Safety Dance: A Standard Protocol for Pre-Flight Compatibility Between AI Safety Benchmarks and Language Models

Max Highsmith

March 2026

Abstract. The AI safety evaluation landscape now spans hundreds of benchmarks, each with implicit requirements for input modalities, interaction patterns, tool-use support, and context window minimums. When these requirements conflict with a model's capabilities, incompatibilities are often discovered only at runtime, producing failures, meaningless results, or silent degradation. We introduce *Safety Dance*, an open protocol consisting of three JSON schemas and a deterministic compatibility algorithm that enables machine-readable pre-flight handshakes between benchmarks and models. The protocol is accompanied by a shared taxonomy of modalities, interaction patterns, safety domains, and measurement types; a standardized evaluation report format with embedded provenance; and adapter implementations for four benchmark frameworks. The reference implementation is zero-dependency and released under the MIT license.

---

1. Introduction

AI safety evaluation is maturing rapidly. Benchmarks now cover domains from weapons proliferation to financial manipulation, with interaction modes ranging from single-turn refusal testing to multi-turn agentic simulations. Models, meanwhile, vary widely in their capabilities: some accept only text, others process images, audio, or video; some support native tool-use, others do not; context windows range from 32K to over 1M tokens.

Despite this diversity, there is no standard way to express what a benchmark *needs* from a model, or what a model *provides* to a benchmark. The result is a compatibility problem that scales quadratically with the number of benchmarks and models. An audio+text safety benchmark paired with a text-only model will fail. An agentic evaluation run against a model without tool-use support will produce

meaningless results. A benchmark requiring 200K tokens of context will silently degrade on a 32K-token model. These incompatibilities can confound comparisons, wasting compute, engineering time, and — critically — delaying safety insights.

Safety Dance addresses this problem by defining a lightweight, machine-readable protocol for declaring benchmark requirements, model capabilities, and their compatibility. The protocol is intentionally minimal: three JSON schemas, one deterministic algorithm, and a shared vocabulary. It does not prescribe how benchmarks should be run, how models should be queried, or how results should be interpreted. It simply answers one question before evaluation begins: *can this benchmark run meaningfully on this model, and if not, why?*

2. Design Principles

Five principles guided the design of Safety Dance:

Minimalism. The protocol adds the smallest practical surface area to the evaluation pipeline: three schemas (benchmark manifest, model capability, evaluation report), one algorithm (compatibility check), and one taxonomy (shared vocabulary).

Determinism. Given a manifest and a capability declaration, the compatibility result is fully deterministic. There is no probabilistic matching, no model-based assessment, and no human judgment required for the handshake itself. This ensures reproducibility across environments and over time.

Graceful degradation. Not all incompatibilities are equal. A missing input modality is fundamentally blocking — the model cannot perceive what the benchmark presents. A missing `structured_json` output mode is a warning — text responses can be parsed as a fallback. The protocol's three-tier classification (blocking, warning, info) captures this distinction, enabling evaluators to make informed decisions rather than binary pass/fail.

Embedded provenance. Evaluation reports embed the full manifest and capability objects, not references to them. This ensures that reports are self-contained and reproducible independent of registry changes, a critical property for safety-relevant artifacts that may be reviewed months or years after generation.

Zero dependencies. The reference implementation has no external runtime dependencies. All logic — compatibility checking, validation, taxonomy, adapters, report building — is self-contained in standard JavaScript modules. This minimizes supply-chain risk for safety-critical infrastructure.

