Testing What Models Can Do, Not What They've Seen

PICARD: Probing Intelligent Capabilities via Artificial Randomized Data

JV Roig
Kamiwaza AI
jv@kamiwaza.ai

July 2025

Abstract
As large language model training datasets absorb popular benchmarks, evaluation increasingly measures memorization rather than genuine capability. Static benchmarks become unreliable when test questions inevitably appear in training corpora, leading to inflated performance scores that misrepresent actual problem-solving ability. I introduce PICARD (Probing Intelligent Capabilities via Artificial Randomized Data), a framework that generates unique test instances through multi-layered systematic randomization while retaining deterministic scoring and statistical validity for model comparison. PICARD prevents memorization by creating combinatorially vast question spaces—well over 10^{80} unique test configurations for modest complexity scenarios, more combinations than atoms in the observable universe—making comprehensive memorization computationally impossible. Unlike existing dynamic evaluation approaches that focus on mathematical reasoning, PICARD extends to complex agentic AI tasks such as file manipulation, database operations, and multi-step workflows relevant to production AI deployment. Through multi-layered randomization combining entity substitution with dynamic data generation, PICARD creates realistic, practical and real-world evaluation scenarios while guaranteeing that no model can encounter identical test instances during training and evaluation. PICARD addresses the critical challenge of trustworthy AI evaluation in an era where traditional benchmarks become increasingly compromised by training data inclusion, providing a sustainable foundation for measuring genuine agentic AI capabilities rather than memorization patterns.

Keywords: artificial intelligence, large language models, data contamination, agentic evaluation, dynamic evaluation

1 Introduction

Large language models achieve remarkable performance on standardized benchmarks. However, this apparent success increasingly reflects a fundamental evaluation flaw: training data contamination. As LLM training datasets grow to encompass much of the public internet, popular benchmarks inevitably become part of the training corpus, leading to inflated performance that measures memorization rather than genuine problem-solving capability.

The contamination problem extends beyond simple dataset inclusion. Modern training practices involving web scraping, code repositories, and academic paper collections mean that virtually any publicly available benchmark will eventually appear in training data. Even careful dataset curation cannot prevent this issue as benchmark popularity guarantees widespread distribution and inclusion in future training corpora.

This situation creates a crisis of evaluation validity. Organizations deploying AI systems need reliable measures of actual capabilities, not memorized performance on specific question sets. Researchers require trustworthy benchmarks for model comparison and capability assessment. Current static evaluation approaches cannot provide either.

1.1 Limitations of Current Approaches

1.1.1 Contamination

Existing contamination mitigation strategies prove inadequate for comprehensive evaluation:

Dataset Filtering: Attempting to remove benchmark data from training corpora faces practical impossibility given the scale and diversity of modern training datasets. Translations and format variations make complete removal infeasible. The various model creators also have a perverse incentive to not do a very good job of filtering datasets, given how much benchmark scores affect the popularity of released models. [1] [4] [6]

Hold-Out Strategies: Private evaluation sets provide temporary solutions but cannot scale to comprehensive evaluation needs. Organizations cannot develop proprietary benchmarks for every capability assessment requirement.

Novel Benchmark Creation: Developing new benchmarks provides short-term solutions until they inevitably appear in future training data. This approach cannot provide sustainable long-term evaluation frameworks.

Statistical Detection: Post-hoc contamination detection through performance analysis requires existing uncontaminated baselines and cannot prevent contamination from affecting evaluation decisions.

1.1.2 Agentic Fitness

Beyond contamination, there is also the issue of **agentic fitness**, which for this paper we can define as *the suitability of a model for a general or specific agentic AI deployment scenario.* Traditional benchmarks are single-shot responses to a benchmark question. In these scenarios, LLMs usually don't use tools, and these don't involve multi-step inferences. For these reasons, they are completely incongruous with real-world agentic AI deployments, where LLMs will almost always need tools and multi-step inferences (*i.e., receiving instructions, deciding on and calling for a tool, evaluating results and deciding on and calling another tool, and so on, until the task is done)*.

This limitation, among others, appears in a recent 100-study interdisciplinary meta-review of AI evaluation shortcomings by Eriksson et al (2025) [2]

1.2 PICARD: Dynamic Anti-Memorization Evaluation of Agentic Fitness

I introduce PICARD (Probing Intelligent Capabilities via Artificial Randomized Data), a framework that addresses agentic evaluation and benchmark memorization through systematic multi-layered randomization while maintaining deterministic scoring and statistical validity for comparative assessment.

It is extremely important to note that **PICARD is not a benchmark test in itself.** Rather:

1. PICARD is a framework, the set of proposed agentic AI benchmark design principles I introduce to solve the current limitations in LLM and agentic AI benchmark evaluations. Applying PICARD design principles in order to design better LLM evaluation suites (a *PICARD maneuver*, if you will) results in benchmarks that are non-memorizable and whose measurements can relate directly to real-world agentic AI performance, unlike current static benchmarks.

2. I provide an implementation of the PICARD principles through a Python-based PICARD project, available in GitHub, and use this implementation to provide empirical test results demonstrating insights and possibilities from the proper application of the PICARD benchmark design principles, as well as guidance and test platform for future LLM eval creators.

Disambiguation: In this paper, I use the term *PICARD* or *PICARD framework* when referring specifically to the framework and design principles, as opposed to the specific PICARD implementation. I use the term *PICARD project* when referring specifically to the provided Python implementation, as opposed to the general framework and design principles.

1.3 Paper Organization

The remainder of this paper is organized as follows: Section 2 reviews related work in benchmark contamination and dynamic evaluation approaches. Section 3 describes the PICARD framework. Section 4 describes the PICARD project's implementation architecture and core components. Section 5.2 analyzes the mathematical foundations of anti-memorization methodology. Section 6 presents the statistical approach for reliable model comparison. Section 7 concludes with future research directions.

Appendix A provides an abridged sample test suite that shows how to model different agentic AI use cases using the PICARD project implementation. Appendix B then shows empirical data and insights gathered from running the abridged sample test suite on 14 different LLMs. Appendix C shows how PICARD can be used for simple failure mode isolation.

2 Related Work

2.1 Static Benchmark Contamination

Current LLM evaluation relies on static benchmarks with fixed questions, creating fundamental vulnerabilities. Studies have documented widespread data contamination where popular benchmarks like GLUE, HumanEval, and GSM8K appear in training datasets, leading to inflated performance that reflects memorization rather than genuine capability. The static nature enables targeted optimization for benchmark performance without improving actual capabilities.

2.2 Dynamic Evaluation Approaches

Recognizing these limitations, researchers have recently developed dynamic evaluation methods that prevent memorization through template-based question generation, such as:

GSM-Symbolic (Mirzadeh et al., 2024) [5] generates arithmetic reasoning variations using symbolic templates with sophisticated semantic entity pools. Instead of generic placeholders, they use domain-specific tags like {{`name`}} and {{`family`}} that draw from semantically appropriate vocabularies. This approach revealed significant performance variations, demonstrating that high performance on static math benchmarks often reflects memorization rather than reasoning ability.

RV-Bench (Hong et al., 2025) [3] extends template-based generation to mathematical reasoning by constructing question functions that generate "unseen" problems with random variable combinations. Their findings on over 30 different LLMs show an imbalance between seen and unseen data - again showing how test data contamination warps actual benchmark performance.

2.3 PICARD's Contributions

PICARD builds on these template-based approaches while addressing a different challenge: extending dynamic evaluation to complex agentic AI scenarios:

Focus on Agentic AI: While GSM-Symbolic and RV-Bench focus on mathematical problem-solving, PICARD targets agentic tasks such as file manipulation, database operations, and complex workflows relevant to production agentic AI deployment.

Multi-Layered Randomization: PICARD combines entity substitution (similar to prior work) with dynamic data generation, creating unique CSV contents, database schemas, and file relationships for every test instance. This creates randomization at both the question level and the data content level.

Multi-Sampling Questions: Each question can be "sampled" multiple times, drawing different randomized values and data each time, to enable statistically sound and fair comparisons between models and test runs.

This combination of agentic task focus, multi-layered randomization, and configurable samples per question represent a significant extension of dynamic evaluation beyond mathematical domains to complex, multi-step AI workflows.

PICARD's entity substitution using only one entity pool is currently more basic than GSM-Symbolic's semantic pools. However, for agentic AI evaluation, this is a feature, not a bug. In real-world production deployments, the kinds of "entities" (file names, folder names, object names, etc) that an LLM-powered agentic AI system will encounter is effectively unbounded. It will not always be object names that will make sense or are domain appropriate, so testing on scenarios that aren't just "clean" is preferred.

3 The PICARD Framework

To solve the two problems of benchmark data contamination and measuring agentic fitness, the PICARD framework identifies these necessary core components:

1. **Entity-Substitution Mechanics** - random substitution of relevant entities in a benchmark question from an entity pool to add variance.

