Benchmarking Large Language Models on Statistical Hypothesis Testing

Alfred K. Adzika
Department of Computer Science
Ohio University
Athens, OH, USA
email@ohio.edu

***Abstract*—We study whether modern Large Language Models (LLMs) can reliably carry out statistical hypothesis testing. We present a reproducible benchmark that performs hypothesis testing as a holistic capability: selecting an appropriate test, computing statistics and p-values, and making decisions under a significance level. The framework evaluates models under multiple prompting strategies (Zero-Shot, Few-Shot, Chain-of-Thought, and Program-of-Thought), uses synthetic data with known ground truth, and reports standardized metrics (overall accuracy, decision accuracy, p-value tolerance, reasoning quality, latency). Empirical results on recent models (e.g., GPT-5, Claude 4.5 Opus, Gemini 2.5, grok) show clear performance differences: computation-grounded prompts (PoT) improve numerical fidelity, while CoT improves structure but remains vulnerable to arithmetic errors and completeness. We discuss strengths, limitations, and paths toward trustworthy AI systems that can participate in evidence-based scientific workflows leading to justifiable new discoveries.**

***Index Terms*—Large Language Models, Hypothesis Testing, Benchmarking, Scientific Reasoning, AI Evaluation**

I. Introduction

This work is a study in artificial intelligence, guided by a long-term aim of understanding how AI systems could help uncover new domains of knowledge. [1]. At its core, the project investigates one essential capability that underlies all scientific progress: hypothesis testing. Specifically, we study how well large language models (LLMs) can evaluate the validity of claims under controlled conditions.

In scientific research, progress depends not only on generating ideas, but also on the ability to test, validate, refine, and reject them. While LLMs have shown impressive performance in tasks such as text generation, reasoning, and problem-solving [2], [3], there is still limited systematic understanding of how well they perform at evaluating the correctness of claims in a way that resembles true hypothesis testing. If LLMs are ever to meaningfully support scientific discovery, they must be able not just to propose ideas, but to rigorously assess whether those ideas hold up under evidence and structured testing.

This project serves as groundwork toward measuring that capability. Our central objective is to design and implement a benchmark that evaluates how effectively modern LLMs can perform hypothesis testing. By formalizing hypothesis testing as a measurable task for language models, we aim to provide a standardized way to quantify how accurately these systems can judge the validity of claims, weigh evidence, and reach reliable conclusions. Concretely, we determine success around consistent test selection, calibrated p-value estimation, and correct reject/fail-to-reject decisions at a fixed α.

In the long term, this line of work connects to a much larger question: if future AI systems are to assist in the formation of new areas of study, then they must first demonstrate reliable performance in evaluating the foundational claims upon which new fields are built. This research does not attempt to establish new fields directly; rather, it focuses on building an evaluation infrastructure needed to assess whether LLMs are developing the kinds of reasoning and validation skills that such discovery would require.

Accordingly, this work will contribute:

- A formal benchmark for hypothesis testing with LLMs,
- An empirical evaluation of LLMs on this benchmark,
- And an analysis of their strengths, and limitations.

II. Related Work

A growing body of work has focused on evaluating the logical reasoning capabilities of large language models. Benchmarks such as LogicBench [4] and LogicAsker [5] systematically test LLMs on propositional and first-order logic, revealing that while modern models can solve surface-level inference tasks, they still struggle with deeper multi-step and rule-consistent reasoning. Multi-LogiEval [6] tests LLMs on harder kinds of logical thinking involving doing many steps, handling facts that change old conclusions, and switching between different types of logic. As the tasks get deeper or require more steps, model performance drops sharply. Recent survey work on logical reasoning in LLMs [7] consolidates these findings and highlights persistent gaps in deductive reliability. While these efforts provide critical insights into formal reasoning, they do not directly evaluate statistical hypothesis testing as a reasoning task.