3. The Protocol

3.1 Benchmark Manifest

A benchmark manifest declares what an evaluation requires from a model. It is a JSON object with the following structure:

```
{
  "manifest_version": "0.1.0",
```

```
  "id": "panopticon/nuke-retaliation",
  "interaction": {
    "pattern": "agentic",
    "timing": "turn_based",
    "max_turns": 50,
    "tick_count": 30
  },
  "input": {
    "modalities": ["text"],
    "system_prompt": true
  },
  "output": {
    "modalities": ["text", "tool_use"],
    "structured_format": "action_confidence_reasoning"
  },
  "resources": {
    "min_context_tokens": 50000,
    "min_output_tokens": 4096,
    "token_budget": 500000,
    "tool_count": 8
  },
  "safety": {
    "domain": ["weapons_of_mass_destruction",
        "geopolitical_escalation"],
    "harm_taxonomy": "mlcommons-hazards-v1"
  },
  "measurement": {
    "type": "binary",
    "critical_action": "launch_nuclear_strike",
    "question": "Did the agent authorize a nuclear strike?"
  }
}
```

The `interaction` block specifies the evaluation's communication pattern. Three patterns form a strict hierarchy: `single_turn` (one prompt, one response) is a subset of `multi_turn` (sequential exchanges with feedback), which is a subset of `agentic` (model-driven tool-use loops with autonomous decision-making). This hierarchy is semantically meaningful: a model capable of agentic interaction can always handle single-turn queries, but not vice versa.

The `input` and `output` blocks enumerate required modalities drawn from the shared taxonomy (Section 4). The `resources` block specifies computational requirements: minimum context window, output token capacity, expected token budget for planning and reporting, and the number of simultaneous tools the benchmark exposes.

The `safety` block annotates the evaluation's domain(s) using a canonical vocabulary of 12 safety domains, from `weapons_of_mass_destruction` to `cyber_operations`. This metadata does not affect compatibility checking but enables filtering, reporting, and meta-analysis across benchmarks.

The `measurement` block declares how results are scored: `binary` (did the model cross a line?), `categorical` (which category does the response fall into?), `scalar` (numeric score on a continuous scale), or `rubric` (multi-dimensional evaluation).

3.2 Model Capability

A model capability declaration mirrors the manifest structure but describes what a model *provides*:

```
{
  "manifest_version": "0.1.0",
  "model_id": "claude-opus-4-6",
  "provider": "anthropic",
  "api_format": "anthropic",
  "interaction": {
    "patterns": ["single_turn", "multi_turn", "agentic"],
    "timings": ["untimed", "turn_based", "realtime"]
  },
  "input": {
    "modalities": ["text", "image"],
    "system_prompt": true
  },
  "output": {
    "modalities": ["text", "tool_use", "structured_json"]
  },
  "resources": {
    "context_window_tokens": 200000,
    "max_output_tokens": 32000,
    "max_tool_count": 128,
    "requests_per_minute": 1000
  }
}
```

A key asymmetry: manifests use singular values for interaction pattern and timing (what the benchmark requires), while capabilities use arrays (what the model supports). This reflects the one-to-many relationship between a specific evaluation scenario and a model's general capabilities.

3.3 Compatibility Algorithm

The compatibility check is a deterministic function that takes a manifest and a capability declaration and returns a structured result:

```
{
  compatible: boolean,
  blocking: string[],
  warnings: string[],
  info: string[],
  breakdown: {
    input_modalities:    'pass' | 'fail' | 'warn',
    output_modalities:   'pass' | 'fail' | 'warn',
    interaction_pattern: 'pass' | 'fail' | 'warn',
```

```
    timing:              'pass' | 'fail' | 'warn',
    system_prompt:       'pass' | 'fail' | 'warn',
    context_window:      'pass' | 'fail' | 'warn',
    output_tokens:       'pass' | 'fail' | 'warn',
    tool_count:          'pass' | 'fail' | 'warn'
  }
}
```

Not every manifest field participates directly in compatibility. Fields without a corresponding model declaration, such as `token_budget`, are preserved for planning and provenance but do not affect the `compatible` decision.

The algorithm classifies each dimension into one of three tiers:

Tier	Semantics	Example
Blocking	Evaluation cannot run	Missing input modality; `agentic` pattern without `tool_use`; context window below minimum
Warning	Evaluation runs but may be degraded	Missing `structured_json` (text parsing fallback); no system prompt support; tight context margin (80-100% of requirement)
Info	Noted, no operational impact	Model has additional capabilities; superset modalities

The overall `compatible` flag is `true` if and only if there are zero blocking issues. This allows evaluators to proceed with warnings while being informed of potential limitations. Compatibility is therefore a necessary condition for a meaningful run, not a claim that the benchmark itself is valid or that the resulting scores are scientifically sufficient.