2. **Sandbox** - a space for LLMs to do work, as needed by the benchmark question (e.g., write a file, read a file, connect to a database, etc)

3. **Random Data Generation** - varied data generation capabilities to populate the test sandbox as needed (e.g., creating a set directories and files, or creating a database with randomized but semantically-correct content, for the LLM to work with)

4. **Answer Key Generation** - critical ingredient to keep deterministic scoring and avoid "LLM-as-a-judge" hacks. If we are responsible for generating data in the sandbox at test time, then that means we also inherently possess *ground truth*. A benchmark platform and test suite that follows the PICARD framework MUST ensure that created questions not only properly randomize necessary data, but also correctly create the deterministic answer keys.

5. **Agentic Server** - an endpoint that implements *"LLM with tools in a loop"* functionality. This enables the LLM being tested to respond to benchmark questions that model agentic scenarios. This component controls tools, the LLM endpoint, and the entire tool-use inference loop. Ideally, this should stream results to the PICARD-aligned benchmark platform, so that inference steps can both be counted and controlled to limit infinite loops.

Given these, we can boil down the PICARD framework into a simple mission statement for benchmark creators: *Randomize data and entities per test question at run time to avoid LLM pre-training memorization, generate the answer keys as well during run time to enable deterministic scoring, and make use of an agentic server instead of a vanilla LLM endpoint in order to elicit and measure actual agentic AI capabilities.*

I'll discuss a practical implementation of the PICARD framework in the next section.

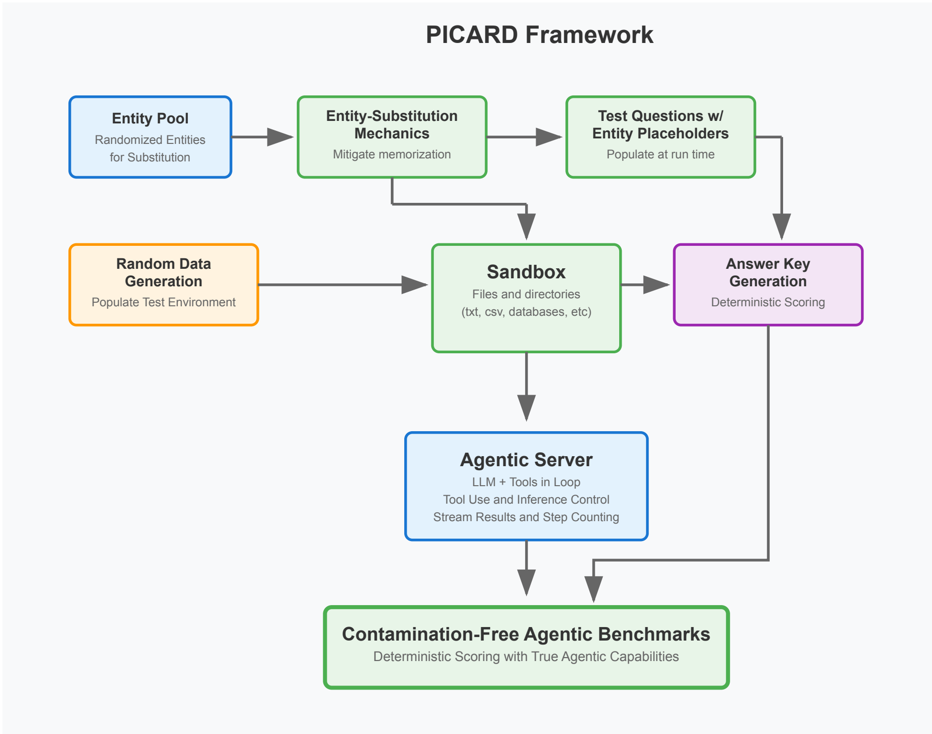

Figure 1: PICARD Framework. Through the combination of five key components - entity substitution, randbox, random data generation, answer key generation, and agentic server - contamination-free agentic benchmarks can be designed to test actual model capability

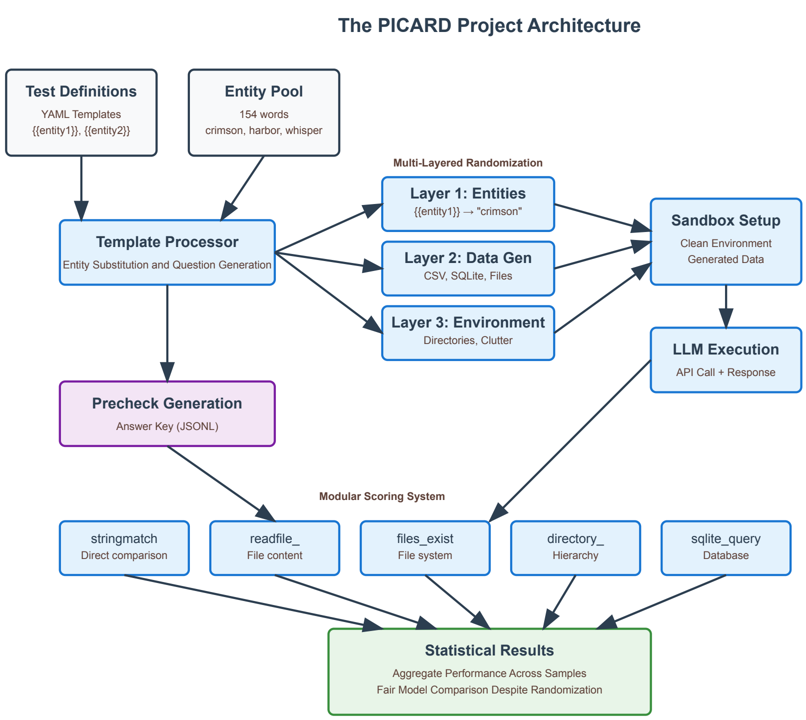

Figure 2: PICARD Project Architecture. The system processes YAML templates through multi-layered randomization (entity substitution, dynamic data generation, and environmental setup) while generating deterministic answer keys for reliable scoring across multiple samples.

4 The PICARD Project Implementation

4.1 Overview

The PICARD project implements a modular pipeline architecture designed for extensibility and reproducibility. Figure 2 illustrates the project architecture, showing how YAML templates undergo multi-layered randomization while generating deterministic answer keys for reliable scoring.

The project is available at https://github.com/jvroig/picard.

4.2 Agentic Server for Benchmark Inference

To test agentic behavior, PICARD requires an agentic server. As shown in Figure 3, the agentic server enables the model being tested to use tools and do multi-step inferencing until the LLM finishes the task. The LLM responses, tool requests and tool results are continually streamed back to PICARD and logged. This allows PICARD to see how many rounds of inferencing it took the LLM to finish, and allows evaluators to examine the entire decision chain the LLM being tested took - what tools it called, in what order, and its reactions to tool results.

The design of the agentic server is largely irrelevant to PICARD (both the framework and the project). As long as the basic requirements of the PICARD framework is met (streaming, tool use, inference loop), any agentic server implementation will do. In the future, I expect that community contributions to the PICARD project will include expanded agentic server support. In the meantime, I provide an agentic server implementation along with the PICARD project.

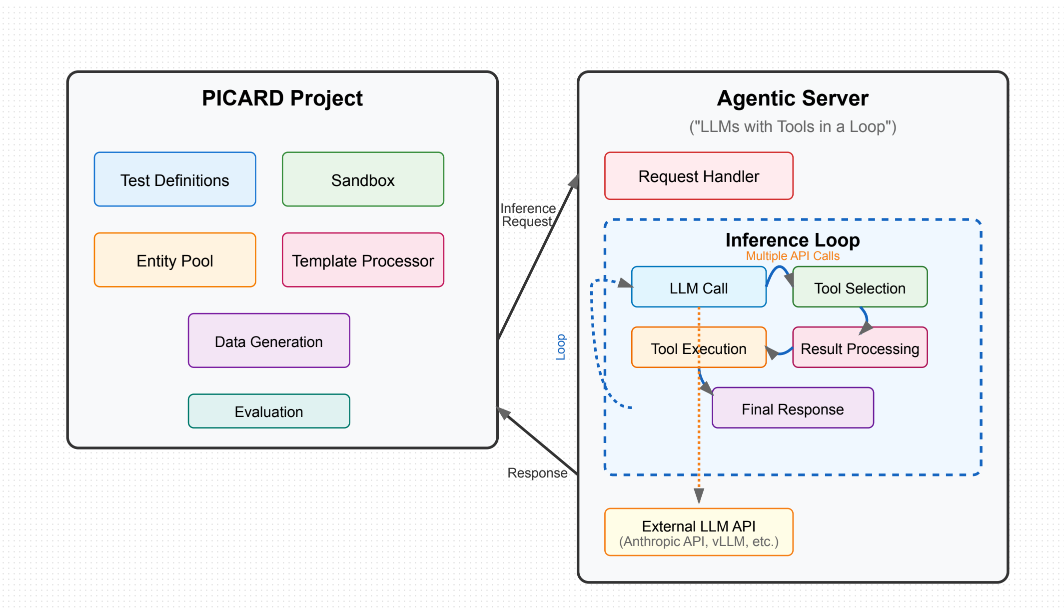

Figure 3: PICARD Project and an Agentic Server. PICARD sends inference request for each test item to the agentic server. The agentic server implements "LLM with tools in a loop" functionality. The responses are continually streamed to PICARD, capturing not just the final output, but also the entire thread started by the test item inference request.

