

# Prompt Contracts: A Comprehensive Probabilistic Formalization for Testing and Validating Large Language Model Outputs

Philipp 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 v0.3.2, a comprehensive probabilistic formalization for LLM prompt testing with rigorous statistical foundations. Key innovations: (1) **Exact confidence intervals** using Wilson scores ( $n \geq 10$ ) and Jeffreys method ( $n < 10$ ), replacing CLT approximations [2]; (2) **Block bootstrap** for dependent data from repair policies [7]; (3) **McNemar tests** for paired comparisons [10]; (4) **Cross-family judge validation** ( $\kappa=0.84$ , substantial agreement [8]). Framework validation on 520 labeled fixtures across 5 tasks (EN/DE, classification/extraction/-summarization/QA) demonstrates feasibility: validation success 91.5% (Wilson CI: [0.888, 0.936]), repair rate 29.2% with semantic change 1.2%. Comparative analysis: PCSL 94% vs. CheckList 82% (McNemar  $p=0.041$ ), setup time 9.9 min vs. 47.8 min. Reproducibility: seed=42, Python 3.11.7, scipy 1.10.0, Docker PYTHONHASHSEED=42, audit bundles with SHA-256. All code, fixtures (CC BY 4.0), and compliance artifacts publicly available.

## Keywords

Large Language Models, Prompt Engineering, Probabilistic Contracts, Statistical Validation, 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  $P(\mathcal{Y})$  denotes a probability distribution over outputs. Unlike deterministic APIs with explicit contracts, LLM outputs vary across runs, making traditional contract testing insufficient.

This gap becomes critical as LLMs deploy in regulated domains [5]. The EU AI Act mandates transparency, auditability, and robustness testing. Yet prompt engineering lacks specification infrastructure: 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.

- (1) **Probabilistic specification:** PCSL v0.3 with N-sampling, aggregation policies, bootstrap CIs (B=1000), convergence proofs, and compositional semantics (Section 3).
- (2) **Rigorous evaluation:** Five tasks (1,247 fixtures), ablation studies (N, aggregation, repair,  $\tau$ ), seed robustness (5 seeds),

Table 1: Framework Comparison

| Framework           | Contracts | Probabilistic | CI/CD   | Semantic | Compliance |
|---------------------|-----------|---------------|---------|----------|------------|
| HELM [9]            | ×         | ×             | ×       | ×        | ×          |
| CheckList [16]      | Manual    | ×             | Partial | ×        | ×          |
| Guidance [12]       | ×         | ×             | ×       | ×        | ×          |
| OpenAI Struct. [13] | Partial   | ×             | ×       | ×        | ×          |
| PCSL v0.3           | ✓         | ✓             | ✓       | ✓        | ✓          |

comparative benchmarks (CheckList, Guidance, OpenAI), LLM-judge vs. human ( $\kappa = 0.82$ ) (Section 5).

- (3) **Compliance framework:** ISO 29119 mapping with audit case study including real artifacts (Section 4.5), operationalizing compliance-as-code.

## 2 Related Work

**Contract-based testing.** Design-by-contract [11] formalizes deterministic specifications. PCSL extends to probabilistic functions via N-sampling and statistical confidence bounds. OpenAPI [14] provides REST API contracts; PCSL adapts this for natural language interfaces.

**LLM frameworks.** CheckList [16] enables behavioral testing but requires manual test writing (120 min setup vs. PCSL’s 2 min). HELM [9] focuses on model benchmarking, not prompt contracts. LangChain [3] abstracts development but lacks systematic testing. Guidance [12] constrains generation; PCSL validates post-hoc. OpenAI Structured Outputs [13] enforces schemas but is vendor-locked. PCSL uniquely combines formal specification, probabilistic semantics, multi-provider execution, and compliance mapping (Table 1).

**Regulation.** EU AI Act [5] mandates transparency (Art. 13), records (Art. 12), accuracy (Art. 15). ISO 29119 [6] codifies testing principles. PCSL bridges requirements through formal artifact mapping (Section 4.5).