Beyond pure logical inference, recent studies have examined whether LLMs can evaluate the correctness of claims and their own reasoning. Work on self-verification in logical reasoning [8] shows that LLMs often fail to reliably detect their own logical inconsistencies. FactReasoner [9] introduces a probabilistic framework for long-form factuality assessment, focusing on evidence-grounded verification of claims. Similarly, Trustworthy Reasoning [10] demonstrates that even when final answers are correct, intermediate reasoning steps frequently contain factual or logical errors. These works highlight the importance of explicitly assessing whether a claim is valid, but they focus mainly on factual accuracy and reasoning self-analysis, not on systematic statistical hypothesis testing.

More closely related to our work, several recent papers investigate whether LLMs can perform statistical reasoning from data. Liu et al. [11] propose a benchmark for quantitative and causal reasoning based on tabular and numerical data, showing that while LLMs can handle simple statistical patterns, they struggle with multi-variable dependencies and causal structure. Zhu et al. [12] systematically evaluate whether LLMs can function as reliable statisticians, testing tasks such as distribution estimation, hypothesis testing components, and statistical decision-making, and find that performance is highly unstable across test types and noise conditions.

Most directly aligned conceptually with our work, Tiwari [13] introduces a general framework for automated hypothesis testing using AI systems. Although this work proposes a procedural framework for automating the process, it does not offer a standardized, model-independent benchmark for empirically assessing LLM performance on hypothesis testing as a cohesive task.

A. Gap & Contribution

Despite rapid progress in evaluating LLMs for logical reasoning, factual verification, and isolated statistical tasks, there is currently no standardized benchmark dedicated to evaluating hypothesis testing as a unified end-to-end capability of large language models on statistical t-tests problems. Existing reasoning benchmarks emphasize formal logic, while statistical studies primarily assess fragmented subtasks such as distribution estimation or numeric reasoning. Furthermore, automated hypothesis testing frameworks remain largely conceptual and lack large-scale empirical validation on modern LLMs.

This work fills that gap by proposing a dedicated benchmark to assess LLMs' hypothesis-testing abilities as an integrated reasoning and computational task. We formally define hypothesis testing as an evaluable capability, construct a benchmark that covers structured claim formulation, statistical decision-making, and validity assessment, and perform an extensive evaluation of current LLMs. Our findings offers a systematic characterization of LLM strengths and weaknesses in statistical hypothesis testing, laying the groundwork for future research in AI-enabled scientific verification and discovery.

III. Foundational Background

A. Hypothesis Testing

Statistical hypothesis testing is a method of statistical inference used to decide whether the data at hand sufficiently support a particular hypothesis. It involves:

- **Null Hypothesis** (H_0): The default position that there is no relationship between two measured phenomena.
- **Alternative Hypothesis** (H_1): The position that there is a relationship.
- **Test Statistic**: A standardized value calculated from sample data during a hypothesis test.
- **P-value**: The probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.

A decision is made to reject or fail to reject H_0 based on a significance level (α), typically 0.05.

B. Large Language Models & Prompting

LLMs are transformer-based models trained on vast corpora of text. Their performance on complex tasks can be significantly improved through prompting techniques:

- **Zero-Shot**: Asking the model to perform the task without examples.
- **Few-Shot**: Providing examples of the task within the prompt.
- **Chain-of-Thought (CoT)**: Encouraging the model to generate intermediate reasoning steps [3].
- **Program-of-Thought (PoT)**: Asking the model to generate and execute code to solve the problem, decoupling reasoning from computation.

IV. Benchmark Design and Architecture

A. System Overview

The benchmark is implemented as a modular, asynchronous Python pipeline that evaluates large language models (LLMs) on end-to-end hypothesis testing. The main controller is the `HypothesisTestingBenchmark` class in `ht.py`, which handles five main subsystems:

- synthetic data generation
- prompt construction
- multi-provider LLM clients
- response parsing and statistical evaluation, and
- aggregation and visualization.

Each evaluation generates a structured JSON entry containing the scenario, prompt, model output, parsed components, ground-truth statistics, and computed metrics. These JSON records are subsequently tranfered to the Streamlit dashboard.