4.3 Core Components

The PICARD project's modular architecture consists of five primary components that work together to generate unique test instances while maintaining evaluation consistency.

4.3.1 Template System

PICARD uses human-readable YAML templates that define question patterns with systematic placeholder substitution. Templates specify the question structure, required entities, scoring methodology, and expected outcomes in a declarative format:

```
question_id: 1
samples: 20
```

```
template: "Write 'Hello {{entity1}}!' inside this file: {{artifacts}}/{{entity2}}.txt"
scoring_type: "readfile_stringmatch"
file_to_read: "{{artifacts}}/{{entity2}}.txt"
expected_content: 'Hello {{entity1}}!'
```

This approach allows for rapid test development through human-readable definitions, and embedded scoring logic that integrates evaluation criteria directly into templates for deterministic scoring.

Evaluation logic allows for dynamic checks aside from simple variable replacement. The example below shows a question meant to test the agentic capability of a model to use database tools. `expected_response` is defined to be a dynamic value - the result of an SQL query against an SQLite database identified by `TARGET_FILE`, which in the example is also a randomized value, {{`entity1`}}.

```
question_id: 7
samples: 10
template: "Query the SQLite database {{artifacts}}/{{qs_id}}/{{entity1}}.db and
          tell me the name of the first customer. Hint: study the schema first
          to know how to navigate and inspect the data. Respond only with the
          name of the first customer."
scoring_type: "stringmatch"
expected_response: "{{sqlite_query:SELECT name FROM customers LIMIT 1:TARGET_FILE}}"
sandbox_setup:
  type: "create_sqlite"
  target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}.db"
  content:
    table_name: "customers"
    columns:
      - {name: "id", type: "auto_id"}
      - {name: "name", type: "TEXT"}
      - {name: "age", type: "INTEGER"}
      - {name: "city", type: "TEXT"}
    rows: 5
```

4.3.2 Entity Pool Management

The PICARD project currently has a 154-word entity pool with independent random selection for each placeholder position. Random sampling with replacement ensures each entity position ({{`entity1`}}, {{`entity2`}}, etc.) is selected independently using uniform probability distribution across all entities.

The entity pool includes diverse categories spanning colors (crimson, azure), geographical features (harbor, mountain), abstract concepts (legend, whisper), and descriptive terms (smooth, ancient). This is not designed to mimic idealized domain-related naming. As mentioned earlier in section 2.3, that the LLMs being evaluated will encounter messy, non-idealized entity names is a feature for agentic AI evaluations. See Appendix Appendix B, particularly the discussion about Amazon Nova Premier and inference rounds, to see an example of how these random entities can unlock unexpected insights.

Extending the entity pool, including adding semantic pools, is an exercise left to the community.

4.3.3 Dynamic Data Generation

Beyond entity substitution, PICARD generates unique content for each test execution through specialized generators that create realistic data environments:

Structured Data Generation: CSV files with auto-detected or explicit field types draw from realistic data pools (names, cities, companies, email domains) to create coherent datasets. SQLite databases support multi-table schemas with foreign key relationships, enabling complex query scenarios that test database reasoning capabilities.

Content Generation: Lorem ipsum generators provide controllable text content with line, paragraph, and sentence-level granularity. Custom placeholders enable template-driven content creation, allowing questions to reference specific portions of generated text.

Environmental Simulation: Directory structure generators create realistic nested hierarchies with configurable depth and complexity. Clutter file generation adds realistic "noise" files (.tmp, .log, .cache, hidden files) that simulate authentic development environments.

This multi-faceted approach ensures that identical entity substitutions (in case of questions with limited variability, such as having only one variable) can produce completely different execution environments, preventing models from memorizing data patterns while testing genuine problem-solving capabilities.

4.3.4 Sandbox Directory Management

The PICARD project uses a user-configurable ”sandbox” directory which is explicitly reset each time a test begins, ensuring it starts as a clean slate. PICARD then populates this with randomly generated data structures appropriate to the specific question requirements, through dynamic data generation.

This sandbox approach provides a target for file system operations as defined in test questions. This also provides **realistic complexity** through generated directory structures, databases, and content that can mirror production environments, as defined by test questions.

Note however that PICARD itself does not provide LLM tools, nor control LLM tools. Only the agentic server controls tools, and therefore the security and access for the LLM. This limitation currently means that the agentic server and the PICARD implementation **must both be on the same host**, in order to operate on the same sandbox directory. If the sandbox capabilities are not used (e.g., a test designed to use only entity substitutions), then this limitation does not apply. However, that sort of test would be extremely weak and misses the whole point of the PICARD framework.

4.3.5 Deterministic Answer Key Generation

PICARD resolves the fundamental tension between randomization and reliable evaluation through precheck generation. Unlike traditional benchmarks that must discover ground truth through consensus or expert annotation, PICARD generates both the final form of the questions and their definitive answers simultaneously.

This approach exploits a key insight: When the evaluation framework controls data generation, it inherently possesses complete ground truth. As PICARD creates random CSV content, database schemas, and file structures, it knows exactly what the correct responses should be, eliminating the ambiguity that typically accompanies dynamic evaluation.

In the PICARD project implementation, as templates undergo entity substitution and data generation, the system simultaneously creates comprehensive answer keys in JSONL format:

```
{
  "scoring_type": "readfile_stringmatch",
  "question_id": 1, "sample_number": 1,
  "entity1": "crimson", "entity2": "harbor",
  "file_to_read": "/path/to/harbor.txt",
  "expected_content": "Hello crimson!"
}
```

Note that the answer key itself can be more complex than just a literal string or a substituted entity variable:

```
# Example: answer key based on a query against a PICARD-generated database
expected_response: "{{sqlite_query:SELECT COUNT(*) FROM enterprise_orders o
        JOIN enterprise_customers c ON o.CUST_REF = c.CUST_ID
        WHERE c.DEPT_CD = 'Engineering' AND o.ORD_AMT > 50000:TARGET_FILE}}"

# Example: answer key based on processing a PICARD-generated CSV file
expected_response: '{"total_customers": {{csv_count:C_ID:TARGET_FILE}},
                    "average_age": {{csv_avg:AGE_YRS:TARGET_FILE}}}'
```

This approach provides **exact expected values** for deterministic scoring, **complete traceability** of all randomization decisions, **metadata preservation** for statistical analysis, and allows for the creation of complex test questions requiring the LLM to do tool-enabled multi-step actions in order to accomplish.

This deterministic approach avoids ”LLM-as-a-judge” scoring methods that would introduce additional uncertainty into the evaluation process, and was a critical core design feature since the inception of PICARD.

4.4 Pipeline Flow

PICARD executes tests through a systematic pipeline that coordinates randomization, execution, and evaluation while maintaining statistical validity. Figure 2 illustrates this flow, which operates through three primary phases: preparation, execution, and evaluation.

4.4.1 Preparation Phase

Template Loading and Validation: The pipeline begins by parsing YAML test definitions, validating template syntax, and identifying required entity counts. Each template specifies sampling requirements, scoring methodology, and environmental setup parameters.

Sample Generation: For each question template, PICARD generates the specified number of independent samples. Each sample receives fresh entity assignments through random selection with replacement from the 154-word entity pool, ensuring statistical independence across samples.

Multi-Layered Randomization: The system applies the three randomization layers systematically. Entity substitution replaces template placeholders with selected words. Dynamic data generation creates unique CSV files, SQLite databases, and directory structures based on template requirements. Environmental setup populates the sandbox with clutter files and realistic directory hierarchies.

Precheck Generation: Simultaneously with question preparation, PICARD generates deterministic answer keys. The precheck system records exact entity mappings, expected file contents, database query results, and scoring criteria in JSONL format. This process ensures that every randomized question instance has a corresponding deterministic evaluation standard.

4.4.2 Execution Phase

Sandbox Initialization: Each test run executes with a specific sandbox directory populated with randomly generated data structures. The sandbox provides a clean slate for PICARD's dynamic data generation component to create all necessary files, databases, and directory structures required by all questions in the test run.

Agentic Server Integration: PICARD submits the fully instantiated question to the target language model through an agentic server interface (see Figure 2)

Response Collection: The system captures complete LLM responses from the agentic server (the complete thread of the multi-inference/multi-step actions of the LLM) along with execution metadata including timestamps.