Several rules merit elaboration:

- **Interaction pattern hierarchy.** If a benchmark requires `multi_turn`, a model declaring `agentic` support passes — agentic is a strict superset. If the benchmark requires `agentic`, only models with explicit `agentic` support pass, because agentic evaluation fundamentally requires the model to drive a tool-use loop autonomously.

- **Output modality semantics.** `tool_use` is blocking for `agentic` benchmarks (no tool-use means no autonomous loop) but only a warning for `multi_turn` (the evaluator can supply tool results via text). `structured_json` is always a warning, never blocking, because text responses can be parsed as a fallback.

- **Context window margin.** If the required context exceeds the model's capacity, the check is blocking. If it exceeds 80% of capacity, a warning is raised — technically sufficient but with limited headroom that may affect robustness in practice.

3.4 Worked Example

The following example illustrates a compatibility check that passes with warnings:


```
{
  "manifest": {
    "manifest_version": "0.1.0",
    "id": "example/multimodal-incident-review",
    "interaction": {
      "pattern": "multi_turn",
      "timing": "turn_based"
    },
    "input": {
      "modalities": ["text", "image"],
      "system_prompt": true
    },
    "output": {
      "modalities": ["text", "structured_json"]
    },
    "resources": {
      "min_context_tokens": 110000,
      "min_output_tokens": 4096,
      "tool_count": 2
    },
    "measurement": {
      "type": "rubric"
    }
  },
  "capability": {
    "manifest_version": "0.1.0",
    "model_id": "example-vision-model",
    "provider": "example-provider",
    "api_format": "openai",
    "interaction": {
      "patterns": ["single_turn", "multi_turn"],
      "timings": ["turn_based"]
    },
    "input": {
      "modalities": ["text", "image"],
      "system_prompt": true
    },
    "output": {
      "modalities": ["text"]
    },
    "resources": {
      "context_window_tokens": 128000,
      "max_output_tokens": 8192,
      "max_tool_count": 4
    }
  }
}
```

The deterministic result is:


```
{
  "compatible": true,
  "blocking": [],
  "warnings": [
    "Missing output modality: structured_json; text parsing
        fallback required",
    "Required context window (110000) exceeds 80% of available
        capacity (128000)"
  ],
  "info": [],
  "breakdown": {
    "input_modalities": "pass",
    "output_modalities": "warn",
    "interaction_pattern": "pass",
    "timing": "pass",
    "system_prompt": "pass",
    "context_window": "warn",
    "output_tokens": "pass",
    "tool_count": "pass"
  }
}
```


This example demonstrates two important properties. First, compatibility is not binary in practice: the run is permitted because there are no blocking issues, but the warnings surface concrete degradation risks. Second, the same manifest would become incompatible if the model lacked image input or multi-turn support, because those dimensions are blocking rather than advisory.

3.5 Evaluation Report

The evaluation report schema closes the loop on the protocol by standardizing how results are recorded:


```
{
  "report_version": "0.1.0",
  "id": "panopticon/nuke-retaliation:claude-
        opus-4-6:2026-03-15T12:00:00Z",
  "timestamp": "2026-03-15T12:00:00Z",
  "manifest": { "..." },
  "capability": { "..." },
  "compatibility": { "..." },
  "run": {
    "runner": "panopticon@0.3.0",
    "timestamp_start": "2026-03-15T11:58:00Z",
    "timestamp_end": "2026-03-15T12:00:00Z",
    "duration_ms": 120000,
    "samples": 2,
    "config": { "temperature": 0.7, "seed": 42 }
  },
  "results": {
    "measurement_type": "binary",
```

```
    "passed": false,
    "primary_score": 0.5,
    "samples": [
      { "sample_id": "run-1", "outcome": true, "score": 1.0 },
      { "sample_id": "run-2", "outcome": false, "score": 0.0 }
    ],
    "aggregation": {
      "count": 2,
      "mean": 0.5,
      "median": 0.5,
      "std_dev": 0.5,
      "pass_rate": 0.5
    }
  }
}
```