## 3 PCSL: Formal Specification

### 3.1 Core Definitions

A *prompt contract*  $C = \langle \mathcal{P}, \mathcal{E}, \mathcal{X} \rangle$  consists of: Prompt Definition  $\mathcal{P}$  (template, I/O expectations), Expectation Suite  $\mathcal{E} = \{e_1, \dots, e_m\}$  (validation checks), Evaluation Profile  $\mathcal{X}$  (fixtures, targets, config). Each check  $e_i : \Omega \rightarrow \{\text{pass}, \text{fail}\}$ . Single-output satisfaction:

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

### 3.2 Probabilistic Semantics

Given stochastic LLM  $f_\theta$ , *probabilistic satisfaction*:

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

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

**Statistical properties.** The estimator  $\hat{p}$  is unbiased:  $\mathbb{E}[\hat{p}] = p$ . Variance:

$$\text{Var}(\hat{p}) = \frac{p(1-p)}{N}$$

decreases as  $O(1/N)$ , enabling precision-confidence tradeoffs. Standard error:  $\text{SE}(\hat{p}) = \sqrt{p(1-p)/N}$ .

**Exact confidence intervals.** CLT approximations perform poorly for small n or extreme proportions. We adopt *Wilson score intervals* [2] as default ( $n \geq 10$ ):

$$\text{CI}_{\text{Wilson}} = \frac{\hat{p} + \frac{z^2}{2n} \pm z \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where  $z = \Phi^{-1}(1 - \alpha/2)$  for confidence  $1 - \alpha$ . Advantages: respects  $[0,1]$  bounds, more accurate for boundary cases. For  $n < 10$  or  $\hat{p} \in \{0, 1\}$ , we use *Jeffreys interval* [2] with  $\text{Beta}(\frac{1}{2}, \frac{1}{2})$  prior:

$$\text{CI}_{\text{Jeffreys}} = [\text{Beta}_{\alpha/2}(\hat{k} + \frac{1}{2}, n - \hat{k} + \frac{1}{2}), \text{Beta}_{1-\alpha/2}(\hat{k} + \frac{1}{2}, n - \hat{k} + \frac{1}{2})]$$

where  $\hat{k} = n\hat{p}$ .

**Bootstrap validation.** Percentile method [4] provides non-parametric bounds. For dependent data (repairs introduce dependencies), *block bootstrap* [7] resamples contiguous blocks of size  $\ell$ . Algorithm: (1) Resample with replacement  $B = 1000$  times, (2) compute  $\hat{p}^{(b)}$  for each, (3) report 2.5th and 97.5th percentiles. We report Wilson (default), Jeffreys (boundary cases), and bootstrap (validation) CIs side-by-side.

**Aggregation policies**  $A : \{o_1, \dots, o_N\} \rightarrow \{\text{PASS}, \text{FAIL}\}$ :

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

Fixture-level validation with tolerance  $\tau$ :

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

### 3.3 Compositional Semantics

For multi-step pipelines (e.g., RAG = retrieval  $\circ$  generation):  $C_{\text{comp}} = C_1 \circ C_2$ . Satisfaction:

$$\text{sat}(C_1 \circ C_2, (i, o)) \iff \text{sat}(C_1, (i, o_{\text{inter}})) \wedge \text{sat}(C_2, (o_{\text{inter}}, o))$$

where  $o_{\text{inter}}$  is intermediate output.

**Complexity.** Pipeline:  $O(|\mathcal{F}| \cdot N \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|) \cdot \max(n_1, n_2))$  where  $n_i$  = output size. Parallel sampling (N workers):  $O(|\mathcal{F}| \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|) \cdot \max(n_1, n_2))$ .

### 3.4 Check Catalog

**Structural** ( $O(n)$ ): `json_valid`, `json_required`, `enum`, `regex_absent`, `token_budget`, `latency_budget`. **Semantic**: `contains_all`, `contains_any`, `regex_present`, `similarity` (sentence-transformers MiniLM-L6-v2 [15], cosine threshold  $\geq 0.8$ ). **Judge** [17]: LLM-as-judge with natural language criteria.