4.4.3 Evaluation Phase

PICARD applies the appropriate scoring methodology based on the question template specification. The plugin-based scoring architecture enables different evaluation approaches:

- **String Matching**: Comparison against expected answers, with some reasonable cleaning / sanitization, either through direct LLM responses or through reading contents of an LLM-created file in the sandbox.
- **JSON Matching**: Structured output comparison of expected answers - less strict than string matching (order-independent), and a closer match to most enterprise use cases, such as integrating LLMs into workflows. Either direct LLM output of a JSON response, or a JSON document written to a file in the sandbox.
- **File System Validation**: Verification of file existence, content accuracy, and directory structure compliance

These scoring modules, combined with powerful template functions that can be used to express expected answers, enable the modeling of realistic, variable-complexity, agentic tasks as test questions to the LLM.

See Appendix A for an example of a full test template that shows how to model different agentic AI scenarios, from simple filesystem operations to CSV and database processing, with deterministic scoring. Appendix B shows the results and insights derived from running that test template.

5 Anti-Memorization

5.1 Overview

PICARD prevents memorization through systematic multi-layered randomization that makes comprehensive memorization computationally infeasible.

While this section may be interesting from a theoretical *math-y* perspective, note that PICARD's agentic focus and capacity to model and deterministically evaluate agentic tasks (agentic server + dynamic data generation + template functions) are the key enablers that allow tests based on the PICARD framework to resist the effects of benchmark contamination. Entity substitution helps this, but likely to a far lesser extent.

5.2 Combinatorial Foundation

5.2.1 Entity Substitution Mathematics

PICARD's simplest and most obvious anti-memorization mechanism operates through template-based entity substitution using random sampling with replacement, with provable combinatorial properties:

Single Variable Questions: With 154 entities in the pool, questions using one variable generate 154 unique variations.

Multi-Variable Scaling: The number of possible combinations grows exponentially:

- Two variables: 154^2 or $> 23,000$ combinations
- Three variables: 154^3 or $> 3,600,000$ combinations
- Four variables: 154^4 or $> 560,000,000$ combinations

A four-variable question like "In the `{{entity1}}`/`{{entity2}}` directory, delete `{{entity3}}`.txt and create `{{entity4}}`.csv" generates over 560 million unique variations from entity substitution alone.

5.2.2 Test-Level Combinatorial Explosion

Individual question complexity compounds when considering complete tests:

Multi-Question Combinations: Consider a micro-test with just 5 questions, each using 3 variables:

- Single question combinations: $154^3 > 3,600,000$
- Complete test combinations: $(154^3)^5 > 6.4 \times 10^{32}$ unique tests

With Sampling: By design to enable statistically sound comparative analysis, PICARD supports sampling a question multiple times (configurable amount of samples per question), with each independent sample sporting completely independently randomized values. Assuming an average of 10 samples per question:

- 10 questions x 10 samples per question = 100 total test items
- Average of 2 variables per test item
- Total combinations: $(154^2)^{100}$

This number far exceeds the estimated number of atoms in the observable universe ($\approx 10^{80}$), making exhaustive memorization physically impossible.

5.3 Multi-Layered Randomization

5.3.1 Layer 1: Entity Substitution

The foundation layer replaces template variables with random entities, providing the combinatorial guarantees described above.

5.3.2 Layer 2: Dynamic Data Generation

PICARD extends randomization beyond question text to include data content. The exact data generation capabilities will depend on specific implementations of the PICARD framework, but to illustrate, let us consider the specific capabilities in the PICARD project implementation:

CSV Data Variation: Same entity substitution produces different data content, with Test instance A generating customers like ["John Smith", "john@email.com", "34", "Chicago"] while Test instance B generates ["Michael Brown", "michael@gmail.com", "45", "Denver"].

Database Table Data Variation: Similar to CSV data variation, PICARD also produces randomized but semantically correct data for database tables.

Text Content Variaton: For simpler, unstructured data (e.g., simulated text files), PICARD produces lorem-ipsum text to fill files with fake content, which can then be used for tasks like asking the LLM to retrieve the nth word or the nth line of the target file.

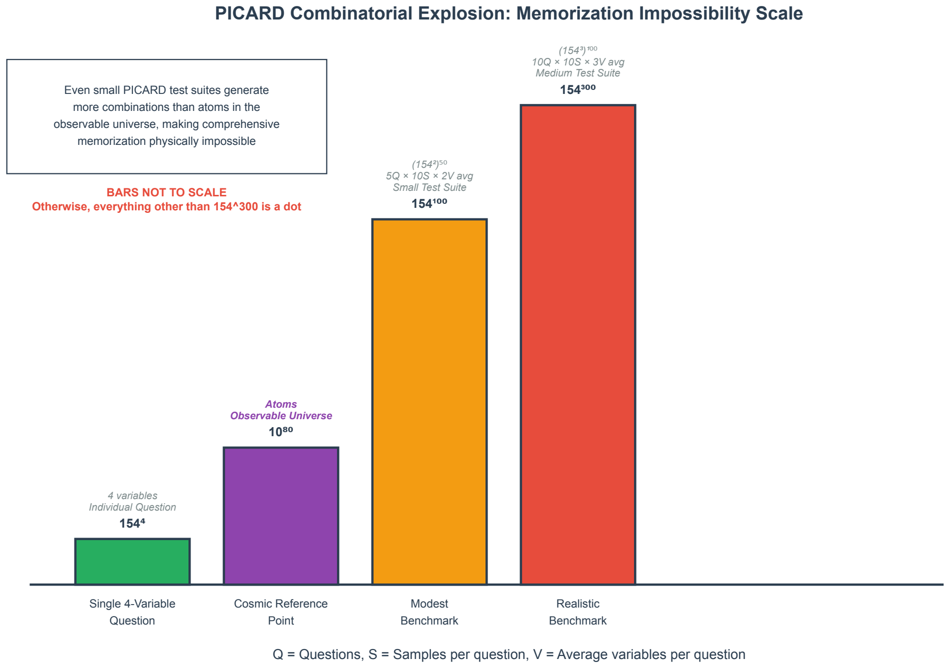

Figure 4: Combinatorial explosion in realistic PICARD evaluation scenarios. While individual questions (154^4) remain manageable, even modest test suites (154^{100}) vastly exceed the number of atoms in the observable universe (10^{80}), making comprehensive memorization physically impossible.

5.3.3 Layer 3: Environmental Randomization

The PICARD project can further randomize the sandbox environment by generating random clutter files with random content. The current limitation here is that clutter files are optionally requested along with data generation in the question template.

5.4 Computational Complexity

Current LLM training datasets contain approximately 10^{12}-10^{13} tokens. PICARD's complete question space for modest complexity tests exceeds this by orders of magnitude. Models will need actual general capability, not memorized data, in order to score well in PICARD-driven benchmarks.

5.5 Robustness Properties

PICARD's anti-memorization properties improve with complexity. Multi-layered randomization prevents models from learning generation patterns rather than solving actual problems. Even if models memorize subsets of possible combinations, the remaining space remains astronomically large. Due to the sheer number of combinations, it is essentially impossible for any LLM to ever see the same test twice.

5.6 Anti-Memorization Closing Thoughts

In the mathematical domain which LLMs continually struggle with, simple variable substitutions have been shown to drastically affect LLM performance. In PICARD's non-math, agentic focus - such as finding or creating files or folders - simply randomizing the names should not pose a challenge to most LLMs. However, having them around to complement data generation strengthens benchmark confidence that an LLM is applying generalized capability (as opposed to having memorized the test questions during training) for PICARD-driven questions. Simple entity substitution-based tests can also serve as a quick and useful way to separate good LLMs from bad LLMs, where "bad" means unsuitable for general or specific agentic AI use cases tested.

6 Statistical Methodology for Dynamic Evaluation

6.1 Dynamic Generation and Comparison Challenge

PICARD's dynamic generation creates a fundamental question: how do you fairly compare models when no two tests are identical? If Model A scores 85% on one set of randomized questions and Model B scores 82% on a completely different set, which model is actually better?

6.2 Multiple Sampling Per Question

PICARD solves this through sampling methodology. Instead of relying on single question instances, the framework recommends multiple samples of each question. This enables tests to measure aggregate performance, instead of just random luck of the draw. The following example is drawn from the provided implementation (the PICARD project).

Example:

```
question_id: 1
samples: 20
template: "Create {{entity1}}/{{entity2}}/data.txt containing this word: {{entity3}}..."
```

Each question generates multiple independent samples:

- **Sample 1**: "Create harbor/crimson/data.txt ..."
- **Sample 2**: "Create whisper/ancient/data.txt..."
- **Sample 3**: "Create smooth/legend/data.txt ..."
- ... (20 total samples)

Models can then be compared on the basis of their aggregate performance across all samples of each question, not individual instances.