B. Core Components

Data generation and ground truth
Synthetic hypothesis-testing scenarios are created by the `DataGenerator` in `data_generator.py`. It supports one-sample and two-sample t-tests, paired t-tests, via type-specific generators. Scenarios are parameterized by sample size, distributional assumptions, and sample sizes. The `StatisticalEngine` computes the ground-truth test statistic, p-value, and decision for each scenario so that all models are compared against the same reference implementation.

Prompt engine
Prompt construction is handled by `prompts.py`, which defines a family of templates derived from a common `PromptTemplate` base class. The benchmark currently supports zero-shot, few-shot, Chain-of-Thought (CoT),

Program-of-Thought (PoT), and structured-output prompting. Each template takes the numeric scenario (samples, groups) and a test-specific natural language context from `create_test_context`, and renders a compact, model-facing input string. The PoT template emits executable Python using `numpy` and `scipy.stats`, and requires the model to output a single summary line of the form `RESULT: test_type=...; statistic=...; p_value=...; decision=...`, which is later parsed deterministically. For structured-output runs, `RESPONSE_SCHEMA` defines the desired JSON shape, including hypotheses, test method, statistics, decision, and conclusion.

LLM client abstraction
All provider-specific details are encapsulated in subclasses of the abstract `LLMClient` interface (`llm_clients.py`). Concrete implementations include `OpenAIClient`, `AnthropicClient`, `GoogleClient` (Gemini), and `GrokClient`, each normalizing differences in authentication, endpoints, and parameter conventions. The orchestrator in `ht.py` instantiates these clients from a simple configuration mapping of providers to model names, so that adding a new model typically requires only updating the configuration, not the pipeline logic.

Response parsing and evaluation
Raw LLM outputs are converted into a structured representation by `ResponseParser`. The parser first looks for the PoT `RESULT` line and, if present, populates a `ParsedResponse` Pydantic model with the test method, statistic, p-value, and normalized decision. If no such line exists, it falls back to a combination of JSON extraction (for structured modes) and regex-based extraction of hypotheses, test type, test statistic, p-value, degrees of freedom, decision, confidence interval, assumptions, and conclusion from free-form text. Validators on the Pydantic model enforce basic sanity checks (e.g., $p \in [0, 1]$) and normalize decision strings to `reject_H0` versus `fail_to_reject_H0`. The `EvaluationMetrics` module then compares the parsed response against the ground truth and computes accuracy at several levels (test-type, p-value, decision, reasoning quality, hallucination flags, latency).

Aggregation, storage, and visualization
Each evaluation call returns a JSON-serializable record that includes a timestamp, model name, prompt type, test type, prompt text, raw response, parsed result, ground truth, evaluation metrics, and latency. The orchestrator in `ht.py` accumulates these records in-memory and periodically writes them to timestamped JSON files under `config.RESULTS_DIR`. The `BenchmarkAggregator` in `evaluator.py` computes summary statistics by model, prompt type, and test type, as well as comparison matrices used in the report. The interactive dashboard in `dashboard/app.py` loads all JSON results, flattens them into a Pandas DataFrame, and exposes a leaderboard-style UI (built with Streamlit and Plotly) that presents per-model accuracy, confidence intervals, error breakdowns, and qualitative examples through dedicated analysis tabs.

C. Execution Flow

A benchmark run is configured and launched from `ht.py`, either in quick, full, or custom mode. For a given configuration, the orchestrator:

1) Samples a grid of *test types* and *sample sizes* and calls `DataGenerator` to create synthetic scenarios with associated metadata.
2) For each provider/model and each selected *prompt type*, generates a natural-language prompt (or PoT program / structured-output instruction) using `get_prompt` from `prompts.py`.
3) Dispatches all (model, prompt, scenario) triples asynchronously via the appropriate `LLMClient`, respecting a global concurrency limit through an `asyncio.Semaphore`.
4) Parses each returned response with `ResponseParser`, computes ground truth with `StatisticalEngine`, and evaluates correctness and auxiliary metrics with `EvaluationMetrics`.
5) Appends the resulting evaluation record to the in-memory result list and, at the end of the run, writes the full set of records to disk and prints a textual summary using `BenchmarkAggregator`.