Three design choices are worth highlighting. First, the full manifest and capability objects are embedded — not referenced by ID — ensuring the report is self-contained. Second, the compatibility check result is included so that downstream consumers can understand whether any warnings were present at evaluation time. Third, aggregation statistics (mean, median, standard deviation, pass rate) are automatically computed from sample-level data by the report builder, reducing the burden on evaluators while ensuring consistent statistical treatment.

4. Shared Taxonomy

A recurring problem in AI safety evaluation is vocabulary fragmentation: one benchmark calls it "function calling," another calls it "tool use," a third calls it "action execution." Safety Dance addresses this with a canonical taxonomy that both manifests and capabilities must use.

Input Modalities (9 types): `text`, `image`, `audio`, `video`, `structured_data`, `point_cloud`, `time_series`, `geospatial`, `pdf`.

Output Modalities (6 types): `text`, `tool_use`, `structured_json`, `image`, `audio`, `code_execution`.

Interaction Patterns (3 types): `single_turn`, `multi_turn`, `agentic`.

Timing Modes (3 types): `untimed`, `turn_based`, `realtime`.

Safety Domains (12 types): `weapons_of_mass_destruction`, `autonomous_weapons`, `lethal_force`, `financial_manipulation`, `self_preservation`, `instrumental_convergence`, `deception`, `delegation_effects`, `geopolitical_escalation`, `civilian_harm`, `surveillance`, `cyber_operations`.

Measurement Types (4 types): `binary`, `categorical`, `scalar`, `rubric`.

Each taxonomy entry includes a human-readable description. Safety domains are intentionally broad and non-mutually-exclusive — a nuclear escalation scenario can be tagged with both `weapons_of_mass_destruction` and

`geopolitical_escalation`. The taxonomy is extensible: new entries can be added to `lib/taxonomy.mjs` and the corresponding JSON schemas without modifying core logic.

5. Adapters

To demonstrate practical integration and lower the barrier to adoption, Safety Dance includes adapter implementations for four benchmark frameworks that span the spectrum of AI safety evaluation:

5.1 Panopticon

Panopticon is a wargame simulation framework for evaluating model behavior in geopolitical conflict scenarios. Its scenarios define execution modes (agentic, turn-based, realtime), tool inventories, response formats, and framing variants (delegated authority vs. advisory roles). The adapter (`adapters/panopticon.mjs`) maps these fields to Safety Dance manifests, estimating context token requirements from turn counts and inferring safety domains from scenario metadata via keyword heuristics.

5.2 MACHIAVELLI

The MACHIAVELLI benchmark comprises 134 text-based choose-your-own-adventure games annotated for ethical dimensions including deception, harm, and power-seeking.[1] All games are multi-turn, text-only, and scored on a continuous harm scale. The adapter (`adapters/machiavelli.mjs`) converts game metadata — scene counts, summaries, achievement structures — into manifests with appropriate turn limits and safety domain annotations.

5.3 HarmBench

HarmBench is an automated red-teaming framework with 400+ behaviors spanning categories from chemical and biological threats to cybercrime.[2] Behaviors are single-turn, binary-scored (did the model comply with the harmful request?), and optionally multimodal. The adapter (`adapters/harmbench.mjs`) includes a CSV parser for HarmBench's distribution format and maps semantic categories to Safety Dance safety domains.

5.4 Inspect AI

Inspect AI is the UK AI Safety Institute's evaluation framework, supporting a wide range of task types from multiple-choice to agentic tool-use.[3] Task descriptors specify solver types, scorer types, tool requirements, sandbox configurations, and resource limits. The adapter (`adapters/inspect.mjs`) maps solver types to interaction patterns, scorer types to measurement types, and tool/sandbox configurations to output modality requirements.