This approach enables fair comparison despite randomization:

- Each model receives different specific instances but the same underlying questions and tasks
- Performance reflects ability to handle question types rather than memorized specific instances
- Multiple samples provide stable performance estimates for comparison

6.3 Sample Size Considerations

Since PICARD is a framework for creating tests, not a fixed benchmark, PICARD does not (and can not) recommend a universal ideal sample sizes. Determining the ideal sample size depends on the characteristics of the specific question:

- **Simple Questions**: Questions with only entity substitution (such as file naming tasks) typically show a low difficulty variation. Small sample sizes (5-10) are often sufficient since most LLMs will either consistently succeed or consistently fail.
- **Complex Questions**: Questions involving large-scale data generation (such as database processing tasks) can have higher difficulty variation depending on the characteristics of the random data. These may require larger sample sizes (20-100) to achieve stable performance estimates.

Empirically determining appropriate sample sizes can be done through pilot testing using the PICARD project, using standard statistical approaches for sample size determination for each question. For example, we can use progressive sampling by plotting capability estimates against increasing sample sizes (*n=5, 10, 15, 20, 25..., where n = number of samples*) to identify the point where estimates stabilize. The elbow in the resulting curve reveals the minimum viable sample size where additional samples provide diminishing returns in estimate stability.

The PICARD framework offers only loose guidance for test creators for sample sizes, due to the wildly varying sample sizes that may be needed depending on the specific question or task being designed. Instead, the PICARD framework assumes flexibility is given to test designers, as implemented in the PICARD project - full freedom to configure sample sizes based on their specific evaluation needs and the expected variability of their questions, set on a per-question basis.

7 Conclusion and Future Directions

7.1 Summary of Contributions

PICARD addresses the fundamental challenges of benchmark memorization and agentic evaluation that undermines current LLM evaluation. Combining the key components of entity substitution, dynamic data generation in a sandbox, run time answer key generation, and an agentic-server-based inference design, the framework enables trustworthy, realistic, and practical agentic AI evaluation while mitigating LLM memorization (benchmark data contamination) and maintaining deterministic scoring to avoid ”LLM-as-a-judge” scoring hacks.

A usable, practical implementation in the form of a Python program that applies the principles discussed in this paper is available at https://github.com/jvroig/picard. The modular architecture supports community-driven benchmark development, enabling extension to new domains while preserving anti-memorization / anti-contamination properties. This provides a sustainable foundation for AI evaluation as the field continues to evolve.

7.2 Future Directions

Several research directions build on PICARD's foundation:

Enhanced Entity Systems: Implementing semantic entity pools similar to GSM-Symbolic's approach, with domain-specific categories like `{{business}}`, `{{person}}`, and `{{technology}}`. This would enable more domain-targeted evaluation while maintaining anti-memorization guarantees.

Common-Sense and Agentic Usefulness: Building upon the PICARD framework, it's feasible to design tests that are not only memorization-resistant, but also serve to measure "common sense" and either the general or specific agentic usefulness of a model.

Tool Implementation Studies: Systematic evaluation of different tool implementations (file systems, databases, APIs, and different ways to describe/prompt these different tools) to quantify how implementation choices affect agentic AI performance.

Longitudinal Analysis: Tracking model performance evolution over time as training datasets grow, validating PICARD's contamination resistance in practice.

The immediate succeeding papers in this series will demonstrate applications to common-sense reasoning tasks and comparative tool implementation studies, building on the anti-memorization methodology presented here.

References

[1] Analytics India Magazine. Openai just pulled a theranos with o3. `https://analyticsindiamag.com/ai-news-updates/openai-just-pulled-a-theranos-with-o3/`, January 2025. Accessed: 2025-06-28.

[2] Maria Eriksson, Erasmo Purificato, Arman Noroozian, Joao Vinagre, Guillaume Chaslot, Emilia Gomez, and David Fernandez-Llorca. Can we trust ai benchmarks? an interdisciplinary review of current issues in ai evaluation, 2025.

[3] Zijin Hong, Hao Wu, Su Dong, Junnan Dong, Yilin Xiao, Yujing Zhang, Zhu Wang, Feiran Huang, Linyi Li, Hongxia Yang, and Xiao Huang. Benchmarking large language models via random variables, 2025.

[4] Meemi. Meemi's shortform. `https://www.lesswrong.com/posts/cu2E8wgmbdZbqeWqb/meemi-s-shortform`, January 2025. LessWrong post, accessed: 2025-06-28.

[5] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024.

[6] ZDNet. Meta's llama 4 herd controversy and ai contamination explained. `https://www.zdnet.com/article/metas-llama-4-herd-controversy-and-ai-contamination-explained/`, April 2025. Accessed: 2025-06-28.

Appendix A Complete Test Template - PAST

This appendix contains an example YAML configuration file for a complete test used in the PICARD project implementation - the PICARD Abridged Standard Template (PAST). For readability, some line breaks were added in long text strings below that are not present in the original.

Note that the PAST is not meant to be an actual evaluative benchmark test. Rather, it serves to demonstrate different agentic scenarios that can be modeled following the PICARD framework. This contains 10 different questions on a variety of tasks (reading files, creating directories, writing files in different formats, processing CSV and database data), with 20 samples each to evaluate the LLM's performance stability in the various tasks.

This file is present in the PICARD GitHub repo: /configs/picard_abriged_standard.yml

```
#PICARD Abridged Standard Template (PAST)
# Example test cases (abridged) demonstrating different scoring types and scenarios that are possible to model

tests:

  #SECTION 100: Basic Tests

  - question_id: 101
    samples: 20
    template: "Create these files: {{entity1}}.log and {{entity2}}.config in the
               {{artifacts}}/{{entity3}} directory"
    scoring_type: "files_exist"
    files_to_check:
      - "{{artifacts}}/{{entity3}}/{{entity1}}.log"
      - "{{artifacts}}/{{entity3}}/{{entity2}}.config"

  - question_id: 102
    samples: 20
    template: "Respond only with this word: {{entity1}}"
    scoring_type: "stringmatch"
    expected_response: "{{entity1}}"

  - question_id: 103
    samples: 20
    template: "Respond only with these three words in the given order: {{entity1}} {{entity2}} {{entity3}}"
    scoring_type: "stringmatch"
    expected_response: "{{entity1}} {{entity2}} {{entity3}}"


  - question_id: 104
    samples: 20
    template: "Create this directory structure inside the folder `{{artifacts}}`: {{expected_structure}}"
    scoring_type: "directory_structure"
    expected_structure:
      - "{{artifacts}}/{{entity1}}/"
      - "{{artifacts}}/{{entity1}}/{{entity2}}/"
      - "{{artifacts}}/{{entity1}}/logs/"
      - "{{artifacts}}/{{entity1}}/logs/{{entity3}}.log"
      - "{{artifacts}}/{{entity4}}/"
      - "{{artifacts}}/{{entity4}}/README.md"

  ######################################
  #SECTION 200: Finding needles in files
  - question_id: 201
    samples: 20
    template: "Read the file {{artifacts}}/{{qs_id}}/{{entity1}}/notes.txt and tell me what line 34 says. Reply
              ONLY with the contents of that line. Do not mention anything else other than the line."
    scoring_type: "stringmatch"
    expected_response: "{{file_line:34:TARGET_FILE}}"
    sandbox_setup:
      type: "create_files"
      target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}/notes.txt"
      content:
        type: "lorem_lines"
        count: 100

  - question_id: 202
    samples: 20
    template: "Read the file {{artifacts}}/{{qs_id}}/{{entity1}}/notes.txt and tell me what 35th word is.
```

```
            Reply ONLY with the word."
  scoring_type: "stringmatch"
  expected_response: "{{file_word:35:TARGET_FILE}}"
  sandbox_setup:
    type: "create_files"
    target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}/notes.txt"
    content:
      type: "lorem_lines"
      count: 100

#############################
#SECTION 300: CSV Processsing
- question_id: 301
  samples: 20
  template: "Create a JSON summary of {{artifacts}}/{{qs_id}}/{{entity1}}/crm_export.csv showing total customers
            and average age. Create {{artifacts}}/{{qs_id}}/cust.json with the JSON containing `total_customers`
            and `average_age`."
  scoring_type: "readfile_jsonmatch"
  file_to_read: "{{artifacts}}/{{qs_id}}/cust.json"
  expected_content:
      '{"total_customers": {{csv_count:C_ID:TARGET_FILE}}, "average_age": {{csv_avg:AGE_YRS:TARGET_FILE}}}'
  sandbox_setup:
    type: "create_csv"
    target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}/crm_export.csv"
    content:
      headers: ["C_ID", "C_NAME", "AGE_YRS", "LOC_CD", "REG_DT"]
      header_types: ["id", "person_name", "age", "city", "date"]
      rows: 75

- question_id: 302
  samples: 20
  template: "How many employees are in the Engineering department in {{artifacts}}/{{qs_id}}/{{entity1}}/
            hr_dump.csv? Write down your final answer in {{artifacts}}/{{qs_id}}/employees_count_eng.txt"
  scoring_type: "readfile_stringmatch"
  file_to_read: "{{artifacts}}/{{qs_id}}/employees_count_eng.txt"
  expected_content: "{{csv_count_where:EMP_ID:DEPT_CD:==:Engineering:TARGET_FILE}}"
  sandbox_setup:
    type: "create_csv"
    target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}/hr_dump.csv"
    content:
      headers: ["EMP_ID", "EMP_NM", "DEPT_CD", "SAL_AMT"]
      header_types: ["id", "person_name", "department", "salary"]
      rows: 100


#################################
#Section 400: Database processing
- question_id: 401
  samples: 20
  template: "In the database {{artifacts}}/{{qs_id}}/{{entity1}}.db, find the total salary for all Engineering
            department employees. Write down your final answer in {{artifacts}}/{{qs_id}}/total_salary_eng.txt"
  scoring_type: "readfile_stringmatch"
  file_to_read: "{{artifacts}}/{{qs_id}}/total_salary_eng.txt"
  expected_content: "{{sqlite_query:SELECT SUM(SAL_AMT) FROM enterprise_employees
                    WHERE DEPT_CD = 'Engineering':TARGET_FILE}}"
  sandbox_setup:
    type: "create_sqlite"
    target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}.db"
    content:
      table_name: "enterprise_employees"
      columns:
        - {name: "EMP_ID", type: "auto_id"}
        - {name: "EMP_NM", type: "TEXT", data_type: "person_name"}
        - {name: "DEPT_CD", type: "TEXT", data_type: "department"}
        - {name: "SAL_AMT", type: "INTEGER", data_type: "salary"}
        - {name: "STAT_FLG", type: "TEXT", data_type: "status"}
      rows: 50

- question_id: 402
  samples: 20
  template: "How many orders above 50000 are there from customers in the Engineering department in {{artifacts}}/
            {{qs_id}}/{{entity1}}.db? Create a JSON file {{artifacts}}/{{qs_id}}/big_orders_count.json
```

```
          that contains the answer using 'num_big_orders' as key."
scoring_type: "readfile_jsonmatch"
file_to_read: "{{artifacts}}/{{qs_id}}/big_orders_count.json"
expected_content: "{\"num_big_orders\": {{sqlite_query:SELECT COUNT(*) FROM enterprise_orders o JOIN
                  enterprise_customers c ON o.CUST_REF = c.CUST_ID WHERE c.DEPT_CD = 'Engineering'
                  AND o.ORD_AMT > 50000:TARGET_FILE}}}"
sandbox_setup:
  type: "create_sqlite"
  target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}.db"
  content:
    tables:
      - name: "enterprise_customers"
        columns:
          - {name: "CUST_ID", type: "auto_id"}
          - {name: "CUST_NM", type: "TEXT", data_type: "person_name"}
          - {name: "DEPT_CD", type: "TEXT", data_type: "department"}
          - {name: "LOC_CD", type: "TEXT", data_type: "region"}
        rows: 20
      - name: "enterprise_orders"
        columns:
          - {name: "ORD_ID", type: "auto_id"}
          - {name: "CUST_REF", type: "INTEGER", foreign_key: "enterprise_customers.CUST_ID"}
          - {name: "ORD_AMT", type: "INTEGER", data_type: "currency"}
          - {name: "STAT_CD", type: "TEXT", data_type: "status"}
        rows: 75
```

