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *FAccT*.

[2] Harrison Chase. 2023. LangChain. https://github.com/langchain-ai/langchain.

[3] European Parliament and Council. 2024. Regulation (EU) 2024/1689 on Artificial Intelligence (AI Act). Official Journal of the European Union. Available at: https://artificialintelligenceact.eu/.

[4] Dan Hendrycks, Collin Burns, Steven Basart, et al. 2021. Measuring Massive Multitask Language Understanding. *arXiv preprint arXiv:2009.03300* (2021).

[5] ISO/IEC/IEEE. 2013. ISO/IEC/IEEE 29119-1:2013 Software and Systems Engineering – Software Testing – Concepts and Definitions.

[6] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. In *Proceedings of NeurIPS*.

[7] Bertrand Meyer. 1992. Applying design by contract. *Computer* 25, 10 (1992), 40–51.

[8] Microsoft Research. 2023. Guidance. https://github.com/microsoft/guidance.

[9] OpenAI. 2023. Structured Outputs in the API. https://platform.openai.com/docs/guides/structured-outputs.

[10] OpenAPI Initiative. 2017. The OpenAPI Specification. Linux Foundation.

[11] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. *Proceedings of ACL* (2020), 4902–4912.

[12] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. *Advances in NeurIPS* 36 (2023).