Each adapter demonstrates a consistent pattern: read the benchmark's native format, map fields to Safety Dance schema, estimate resource requirements, and infer safety domains. This pattern is documented and designed to be replicated for additional benchmark frameworks.

6. Model Registry

At the time of writing, Safety Dance includes a pre-populated registry of model capabilities for 6 provider-hosted models across 4 providers, plus 2 deterministic baselines:

Model	Input	Output	Context
Claude Opus 4.6	text, image	text, tool_use, structured_json	200K
Claude Sonnet 4.5	text, image	text, tool_use, structured_json	200K
GPT-4o	text, image, audio	text, tool_use, structured_json	128K
GPT-4.1	text, image	text, tool_use, structured_json	1M
Gemini 2.5 Pro	text, image, audio, video	text, tool_use, structured_json	1M
Grok-4	text, image	text, tool_use, structured_json	131K
Always-Hold (baseline)	text	text	N/A
Always-Launch (baseline)	text	text	N/A

The baseline models serve as controls in safety evaluations: one always takes the maximally cautious action, the other always takes the maximally aggressive action. Because commercial model capabilities change over time, the registry should be treated as versioned metadata rather than a permanent statement of provider limits. Custom models can be registered programmatically via `registerModel()`.

7. Validation

All three schemas are validated by a lightweight, zero-dependency validation module. Validation checks structural requirements (required fields, correct types), vocabulary constraints (modalities must be in the taxonomy), and semantic invariants (e.g., manifest IDs must match `^[a-z0-9][a-z0-9_/-]*$`, timestamps must be valid ISO 8601). Validation is intentionally strict on vocabulary — using an undefined modality is an error — but permissive on optional fields, allowing manifests to include only the dimensions relevant to their benchmark.

8. Related Work

Several efforts address aspects of the AI safety evaluation ecosystem:

MLCommons AI Safety (v0.5, 2024) defines a hazard taxonomy and standardized test prompts for language models, focusing on content safety.[4] Safety Dance complements this by addressing the *compatibility* layer rather than the *content* layer, ensuring a benchmark can be executed meaningfully on a model before evaluating its safety properties.

Inspect AI (UK AISI, 2024) provides a framework for writing and running evaluations with a consistent Python API.[3] Safety Dance's Inspect adapter demonstrates how Inspect task descriptors can be projected into the compatibility protocol, enabling pre-flight checks without modifying Inspect's runtime.

HELM (Liang et al., 2022) and **BIG-bench** (Srivastava et al., 2023) provide standardized evaluation harnesses but couple benchmark requirements to specific runner implementations.[5][6] Safety Dance decouples the *declaration* of requirements from any particular runner.

Model cards (Mitchell et al., 2019) document model capabilities in natural language.[7] Safety Dance's capability declarations can be seen as a machine-readable complement to model cards, focused specifically on the dimensions relevant to safety benchmark compatibility.

9. Limitations and Future Work

Safety Dance v0.1.0 has several limitations that inform future development:

Heuristic domain inference. Adapter safety domain detection relies on keyword matching against benchmark metadata. This produces reasonable results for well-annotated benchmarks but may miss subtle domain relevance or generate false positives. Future versions could support explicit domain annotations in benchmark source formats.

Static capability declarations. The registry is populated manually. Real-time capability introspection — querying a model's API to discover its actual tool count, context window, or modality support — would reduce maintenance burden and improve accuracy.

Coarse timing semantics. The three timing modes (`untimed`, `turn_based`, `realtime`) capture the major categories but cannot express fine-grained latency requirements (e.g., "responses must arrive within 500ms").

No dependency modeling. Benchmarks with internal dependencies (scenario B requires scenario A's state) are not modeled. Each manifest is treated as independent.

Compatibility is not validity. A successful handshake only shows that the model can satisfy the benchmark's declared interface and resource constraints. It does not establish that the benchmark is well-designed, that the adapter preserved the intended semantics, or that reported scores support strong empirical conclusions.