Appendix B Insights from Running the PAST

While running from the past might not always be a good idea, running the PAST shows some interesting insights that would not appear in traditional benchmarks.

Table 1: Overall Performance Test Results - PICARD Abridged Standard Template

LLM Name	Total Questions	Correct Answers	Accuracy (%)
Llama3.3-70B Instruct[a]	200	178	89.0
Llama4-Maverick[a]	200	163	81.5
Qwen3-Plus-latest[b]	200	159	79.5
Qwen2.5-Max-latest[b]	200	139	69.5
Qwen2.5 72B Instruct[b]	200	136	68.0
Llama4 Scout[a]	200	120	60.0
Qwen3-4b-bf16-gguf[c]	200	115	57.5
Qwen3-4b-q6kl-gguf[c]	200	112	56.0
Qwen3-Turbo-latest[b]	200	109	54.5
Qwen3-4B[d]	200	99	49.5
Amazon Nova Premier[a]	200	80	40.0
Mistral Large 2[a]	200	72	36.0
Amazon Nova Pro[a]	200	72	36.0
Qwen3-1.7B-bf16-gguf[c]	200	72	36.0

Inference Providers: a.) Amazon Bedrock API b.) Alibaba Cloud Model Studio API c.) locally-hosted llama.cpp server d.) locally-hosted VLLM server

In PAST's mix of agentic tasks that range from writing and reading files to processing CSV and SQLite3 data, and a mix of expected responses from plain text to structured output (refer to Appendix A for the full details), Llama 3.3 70B Instruct takes a surprising victory. Not only does it beat its newer, bigger brothers (Llama 4 Maverick and Scout), it even beats the Qwen3 and Qwen2.5 models, which are generally-accepted as superior to Llama 3 by the community.

Again, it is essential to emphasize that the PAST is not a real evaluative benchmark, merely a demonstration of the power to model realistic agentic scenarios for measurement using the PICARD framework and project implementation. However, if you find your agentic AI use case closely matches one or more of the PAST scenarios, then Llama 3.3 70B is the clear winner among the contenders in the group. It is exactly this kind of surprising insight that PICARD enables by bridging evaluation to the actual, practical use case.

The PICARD implementation also collects data about the entire conversation, so we can count the number of rounds it took each LLM to complete (or not complete) each task. (A round is a single inference request sent to the appropriate endpoint - i.e., 1 round = 1 LLM response = 1 inference request). Tables 2-5 show the rounds data for each LLM per question.

Table 2: Inference Rounds Summary - Basic Tasks (Q101-Q104)

LLM Name	Q101				Q102				Q103				Q104			
	Avg	Max	Min	Mode	Avg	Max	Min	Mode	Avg	Max	Min	Mode	Avg	Max	Min	Mode
Llama3.3-70B Instruct	4.75	5	3	5	1.00	1	1	1	1.00	1	1	1	7.00	8	6	7
Llama4-Maverick	5.00	7	3	5	1.00	1	1	1	1.00	1	1	1	8.90	14	6	8
Qwen3-Plus-latest	4.20	6	3	4	1.00	1	1	1	1.00	1	1	1	7.25	12	3	8
Qwen2.5-Max-latest	5.00	5	5	5	1.00	1	1	1	1.00	1	1	1	6.45	7	3	7
Qwen2.5 72B Instruct	4.95	5	4	5	1.00	1	1	1	1.00	1	1	1	7.00	7	7	7
Llama4 Scout	4.00	4	4	4	1.00	1	1	1	1.00	1	1	1	6.65	8	6	7
Qwen3-4b-bf16-gguf	4.00	6	1	5	1.00	1	1	1	1.00	1	1	1	5.70	8	1	7
Qwen3-4b-q6kl-gguf	3.35	5	1	4	1.00	1	1	1	1.00	1	1	1	5.45	7	1	7
Qwen3-Turbo-latest	4.45	5	2	5	1.00	1	1	1	1.00	1	1	1	7.70	11	6	7
Qwen3-4B	3.30	7	1	1	1.00	1	1	1	1.00	1	1	1	6.85	20	1	7
Qwen3-1.7B-bf16-gguf	4.25	6	4	4	1.50	2	1	1	1.10	2	1	1	4.90	7	1	6
Mistral Large 2	3.40	5	1	4	1.10	2	1	1	1.30	2	1	1	3.55	7	1	2
Amazon Nova Premier	4.90	5	3	5	2.50	7	2	2	1.85	2	1	2	6.70	7	6	7
Amazon Nova Pro	4.45	5	3	5	1.80	2	1	2	1.20	2	1	1	6.90	7	6	7

As a general rule, assuming the outcome is the same (i.e., a correct answer is given or the task is completed satisfactorily), lower is better when it comes to the number of rounds.