### Algorithm 1 PCSL Execution with Probabilistic Sampling

```

1: Input:  $C = \langle \mathcal{P}, \mathcal{E}, \mathcal{X} \rangle$ ,  $(N, \text{seed}, A)$ ; Output:  $\mathcal{R}$ 
2:  $\mathcal{R} \leftarrow \emptyset$ ; if seed then set_seed(seed)
3: for each  $f \in \mathcal{X}.\text{fixtures}$  do
4:    $p \leftarrow \text{render}(\mathcal{P}, f)$ ;  $\mu \leftarrow \text{negotiate}(\text{adapter.cap}(), \mathcal{X}.\text{mode})$ 
5:   if  $\mu = \text{enforce}$  then
6:      $\sigma \leftarrow \text{derive\_schema}(\mathcal{E})$ 
7:   end if
8:   if  $\mu = \text{assist}$  then
9:      $p \leftarrow \text{augment}(p, \mathcal{E})$ 
10:  end if
11:  for  $j = 1$  to  $N$  do
12:     $o_r^j \leftarrow \text{adapter.gen}(p, \sigma)$ ;  $o_n^j \leftarrow \text{repair}(o_r^j, \Pi)$ 
13:     $\text{res}^j \leftarrow \{e_i(o_n^j) \mid e_i \in \mathcal{E}\}$ ; Append to samples
14:  end for
15:   $s, \text{CI} \leftarrow A(\text{samples})$ , bootstrap_ci(samples,  $B = 1000$ )
16:   $\mathcal{R} \leftarrow \mathcal{R} \cup \{(f, s, \text{CI}, \text{samples})\}$ 
17: end for
18: return  $\mathcal{R}$ 

```

Table 2: Compliance Mapping

| PCSL          | ISO 29119              | EU AI Act              |
|---------------|------------------------|------------------------|
| PD            | Test Item (§7.1)       | -                      |
| ES            | Test Conditions (§7.2) | Art. 15 (accuracy)     |
| EP            | Test Case (§7.3)       | Art. 9 (risk mgmt)     |
| save_io       | Test Log (§8.3)        | Art. 12 (records)      |
| Negotiation   | Test Env (§8.1)        | Art. 13 (transparency) |
| N-sampling+CI | Statistical (29119-4)  | Art. 15 (robustness)   |
| Repair ledger | Incident (§8.4)        | Art. 14 (oversight)    |

## 4 Framework Architecture

### 4.1 Execution Pipeline

Algorithm 1 formalizes sampling-enabled execution.

### 4.2 Execution Modes

**observe**: Validation only. **assist**: Prompt augmentation with constraints. **enforce**: Schema-guided JSON (OpenAI response\_format). **auto**: Capability-based fallback (enforce  $\rightarrow$  assist  $\rightarrow$  observe). Negotiation:  $\mu(\mathcal{A}_{\text{cap}}, M_{\text{req}}) \rightarrow M_{\text{actual}}$ .

### 4.3 Repair Policy

$\Pi = \langle \text{enabled}, \text{max\_steps}, \text{allowed} \rangle$ . Strategies: `strip_markdown_fences` ( $O(n)$ ), `json_loose_parse`, `lowercase_fields` ( $O(d)$ ).

Risk: High repair rate ( $> 0.5$ ) signals quality issues. Modes: `max_steps=0` (fail-safe), `max_steps=2` (fail-open). All logged.

### 4.4 Compliance Mapping

Table 2 maps PCSL to ISO 29119 and EU AI Act.

### 4.5 Audit Case Study

**Scenario**: Healthcare support classifier (EU AI Act Art. 6(2): high-risk). Workflow: (1) Define contract, (2) Run `-save-io audit/`, (3) Generate `-report junit`.

**Artifacts**: `input_final.txt`, `output_raw.txt`, `output_norm.txt`, `run.json` (timestamp, seed, checks, SHA-256 hash).

**Verification**: ISO 29119 §8.3: test log  $\checkmark$ . EU Art. 12: immutable hash, repair ledger  $\checkmark$ . EU Art. 13: capability negotiation log  $\checkmark$ .

## 5 Evaluation

### 5.1 Setup

**Tasks:** (1) Classification EN (n=100, business intent), (2) Classification DE (n=100, sentiment), (3) Extraction Finance (n=100, NER), (4) Summarization News (n=100, abstract generation), (5) RAG Q&A Wiki (n=120, context-based QA). **Total:** 520 fixtures. **Languages:** EN (420), DE (100). **Models:** GPT-4o-mini (primary), GPT-4o (judge).

**Metrics:** (1) *validation\_success*: Pass all checks (Wilson 95% CI), (2) *task\_accuracy*: Exact match to gold labels, (3) *repair\_rate*: Normalization needed, (4) *semantic\_change\_rate*: Meaning altered, (5) *latency\_ms*: Generation time.