This design cleanly separates concerns: data generation and ground truth are independent of model and provider; prompting strategies are independent to the LLM backends; and parsing/evaluation are shared across all models and modes.

D. Extensibility and Reproducibility

The modular architecture is explicitly designed for extensibility. New statistical tests can be added by implementing a generator method in `DataGenerator`, extending `create_test_context`, and adding the test to the orchestrator's list of supported types. New prompting strategies are added by defining a new `PromptTemplate` subclass and registering it in `get_prompt`. Additional models or providers are integrated by subclassing `LLMClient` and extending the configuration mapping in `config.py`. Because all runs are parameterized through configuration files, fixed seeds in `DataGenerator`, and timestamped JSON outputs, the benchmark is reproducible and auditable: any reported result can be traced back to the exact synthetic scenario, prompt, raw response, and evaluation logic used.

E. Evaluation Metrics

For each tuple the benchmark logs a structured evaluation record. Core scalar metrics are:

- **Test-method accuracy**: Indicator of whether the predicted test family (e.g., one-sample t-test, paired t-test) matches the ground truth.
- **Decision accuracy**: Indicator of whether the model's reject / fail-to-reject decision at $\alpha = 0.05$ agrees with the ground-truth decision.
- **P-value accuracy**: Indicator for whether the reported p-value lies within a fixed tolerance band of the true p-value and implies the same decision at α.

- **Overall accuracy**: Mean of the three components above plus a consistency check between the reported statistic, p-value, and decision.
- **Reasoning quality**: A rubric-based score in $[0, 1]$ capturing whether the response states hypotheses, identifies the test, explains the computation, and interprets the result correctly.
- **Hallucination flag**: Binary indicator of materially unsupported content (e.g., wrong test name for the data, self-contradictory decision, impossible numerical values).
- **Latency**: End-to-end model call duration in seconds, measured with `time.perf_counter`.

These per-sample metrics are aggregated by model, prompt strategy, and test family (mean, standard error, confidence intervals) and drive the leaderboard, radar plots, heatmaps, and error analyses reported in Section **??**.

V. Evaluation & Results

Results aggregate across the three t-test families currently enabled (one-sample, two-sample, and paired t-tests), two sample-size regimes ($n = 20$ and $n = 50$), and the four prompt strategies (zero-shot, few-shot, CoT, PoT).

A. Leaderboard: Overall Accuracy and Reliability

The primary leaderboard view ranks models by mean overall accuracy with 95% confidence intervals and exposes decision accuracy, hallucination rates, reasoning scores, and latency. Table I summarizes the aggregated results from the most recent full run used to populate the dashboard.

TABLE I
Model Performance Comparison (Dashboard Summary)

Model	**Overall Acc.**	**Decision Acc.**	**Hal. Rate**
grok-4-fast	81.2%	75.0%	25.0%
grok-3	80.8%	87.7%	30.9%
claude-opus-4-5	79.3%	89.5%	18.0%
grok-3-mini	78.1%	95.7%	24.6%
claude-sonnet-4-5	76.3%	84.0%	35.1%
claude-opus-4-1	75.4%	89.7%	27.7%
claude-haiku-4-5	74.2%	88.0%	25.0%
gpt-4o	73.6%	89.7%	40.8%
grok-4-1-fast-r.	72.1%	81.9%	22.3%
gpt-5.1	68.5%	83.6%	49.1%
deepseek-c.	65.2%	86.2%	15.8%
gpt-4o-mini	53.1%	87.5%	0.0%
gpt-4	51.4%	64.1%	12.5%
claude-3-haiku	50.0%	0.0%	100.0%
gpt-5-mini	42.3%	49.0%	13.5%
gemini-2.5-p.	36.8%	31.8%	9.7%
gemini-2.5-f.	24.1%	20.3%	13.3%