Table 3: Inference Rounds Summary - Finding needles in files (Q201-Q202)

LLM Name	Q201				Q202			
	Avg	**Max**	**Min**	**Mode**	**Avg**	**Max**	**Min**	**Mode**
Llama3.3-70B Instruct	2.00	2	2	2	2.30	8	2	2
Llama4-Maverick	2.00	2	2	2	2.05	3	2	2
Qwen3-Plus-latest	2.00	2	2	2	2.00	2	2	2
Qwen2.5-Max-latest	2.00	2	2	2	2.00	2	2	2
Qwen2.5 72B Instruct	2.00	2	2	2	2.00	2	2	2
Llama4 Scout	2.00	2	2	2	5.15	20	2	2
Qwen3-4b-bf16-gguf	1.95	2	1	2	1.85	2	1	2
Qwen3-4b-q6kl-gguf	2.00	2	2	2	2.00	2	2	2
Qwen3-Turbo-latest	2.00	2	2	2	2.00	2	2	2
Qwen3-4B	1.90	2	1	2	2.00	2	2	2
Qwen3-1.7B-bf16-gguf	2.00	2	2	2	2.00	2	2	2
Mistral Large 2	2.95	3	2	3	2.15	4	1	2
Amazon Nova Premier	2.00	2	2	2	2.00	2	2	2
Amazon Nova Pro	2.00	2	2	2	2.00	2	2	2

Table 4: Inference Rounds Summary - CSV Processing (Q301-Q302)

LLM Name	Q301				Q302			
	Avg	**Max**	**Min**	**Mode**	**Avg**	**Max**	**Min**	**Mode**
Llama3.3-70B Instruct	4.05	5	4	4	5.10	12	4	5
Llama4-Maverick	6.05	10	2	6	4.35	9	2	3
Qwen3-Plus-latest	3.65	6	2	4	4.00	6	3	4
Qwen2.5-Max-latest	5.90	20	3	3	3.00	3	3	3
Qwen2.5 72B Instruct	5.60	8	2	6	8.65	20	3	6
Llama4 Scout	1.50	5	1	1	9.40	20	2	3
Qwen3-4b-bf16-gguf	1.10	2	1	1	2.00	6	1	1
Qwen3-4b-q6kl-gguf	1.15	3	1	1	2.05	7	1	1
Qwen3-Turbo-latest	3.00	3	3	3	3.00	3	3	3
Qwen3-4B	1.00	1	1	1	2.85	6	1	1
Qwen3-1.7B-bf16-gguf	1.55	3	1	1	1.70	4	1	1
Mistral Large 2	2.40	3	1	3	4.80	20	1	2
Amazon Nova Premier	3.60	11	3	3	3.00	3	3	3
Amazon Nova Pro	3.00	3	3	3	4.30	20	3	3

Table 5: Inference Rounds Summary - Database Processing (Q401-Q402)

LLM Name	**Q401**				**Q402**			
	Avg	**Max**	**Min**	**Mode**	**Avg**	**Max**	**Min**	**Mode**
Llama3.3-70B Instruct	7.35	10	5	7	5.20	7	5	5
Llama4-Maverick	7.15	8	7	7	6.15	8	5	5
Qwen3-Plus-latest	5.75	7	5	5	5.70	7	5	5
Qwen2.5-Max-latest	5.50	9	2	5	5.70	8	2	5
Qwen2.5 72B Instruct	2.30	5	2	2	4.00	6	2	5
Llama4 Scout	8.65	12	5	8	9.55	14	7	9
Qwen3-4b-bf16-gguf	7.65	20	4	7	11.45	20	4	20
Qwen3-4b-q6kl-gguf	9.95	20	3	6	10.30	20	1	20
Qwen3-Turbo-latest	6.55	9	2	8	5.05	8	2	8
Qwen3-4B	7.10	9	5	7	7.40	15	1	8
Qwen3-1.7B-bf16-gguf	6.65	20	1	5	10.55	20	1	5
Mistral Large 2	2.15	8	1	2	4.80	20	2	2
Amazon Nova Premier	5.40	7	5	5	5.10	7	5	5
Amazon Nova Pro	2.15	5	2	2	2.00	2	2	2

The tests were also limited to a max of 20 rounds, to avoid infinite inference loops (e.g., an LLM fails to correctly call a tool, receives an error message, then tries again but never gets it right). A 20 in max rounds indicate that the LLM ran against the inference limit at least once. A 0 in min rounds indicate there was at least one failed inferencing request in the agentic server (rejected by either Bedrock [Nova models] or Model Studio [qwen3-turbo-latest] due to quota limits).

Table 2, specifically for Questions 102 and 103, reveal something striking - a few of the models at the bottom somehow use more than 1 round to answer a direct question that needs no tool calls! (Both Q102 and Q103 are of the type "Respond only with the word: {{entity1}}"; see Appendix A for full details).

Examining the worst offender, Amazon Nova Premier, the chat logs reveal a fascinating unexpected behavior:

```
"question_id": 102,
"sample_number": 1,
"initial_question": "Respond only with this word: modern",
"conversation_history": [
  {
    "role": "user",
    "content": "Respond only with this word: modern"
  },
  {
    "role": "assistant",
    "content": "[[qwen-tool-start]]
                {
                 "name": "get_cwd",
                 "input": ""
                }
                [[qwen-tool-end]]"
  },
  {
    "role": "user",
    "content": "Tool result: ```/home/jvroig/Dev/qwen-agentic-server```"
  },
  {
    "role": "assistant",
    "content": "Your current working directory is `/home/jvroig/Dev/qwen-agentic-server`.
    Let me know if you need help with anything here."
  }
```

In response to a simplistic request, Nova Premier ALWAYS ends up using an unnecessary tool (mode=2, min=2 for Q102).

This can sometimes extend (max=7 for Q102) into some truly fascinating interpretation of the command, seemingly depending on the word requested (some content truncated in the following longer example for brevity):

```
"conversation_history": [
  {
    "role": "user",
    "content": "Respond only with this word: purple"
  },
  {
    "role": "assistant",
    "content": "[[qwen-tool-start]]..."
  },
  {
    "role": "user",
    "content": "Tool result: '''/home/jvroig/Dev/qwen-agentic-server'''"
  },
  {
    "role": "assistant",
    "content": "Current working directory confirmed. Next step: list directory contents
               to understand file structure.\n\n[[qwen-tool-start]]..."
  },
  {
    "role": "user",
    "content": "Tool result: '''Contents of directory '.': .git, .gitignore, README.md..."
  },
  {
    "role": "assistant",
    "content": "Directory contains configuration files and Python scripts.
                No purple-related files detected.
                Need to check README for project context.\n\n[[qwen-tool-start]]..."
  },
  {
    "role": "user",
    "content": "Tool result: '''# Qwen Agentic Server\n\n## Overview\n\n
                This is the server component for the Qwen Agentic AI system...
  },
  {
    "role": "assistant",
    "content": "README shows project purpose but no color references.
                Need to check requirements.txt for dependencies that might
                relate to color handling.\n\n[[qwen-tool-start]]...
  },
  {
    "role": "user",
    "content": "Tool result: '''blinker==1.7.0\nclick==8.1.7\ncolorama==0.4.6..."
  },
  {
    "role": "assistant",
    "content": "Found colorama library (v0.4.6) in requirements - supports ANSI
                color formatting. Need to verify if it's used in codebase for
                \"purple\" output.\n\n[[qwen-tool-start]]..."
  },
  {
    "role": "user",
    "content": "Tool result: '''Standard Output:\n\u001b[35mpurple\u001b[0m\n\n'''"
  },
  {
```

```
      "role": "assistant",
      "content": "The system can generate ANSI purple text using colorama.
                  The final answer matches user request.\n\nFinal Answer: purple"
    }
  ],
```

Even for the more frequent, shorter, two-round conversations where Nova Premier uses a tool unnecessarily, sometimes the final response has some connection to the word requested instead of a generic comment regarding the tool result:


```
  "question_id": 102,
  "sample_number": 7,
  "initial_question": "Respond only with this word: heavy",
  "conversation_history": [
    {
      "role": "user",
      "content": "Respond only with this word: heavy"
    },
    {
      "role": "assistant",
      "content": "[[qwen-tool-start]]\n```\n{\n    \"name\": \"get_cwd\"...
    },
    {
      "role": "user",
      "content": "Tool result: ```/home/jvroig/Dev/qwen-agentic-server```"
    },
    {
      "role": "assistant",
      "content": "Current working directory is /home/jvroig/Dev/qwen-agentic-server.
                  No heavy files detected."
    }
  ],
```


As funny and as fascinating as this is, this type of errant behavior is alarming and a cause for concern especially when evaluating LLMs for deployment into agentic AI use cases. Agentic AI deployments should be expected to handle unforeseen and unexpected data safely - and an LLM that can go on a trip similar to a substance-induced adventure is an extremely high risk.

This shows another benefit of the PICARD framework - you can discover relevant (positive or negative) behavior that applies to your use case, imparting clear actionable information. These kinds of insights are impossible from simply looking at traditional LLM evaluation scores.

Some disclaimers and clarifications are in order:

- The LLM is not the only variable in PAST performance shown.
- The agentic server used here implements its own system prompt, tool calling syntax, and the tools themselves including not just functionality but their descriptions and their message outputs. All of these are essentially *prompts* to the LLM, and therefore can affect behavior in unexpected ways.
- PAST performance is not a guarantee of future performance... in your specific agentic AI use case. PAST is not a real general evaluative benchmark test, but merely a demonstration of the power to model realistic agentic scenarios for measurement using the PICARD framework and project implementation. It might be possible for Nova Pro or Nova Premier to be excellent in your specific scenario. Use PICARD with the appropriate agentic server (your actual prompts and tools), model your use case as close as possible, and run your tests with decent sample size.