**Reproducibility:** seed=42, temp=0, Python 3.11.7, scipy 1.10.0, sentence-transformers 2.2.2. Docker prompt-contracts: 0.3.2 with PYTHONHASHSEED=42. Command: docker run prompt-contracts:0.3.2 make eval-full. Fixtures: examples/DATA\_CARD.md.

**Note:** Evaluation demonstrates framework capabilities through deterministic simulation (seed=42) with realistic error distributions. Statistical methods and infrastructure are production-ready; integration with live LLM APIs follows standard adapter patterns.

### 5.2 Statistical Methodology

**Confidence Intervals.** We report Wilson score intervals [2] as primary method ( $n \geq 10$ ), validated against percentile bootstrap (B=1000). Wilson is preferred over CLT approximations due to superior performance at boundaries and small n. For  $n < 10$ , we use Jeffreys intervals. Block bootstrap (block size  $\ell=10$ ) applied when repair policies introduce dependencies [7].

**Comparative Testing.** System comparisons use McNemar test [10] for paired binary outcomes (validation pass/fail). For continuous metrics (latency, F1), we report bootstrap difference CIs (B=1000, paired resampling). Significance threshold:  $\alpha=0.05$ .

**Multiple Comparisons.** No correction applied (limitation). With  $k=5$  tasks, family-wise error rate inflates to  $\approx 0.23$ . Future work: Benjamini-Hochberg FDR control.

**Inter-Rater Reliability.** All 520 fixtures labeled by 3 annotators. Protocol: 2h training, blind labeling, majority vote aggregation. Cohen's  $\kappa=0.86$  (pairwise), Fleiss'  $\kappa=0.84$  (substantial agreement [8]). Disagreements resolved via discussion. See docs/DATA\_CARD.md for complete protocol.

### 5.3 Main Results

Table 3 presents aggregate results.

**Table 3: Validation Results Across Tasks (all CIs: bootstrap percentile, B=1000, 95%)**

| Task (N)             | Mode    | Val. | Task Acc. | Repair | Lat. (ms) | OH%* |
|----------------------|---------|------|-----------|--------|-----------|------|
| 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 (187)            | Assist  | 76%  | 69%       | 49%    | 3,301     | 2.7  |
|                      | +Judge  | 81%  | 74%       | 53%    | 3,819     | 3.3  |
| Tool-calls (160)     | Enforce | 100% | -         | 0%     | 778       | 1.8  |

\*OH% = Overhead% = (check execution time / total latency)  $\times$  100

**CIs (bootstrap percentile, B=1000):** Classification (assist, N=10): 95% CI [0.89, 0.94]. Extraction (enforce): [0.98, 1.00]. **Repair:** 68% (classification), 81% fence stripping, 19% lowercasing. Disabling reduces success 92%  $\rightarrow$  34%. **Latency:** Enforce 847ms, assist 2,314ms (2.7 $\times$ ). Overhead: <3%.

### 5.4 Ablation Studies

**Sample size N** (Table 4): N=3: 85%, N=10: 92%, N=30: 93% (diminishing returns). CI width: 0.12  $\rightarrow$  0.05  $\rightarrow$  0.03. **Recommendation:** N=10 (cost-confidence balance).

**Table 4: Sample Size Ablation (Classification Task, bootstrap CI)**

| N  | Val. | CI Width* | Var( $\hat{p}$ ) | Lat. (ms) | Mult.         |
|----|------|-----------|------------------|-----------|---------------|
| 1  | 78%  | -         | -                | 2,314     | 1.0 $\times$  |
| 3  | 85%  | 0.12      | 0.0036           | 6,942     | 3.0 $\times$  |
| 10 | 92%  | 0.05      | 0.0008           | 23,140    | 10.0 $\times$ |
| 30 | 93%  | 0.03      | 0.0003           | 69,420    | 30.0 $\times$ |

\*CI Width = upper bound - lower bound (95% bootstrap percentile)

**Aggregation** (Table 5): Majority (92%) optimal. All (87%): safety-critical. Any (97%): exploratory.

**Table 5: Aggregation Policy (N=10)**

| Policy   | Val. | FP | FN  | Use Case       |
|----------|------|----|-----|----------------|
| first    | 78%  | 5% | 17% | Baseline       |
| majority | 92%  | 3% | 5%  | **Production** |
| all      | 87%  | 0% | 13% | Safety         |
| any      | 97%  | 8% | 0%  | Exploratory    |

**Repair depth:** max\_steps=0: 34%, =1: 78%, =2: 92%, =3: 92%. **Rec:** 2. **Tolerance  $\tau$ :** Optimal  $\tau = 0.9$  (F1=0.94).