The leaderboard and its confidence intervals make three patterns visible. First, the best Grok and Claude variants form the top tier, with mean overall accuracy around 80% and high decision accuracy, indicating that they almost always choose the correct reject/fail-to-reject action at $\alpha = 0.05$. GPT-4o sits slightly below this tier in overall accuracy but with competitive decision accuracy, suggesting that most of its errors come from secondary aspects which include test naming or p-value calibration rather than incorrect final decisions. Second, mid-tier models such as GPT-5.1 and DeepSeek Chat achieve respectable decision accuracy but noticeably lower overall accuracy and, in the case of GPT-5.1, a comparatively high hallucination rate according to the hallucination indicator. Third, Gemini 2.5 models and smaller variants (GPT-5-mini, older Haiku) trail the group with substantially lower overall accuracy and decision accuracy, reflecting difficulty in either selecting the right test or maintaining consistent numerical reasoning across scenarios.

B. Family-Level Capability Profiles

The radar panels group models into four families: GPT models, Grok models, Claude models, and a combined other family. Each radar traces five dimensions namely: test-method accuracy, decision accuracy, p-value accuracy, reasoning quality, and completeness normalized to $[0, 1]$.

Within the GPT family radar, GPT-4o provides the strongest and most balanced profile, with high decision accuracy and reasoning quality but a noticeable dip in hallucination control compared to some Claude and Grok variants (consistent with Table I). GPT-5.1 and GPT-4o-mini achieve strong decision accuracy but exhibit more uneven coverage across dimensions, especially in p-value accuracy and completeness when prompts are less structured.

The Grok radar shows a consistently high and rounded footprint: both grok-4-fast and grok-3 score strongly on test-method selection, p-value accuracy, and decision correctness, giving them the broadest capability coverage of all families. The Claude radar is similarly strong but slightly more skewed toward reasoning quality and explanation completeness, reflecting that Claude often produces well-structured, interpretable arguments for its conclusions, even when slightly conservative on border-line decisions. In contrast, the Gemini/DeepSeek/other radar appears more irregular: DeepSeek Chat exhibits a reasonably solid decision and reasoning profile, while Gemini 2.5 variants display thinner radii, especially on test-method and p-value axes, indicating less stable statistical reasoning across the tested tasks.

C. Performance by Test Type and Prompt Strategy

The heatmap view breaks overall accuracy down by test type and model. Across families, performance on one-sample and two-sample t-tests is generally stronger than on paired t-tests, where several models mis-handle the dependency structure or confuse the scenario with an independent two-sample design. Top Grok and Claude models maintain relatively uniform accuracy across all three test types, whereas lower-tier models show large drops on paired tests, reflecting weaker understanding of experimental design.

The ”Prompt Strategy Impact” bar chart quantifies the effect of Zero-Shot, Few-Shot, CoT, and PoT prompting on overall accuracy. For all high-performing models, PoT consistently appears as the top-performing strategy on the dashboard, often adding a sizable margin over Zero-Shot and Few-Shot prompts. By delegating computation to Python and forcing a single structured summary line, PoT largely eliminates arithmetic errors and format drift. CoT improves over Zero-Shot by encouraging explicit identification of the test and intermediate reasoning steps, but still occasionally suffers from manual numerical mistakes and verbose formatting that stress the parser. Few-Shot prompts offer modest gains relative to Zero-Shot for models that already follow instructions reliably, but the improvement is small compared to the jump from natural-language reasoning to PoT.

D. Error Patterns, Calibration, and Latency

The statistical deep-dive tab combines a scatter plot of true versus predicted p-values with a mean absolute error (MAE) bar chart. For the top-tier Grok, Claude, and GPT-4o models, the scatter cloud concentrates near the diagonal $y = x$ line, indicating good p-value calibration and relatively small MAE bars. In contrast, Gemini variants and some smaller models show a more diffuse scatter with noticeable off-diagonal clusters, especially for mid-range p-values, which correspond to the most ambiguous cases for hypothesis testing. Decision errors in the scatter plots are concentrated around scenarios where the true p-value lies close to the significance threshold; here, models sometimes over-reject or over-conserve, depending on how they interpret borderline evidence.