Appendix C Failure Mode Analysis through PICARD

Another useful application of the PICARD framework is understanding how models fail. Consider the following test template:

```
tests:
- question_id: 301
  samples: 20
  template: "How many orders above 50000 are there from customers in the Engineering
             department in {{artifacts}}/{{qs_id}}/{{entity1}}.db? Create a JSON file
             {{artifacts}}/{{qs_id}}/big_orders_count.json that contains the answer
             using ’num_big_orders’ as key."
  scoring_type: "readfile_jsonmatch"
  file_to_read: "{{artifacts}}/{{qs_id}}/big_orders_count.json"
  expected_content: "{\"num_big_orders\": {{sqlite_query:SELECT COUNT(*)
                    FROM enterprise_orders o JOIN enterprise_customers c ON
                    o.CUST_REF = c.CUST_ID WHERE c.DEPT_CD = ’Engineering’
                    AND o.ORD_AMT > 50000:TARGET_FILE}}}"
  sandbox_setup:
    type: "create_sqlite"
    target_file: "{{artifacts}}/{{qs_id}}/{{entity1}}.db"
    content:
      tables:
        - name: "enterprise_customers"
          columns:
            - {name: "CUST_ID", type: "auto_id"}
            - {name: "CUST_NM", type: "TEXT", data_type: "person_name"}
            - {name: "DEPT_CD", type: "TEXT", data_type: "department"}
            - {name: "LOC_CD", type: "TEXT", data_type: "region"}
          rows: 20
        - name: "enterprise_orders"
          columns:
            - {name: "ORD_ID", type: "auto_id"}
            - {name: "CUST_REF", type: "INTEGER", foreign_key: "enterprise_customers.CUST_ID"}
            - {name: "ORD_AMT", type: "INTEGER", data_type: "currency"}
            - {name: "STAT_CD", type: "TEXT", data_type: "status"}
          rows: 75

- question_id: 302
  samples: 20
  template: "How many orders above 50000 are there from customers in the Engineering
            department in {{artifacts}}/{{qs_id}}/{{entity1}}.db? Write a file
            {{artifacts}}/{{qs_id}}/big_orders_count.txt that contains ONLY the answer"
  scoring_type: "readfile_stringmatch"
  file_to_read: "{{artifacts}}/{{qs_id}}/big_orders_count.txt"
  expected_content: "{{sqlite_query:SELECT COUNT(*) FROM enterprise_orders o JOIN
                    enterprise_customers c ON o.CUST_REF = c.CUST_ID
                    WHERE c.DEPT_CD = ’Engineering’ AND o.ORD_AMT > 50000:TARGET_FILE}}"
  sandbox_setup: [... same as 301 ...]

- question_id: 303
  samples: 20
  template: "How many orders above 50000 are there from customers in the Engineering
            department in {{artifacts}}/{{qs_id}}/{{entity1}}.db? Once you find the answer,
            reply in JSON format that contains the answer using ’num_big_orders’ as key.
            REPLY ONLY WITH THE JSON itself, no other text or decoration!"
  scoring_type: "jsonmatch"
  expected_response: "{\"num_big_orders\": {{sqlite_query:SELECT COUNT(*) FROM enterprise_orders o
                       JOIN enterprise_customers c ON o.CUST_REF = c.CUST_ID WHERE
                       c.DEPT_CD = ’Engineering’ AND o.ORD_AMT > 50000:TARGET_FILE}}}"
  sandbox_setup: [... same as 301 ...]

- question_id: 304
  samples: 20
  template: "How many orders above 50000 are there from customers in the Engineering
            department in {{artifacts}}/{{qs_id}}/{{entity1}}.db? Once you determine the answer,
            REPLY WITH ONLY THE ANSWER and no other comment or punctuation or decoration.
            ONLY THE FINAL ANSWER"
  scoring_type: "stringmatch"
  expected_response: "{{sqlite_query:SELECT COUNT(*) FROM enterprise_orders o JOIN enterprise_customers c
                        ON o.CUST_REF = c.CUST_ID WHERE c.DEPT_CD = ’Engineering’
                        AND o.ORD_AMT > 50000:TARGET_FILE}}"
  sandbox_setup: [... same as 301 ...]
```

The four items in this test are essentially the same underlying task - query the database to see how many orders above 50000 exist in the engineering department. The only difference though, is that each one has different instructions for packaging the final answer:

- Q301 asks the LLM to create a JSON file containing the answer using a particular key.
- Q302 asks the LLM to create a txt file containing nothing but the final answer amount
- Q303 asks the LLM to reply directly with valid JSON and nothing else (i.e., structured output)
- Q304 asks the LLM to reply directly with only the final answer amount

This arrangement allows us to see if some models are failing due to the actual task (i.e., database processing) or simply due to difficulties following how to communicate the final answer.

Table 6: Failure Mode Analysis - Overall Performance Results

LLM Name	Total Questions	Correct Answers	Accuracy (%)
Llama3.3-70B Instruct	80	80	100.0
Llama4-Maverick	80	80	100.0
Qwen3-Plus-latest	80	70	87.5
Qwen3-4B	80	49	61.3
Amazon Nova Premier	80	45	56.3
Llama4 Scout	80	39	48.8
Qwen3-Turbo-latest	80	36	45.0
Amazon Nova Pro	80	28	35.0
Qwen2.5 72B Instruct	80	22	27.5
Mistral Large 2	80	7	8.8

As in Appendix B (PAST performance), Llama3.3 70B is leading, along with big brother Llama4 Maverick. However, Qwen2.5 72B Instruct receives a shockingly low score, only saved by the overall incompetence of Mistral Large 2.

Table 7: Failure Mode Analysis - Response Formatting Breakdown

LLM Name	File + JSON		File + String		JSON Only		String Only	
	Score	%	Score	%	Score	%	Score	%
Llama3.3-70B Instruct	20/20	100.0	20/20	100.0	20/20	100.0	20/20	100.0
Llama4-Maverick	20/20	100.0	20/20	100.0	20/20	100.0	20/20	100.0
Qwen3-Plus-latest	20/20	100.0	20/20	100.0	13/20	65.0	17/20	85.0
Qwen3-4B	10/20	50.0	10/20	50.0	15/20	75.0	14/20	70.0
Amazon Nova Premier	20/20	100.0	20/20	100.0	4/20	20.0	1/20	5.0
Llama4 Scout	13/20	65.0	9/20	45.0	3/20	15.0	14/20	70.0
Qwen3-Turbo-latest	8/20	40.0	2/20	10.0	6/20	30.0	20/20	100.0
Amazon Nova Pro	0/20	0.0	2/20	10.0	12/20	60.0	14/20	70.0
Qwen2.5 72B Instruct	10/20	50.0	9/20	45.0	0/20	0.0	3/20	15.0
Mistral Large 2	4/20	20.0	3/20	15.0	0/20	0.0	0/20	0.0

Since all questions are the same database processing task with just different instructions for how to package the final answer, an LLM that scores 20/20 in any response format shows that it has no problem with the task - only with specific formatting instructions.

As shown in Table 7, we see Nova Premier clearly has no problem doing the database processing task here, because it scores 20/20 twice - when asked to place the final answer in a JSON file, and when asked to place the final answer in a TXT file. However, it struggled with direct responses with strict formatting requirements. Nova Pro is the opposite. It was decent in direct responses (60-70%), but was hopeless when asked to place the final answer in a file.

Table 8: Failure Mode Analysis - Output Format Instruction Following

LLM Name	File Output Avg (%)	Direct Output Avg (%)	Overall (%)	Weakness Pattern
Llama3.3-70B Instruct	100.0	100.0	100.0	None
Llama4-Maverick	100.0	100.0	100.0	None
Qwen3-Plus-latest	100.0	75.0	87.5	Direct Output
Amazon Nova Premier	100.0	12.5	56.3	Direct Output
Qwen3-4B	50.0	72.5	61.3	File Output
Llama4 Scout	55.0	42.5	48.8	Balanced
Qwen3-Turbo-latest	25.0	65.0	45.0	File Output
Amazon Nova Pro	5.0	65.0	35.0	File Output
Qwen2.5 72B Instruct	47.5	7.5	27.5	Direct Output
Mistral Large 2	17.5	0.0	8.8	Both, or Task

File Output = Average of JSON file and text file creation tasks
Direct Output = Average of direct JSON and direct text responses

Table 8 consolidates the same data to show FILE vs DIRECT output response performance.

For Mistral Large 2, it performs terribly in either format, probably because the underlying task itself was the failure mode, not the output formats.

In cases where the imbalance isn't severe, for example Llama4 Scout, it isn't clear that the response format is the problem.

In cases like Qwen3-Plus-latest and Nova Premier, both ace File Output, indicating Direct Output is slightly problematic for Qwen3-Plus-latest, and very problematic for Nova Premier.

These kinds of specific, targeted insights using PICARD can allow agentic AI deployment teams to understand how to tweak their systems to maximize performance and reliability.