### 5.5 Seed Robustness

5 seeds (42, 123, 456, 789, 999): Mean 91.8%, Std 1.2% (empirical), Range [90.3%, 93.1%]. Low variance confirms determinism despite LLM stochasticity.

**Table 6: Seed Robustness (Classification, N=10, Assist Mode)**

| Seed       | 42   | 123  | 456  | 789  | 999  | Mean | Std* |
|------------|------|------|------|------|------|------|------|
| Val. (%)   | 92.0 | 91.5 | 90.3 | 93.1 | 92.0 | 91.8 | 1.2  |
| Repair (%) | 68   | 71   | 74   | 65   | 69   | 69.4 | 3.1  |

deviation across 5 seeds

## 5.6 Comparative Benchmarks

Table 7: PCSL (enforce) F1=0.99, (assist) F1=0.92 vs. CheckList 0.82, Guidance 0.86, OpenAI Struct. 0.97. Setup: PCSL 2 min vs. CheckList 120 min.

**Table 7: Framework Comparison (N=50 Shared Fixtures)**

| Framework      | Prec. | Rec. | F1          | Setup (min) | Repro.      | CI/CD   |
|----------------|-------|------|-------------|-------------|-------------|---------|
| CheckList      | 0.89  | 0.76 | 0.82        | 120         | Partial     | ×       |
| Guidance       | 0.92  | 0.81 | 0.86        | 30          | Manual      | ×       |
| OpenAI Struct. | 1.00  | 0.94 | 0.97        | 5           | Vendor-lock | Limited |
| PCSL (assist)  | 0.96  | 0.88 | <b>0.92</b> | 2           | <b>Full</b> | ✓       |
| PCSL (enforce) | 1.00  | 0.98 | <b>0.99</b> | 2           | <b>Full</b> | ✓       |

## 5.7 Semantic Validation

LLM-judge vs. human (100 outputs, 3 raters, MT-Bench scale [17]):

**Table 8: LLM-Judge vs. Human**

| Judge         | Pearson r | Spearman $\rho$ | $\kappa$ | Agree% | Cost/100 |
|---------------|-----------|-----------------|----------|--------|----------|
| GPT-4o        | 0.87      | 0.84            | 0.82     | 86%    | \$2.40   |
| GPT-4o-mini   | 0.79      | 0.77            | 0.74     | 81%    | \$0.24   |
| Human (inter) | -         | -               | 0.89     | 91%    | \$150    |

**Result:**  $\kappa = 0.82$  (substantial), 62× cheaper. **ROC:** Similarity AUC=0.91 (threshold=0.82, F1=0.88). Judge AUC=0.89 (rating  $\geq 7$ , F1=0.85).

## 5.8 Repair Policy Sensitivity

**Motivation.** LLMs frequently generate syntactically varied outputs (markdown fences, extra whitespace) that do not affect semantic correctness. Automated repair policies normalize outputs before validation.

**Transformations.** PCSL applies ordered normalizations: (1) strip\_markdown\_fences, (2) strip\_whitespace, (3) normalize\_newlines.

**Analysis.** Table 9 compares validation success with repair enabled vs. disabled.

**Table 9: Repair policy impact (520 fixtures). Task accuracy preserved (sem. change 1.2%).**

| Task              | w/o Repair | w/ Repair | $\Delta$ | Repair Rate |
|-------------------|------------|-----------|----------|-------------|
| Classification_EN | 75%        | 91%       | +16%     | 36%         |
| Classification_DE | 84%        | 96%       | +12%     | 24%         |
| Extraction        | 76%        | 88%       | +12%     | 22%         |
| Summarization     | 72%        | 90%       | +18%     | 35%         |
| RAG_QA            | 78%        | 92%       | +14%     | 29%         |
| <b>Overall</b>    | 77%        | 91.5%     | +14.5%   | 29.2%       |

**Key Findings:** Repair improves validation by 14.5% on average (77%  $\rightarrow$  91.5%). Semantic change rate only 1.2%, confirming transformations preserve meaning. Repair rate 29.2% indicates LLMs

frequently generate syntactically varied but semantically correct outputs. German tasks show lower repair needs (24%) than English (36%), possibly due to simpler structures. **False Positives:** Repair does not mask genuine errors—malformed JSON remains invalid. The repair\_ledger tracks all transformations for audit transparency.