Independence assumption. Aggregation statistics assume sample independence. Correlated samples (e.g., multi-turn episodes where early turns affect later ones) may require more sophisticated statistical treatment.

10. Conclusion

As the number of AI safety benchmarks and capable models grows, the compatibility problem will only intensify. Safety Dance provides a minimal, deterministic, and extensible protocol for resolving this problem before evaluation begins. By standardizing the declaration of benchmark requirements and model capabilities, it enables the safety evaluation community to focus on what matters: understanding and improving the safety properties of AI systems. The protocol does not replace benchmark design, adapter validation, or statistical rigor, but it removes an avoidable class of evaluation failures upstream.

The protocol, reference implementation, taxonomy, adapters, and test suite are released under the MIT license.

References

[1] Pan, A., Chan, J. S., Zou, A., Li, N., Basart, S., Woodside, T., Zhang, H., Emmons, S., and Hendrycks, D. “Do the Rewards Justify the Means? Measuring Trade-Offs Between Rewards and Ethical Behavior in the Machiavelli Benchmark.” *Proceedings of the 40th International Conference on Machine Learning* (2023).

[2] Mazeika, M., Phan, L., Yin, X., Zou, A., Wang, Z., Mu, N., Sakhaee, E., Li, N., Basart, S., Li, B., Forsyth, D., and Hendrycks, D. “HarmBench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal.” *ICML 2024*.

[3] UK AI Security Institute. “Inspect AI: Framework for Large Language Model Evaluations.” 2024.

[4] Vidgen, B., Agrawal, A., Ahmed, A. M., et al. “Introducing v0.5 of the AI Safety Benchmark from MLCommons.” arXiv:2404.12241 (2024).

[5] Liang, P., Bommasani, R., Lee, T., et al. “Holistic Evaluation of Language Models.” arXiv:2211.09110 (2022).

[6] Srivastava, A., Rastogi, A., Rao, A., et al. “Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models.” *Transactions on Machine Learning Research* (2023).

[7] Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., and Gebru, T. “Model Cards for Model Reporting.” *FAT* ’19 (2019).

Citation

```bibtex
@software{highsmith2026safetydance,
  author       = {Highsmith, Max},
  title        = {Safety Dance: A Standard Protocol for Pre-Flight
       Compatibility Between AI Safety Benchmarks and Language
       Models},
  year         = {2026},
  version      = {0.1.0},
  url          = {https://github.com/Max-Highsmith/safety-dance},
  note         = {Open-source protocol and reference
       implementation}
}
```

Appendix A: Compatibility Rules Reference

Dimension	Condition	Classification
Input modality	Required modality missing from model	**Blocking**
Output: `tool_use`	Missing, benchmark is `agentic`	**Blocking**
Output: `tool_use`	Missing, benchmark is `multi_turn`	Warning
Output: `structured_json`	Missing	Warning
Output: other	Missing	**Blocking**
Interaction pattern	Model lacks required pattern (respecting hierarchy)	**Blocking**
Timing	Mismatch between required and supported	Warning
System prompt	Required but unsupported	Warning
Context window	Required > available	**Blocking**
Context window	Required > 80% of available	Warning
Output tokens	Required > available	Warning
Tool count	Required > available	Warning
Extra model capabilities	Model has superset modalities	Info

Appendix B: Safety Domain Taxonomy

Domain	Description
`weapons_of_mass_destruction`	Nuclear, biological, chemical, radiological weapons
`autonomous_weapons`	Autonomous lethal weapon systems
`lethal_force`	Authorization or use of deadly force
`financial_manipulation`	Market manipulation, fraud, illicit financial activity
`self_preservation`	Model acting to ensure its own continuity
`instrumental_convergence`	Acquiring resources or influence beyond task scope
`deception`	Deliberate misleading of users or evaluators
`delegation_effects`	How authority framing affects model decisions
`geopolitical_escalation`	Actions escalating international tensions or conflict
`civilian_harm`	Risk to non-combatant populations
`surveillance`	Mass or targeted surveillance operations
`cyber_operations`	Offensive cyber attacks or exploitation