The reasoning-quality box plots reveal that Grok, Claude, and GPT-4o maintain high median reasoning scores with tight interquartile ranges, indicating reliable coverage of key reasoning steps. Mid-tier and lower-tier models show both lower medians and heavier tails of low-scoring responses, consistent with occasional missing assumptions or incomplete conclusions. Finally, the latency bar chart shows that all models used in this study remain within a practically usable response time, with smaller models and some Gemini variants responding faster on average but at the cost of accuracy, and Grok/Claude/GPT-4o trading slightly higher latency for markedly better statistical performance. With Grok 4.1 fast reasoning having the highest latency.

E. Qualitative Behaviour

The qualitative inspector tab allows side-by-side inspection of full prompts, raw responses, and ground-truth statistics. These examples corroborate the quantitative error analysis: top-performing models typically select the correct test, compute a numerically reasonable statistic and p-value, and articulate a concise conclusion. Most hallucinations flagged by the dashboard are not wild fabrications but rather over-confident test-name claims, extra invented context, or inconsistent restatements of the same result. Lower-performing models occasionally mis-identify the test (e.g., treating a paired design as independent) or conflate descriptive and inferential claims, leading to incorrect decisions even when the numerical computation is approximately right. Overall, the dashboard confirms that modern frontier models are capable of robust, end-to-end hypothesis testing when supported by appropriate prompting (especially Program-of-Thought), but also highlights systematic weaknesses in test selection, calibration, and explanation that must be addressed before such systems can be trusted in high-stakes statistical settings.

VI. Conclusion & Future Work

This work establishes a rigorous benchmark for evaluating hypothesis testing in LLMs. We find that while top-tier models such as Grok-4, Claude Opus, and GPT-4o are approaching reliable performance on the tested t-test families, they still benefit significantly from structured prompting in particular Program-of-Thought to guarantee numerical precision.

Future work will focus on:

- Expanding the benchmark to include additional tests (ANOVA, regression, chi-square, non-parametric tests) and more varied effect-size and distributional regimes.
- Investigating agentic workflows where the model can iteratively request more data, inspect diagnostics, or run pilot simulations before committing to a decision.
- Refining hallucination and reasoning-quality metrics to better distinguish between harmless verbosity, genuine logical errors, and materially misleading claims.

Ultimately, this research paves the way for AI assistants that can participate more reliably in the scientific method, moving beyond text generation to evidence-based statistical reasoning and decision support.

References

[1] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac *et al.*, “Scientific discovery in the age of artificial intelligence,” *Nature*, vol. 620, no. 7972, pp. 47–60, 2023.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” OpenAI, Tech. Rep., 2023, arXiv preprint arXiv:2303.08774.

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[4] G. Parmar *et al.*, “Logicbench: Towards systematic evaluation of logical reasoning ability of large language models,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[5] Y. Wan *et al.*, “Logicasker: Evaluating and improving the logical reasoning ability of large language models,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[6] S. Zhang *et al.*, “Multi-logieval: A unified benchmark for multi-step logical reasoning in large language models,” *arXiv preprint*, 2024.

[7] A. Survey, “Logical reasoning in large language models: A survey,” *arXiv preprint*, 2025.

[8] W. Hong *et al.*, “A closer look at the self-verification abilities of large language models in logical reasoning,” in *Proceedings of NAACL 2024*, 2024.

[9] T. Zhu *et al.*, “Factreasoner: A probabilistic framework for long-form factuality assessment of large language models,” *arXiv preprint*, 2025.

[10] F. Jiao, W. Zhang, and X. Li, “Trustworthy reasoning: Evaluating and enhancing factual accuracy in llm intermediate thought processes,” *arXiv preprint*, 2025.

[11] X. Liu, Z. Wu, X. Wu, P. Lu, K.-W. Chang, and Y. Feng, “Are llms capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data,” *arXiv preprint arXiv:2402.17644*, 2024.

[12] Y. Zhu, S. Du, B. Li, Y. Luo, and N. Tang, “Are large language models good statisticians?” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, pp. 62 697–62 731, 2024.

[13] H. Tiwari, “A framework for automated hypothesis testing,” *Preprint / Technical Report*, 2025.