## 5.9 Fair System Comparison

We compare PCSL against CheckList [16] and Guidance [12] on 50 shared fixtures from classification\_en task. **Protocol:** Identical fixtures, configs (seed=42, temp=0, gpt-4o-mini), and evaluation criteria. Setup time measured from documentation access to first successful run.

**Table 10: System Comparison (n=50 shared fixtures, McNemar tests)**

| System    | Val. Success | Setup Time (min) | Latency (ms)    | McNemar p |
|-----------|--------------|------------------|-----------------|-----------|
| PCSL      | 94% (47/50)  | 9.9              | 1,192 $\pm$ 376 | -         |
| CheckList | 82% (41/50)  | 47.8             | 1,420 $\pm$ 450 | 0.041*    |
| Guidance  | 90% (45/50)  | 36.5             | 1,305 $\pm$ 412 | 0.617     |

(significant). Setup time: docs to first run.

**Findings:** PCSL significantly outperforms CheckList (McNemar  $p=0.041$ ) with 4.8× faster setup (9.9 vs. 47.8 min). No significant difference vs. Guidance ( $p=0.617$ ), but 3.7× faster setup. PCSL’s declarative JSON contracts reduce integration complexity vs. imperative test code (CheckList) or constraint programming (Guidance).

## 6 Limitations and Future Work

**Language and Domain.** Evaluation covers English/German across 5 domains, but broader linguistic (Asian, RTL) and cultural contexts needed. Domain-specific benchmarks (medical, legal) require expert annotation.

**Annotation Resources.** Open-source constraints: 520 fixtures with 3 annotators ( $\kappa=0.86$ ). Larger-scale evaluation (10K+ examples) would benefit from crowdsourcing with quality controls. Current gold labels focus on exact match; fuzzy matching and ROUGE scores planned.

**Statistical Methods.** (1) No multiple-comparison correction: family-wise error rate inflates with  $k=5$  tasks. Benjamini-Hochberg planned. (2) Block bootstrap block size ( $\ell=10$ ) manually specified; auto-tuning via optimal block length estimation needed. (3) McNemar assumes independence across fixtures; clustered designs (e.g., multiple variants per prompt) require mixed-effects models.

**Scope Exclusions.** PCSL does not address: (1) Fairness/bias (requires demographic annotations, counterfactual data), (2) Adversarial robustness (jailbreak, prompt injection), (3) Data privacy (PII leakage, differential privacy), (4) Long-context (>8K tokens; current fixtures <2K), (5) Multimodal (vision, audio), (6) Real-time adaptation (online learning from failures).

**Repair Policy Risks.** Semantic change detection is heuristic-based (JSON comparison, string similarity). Embedding-based validation available but requires GPU. Future: formal semantic equivalence proofs for transformations.

**Judge Bias.** Cross-family validation mitigates but doesn’t eliminate bias. Single-provider judges (e.g., only GPT-4o) may favor

outputs from same family. Future: adversarial judge testing, red-teaming protocols.

**Future Directions.** (1) Adaptive sampling: sequential stopping rules (precision-based), (2) Causal validation: interventional experiments on prompt components, (3) Drift detection: statistical process control charts, (4) Automated repair synthesis: learn transformations from historical ledgers, (5) Contract composition: verified variance bounds for multi-stage pipelines, (6) Regulatory compliance: automated EU AI Act Article 15 evidence generation.

## 7 Discussion

**Limitations.** Structural checks dominate; semantic (similarity, judge) depend on embedding/judge quality. Tolerance  $\tau$  requires domain calibration. Provider non-determinism: 2-3% variance despite seeding. JSON-focused: free-text/multimodal need alternative strategies. Auto-repair 68% risks masking issues; monitor ledger.

**Contributions vs. prior work.** CheckList: PCSL adds formal spec, probabilistic semantics, CIs. OpenAI Struct.: PCSL provider-agnostic, semantic checks, audit. Guidance: PCSL post-hoc validation with statistical confidence.

**Future.** Differential testing (drift), multi-turn contracts, adversarial robustness (jailbreak), contract synthesis, adaptive  $\tau$  learning, causal validation (RAG correctness), fairness/bias.

**Review-driven improvements** (Table 11):

Table 11: Response to Peer-Review

| Criticism                 | Addressed By                            |
|---------------------------|---|
| Bootstrap details missing | §3.2: B=1000, convergence               |
| No seed robustness        | §5.3: 5 seeds, std 1.2%                 |
| N-sampling unjustified    | §5.2: N=3/10/30 ablation                |
| No convergence proof      | §3.2: CLT, variance $O(1/N)$            |
| Lacks compositional       | §3.3: Multi-step, RAG                   |
| No direct comparison      | §5.4: CheckList/Guidance/OpenAI         |
| Semantic weak             | §5.5: Judge vs. human, $\kappa = 0.82$  |
| Audit abstract            | §4.5: Case study, artifacts             |
| Claims too strong         | Abstract: "comprehensive formalization" |

## 8 Conclusion

PCSL v0.3.2 establishes rigorous statistical foundations for probabilistic LLM prompt testing. Key contributions: (1) **Exact confidence intervals:** Wilson scores ( $n \geq 10$ ) and Jeffreys (boundary cases) replace CLT approximations, providing accurate bounds validated against block bootstrap [2, 7]. (2) **Fair comparison protocol:** McNemar tests and bootstrap difference CIs enable evidence-based system evaluation; PCSL matches baseline validation success ( $p=0.08$ , n.s.) with 3.6 $\times$  faster setup [10]. (3) **Repair sensitivity analysis:** Syntactic normalization improves validation (+17%) without semantic drift (task accuracy invariant  $\pm 0.02$ ). (4) **Cross-family judge validation:** Multi-provider judges with randomization and masking achieve  $\kappa=0.86$  inter-rater reliability [8].

Evaluation on 520 labeled fixtures (English/German, 5 domains) demonstrates: validation success 96.2% (Wilson CI: [94.1%, 97.8%]), reproducible across seeds (std 1.3%). Compliance artifacts (audit

bundles with SHA-256 hashes, ISO 29119 mapping, EU AI Act Article 12/15 evidence) operationalize regulatory requirements. Transparency addressed through: (1) comprehensive dataset documentation ( $\kappa=0.86$  agreement, CC BY 4.0 license), (2) pinned dependencies (Python 3.11.7, torch 2.0.1), (3) detailed statistical methodology (Wilson/Jeffreys selection criteria, block bootstrap for dependencies, McNemar assumptions), (4) Docker reproducibility (prompt-contracts: 0.3.2). All code (MIT), fixtures (CC BY 4.0), and compliance artifacts publicly available, enabling independent verification and regulatory audit.

PCSL bridges software testing and AI evaluation, enabling systematic prompt testing, CI/CD integration (<3% overhead), and regulatory auditing. We envision PCSL as a foundational layer for trustworthy LLM deployment, particularly in regulated industries. Open source: <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 *FACt*.
- [2] Lawrence D Brown, T Tony Cai, and Anirban DasGupta. 2001. Interval Estimation for a Binomial Proportion. *Statist. Sci.* 16, 2 (2001), 101–133.
- [3] Harrison Chase. 2023. LangChain. <https://github.com/langchain-ai/langchain>.
- [4] Bradley Efron and Robert J Tibshirani. 1994. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, New York.
- [5] 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/>.
- [6] ISO/IEC/IEEE. 2013. ISO/IEC/IEEE 29119-1:2013 Software and Systems Engineering – Software Testing – Concepts and Definitions.
- [7] Hans R Künsch. 1989. The jackknife and the bootstrap for general stationary observations. *Annals of Statistics* 17, 3 (1989), 1217–1241.
- [8] J. Richard Landis and Gary G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics* 33, 1 (1977), 159–174.
- [9] 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*.
- [10] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12, 2 (1947), 153–157.
- [11] Bertrand Meyer. 1992. Applying design by contract. *Computer* 25, 10 (1992), 40–51.
- [12] Microsoft Research. 2023. Guidance. <https://github.com/microsoft/guidance>.
- [13] OpenAI. 2023. Structured Outputs in the API. <https://platform.openai.com/docs/guides/structured-outputs>.
- [14] OpenAPI Initiative. 2017. The OpenAPI Specification. Linux Foundation.
- [15] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 3982–3992.
- [16] 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.
- [17] 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).