LLM Evaluation Study

Methodology and findings from 30 evaluations across 9 production models

30 evaluations · 20,110 tests · 9 models · 3 providers

By David Hood, 42 Robots AI

June 2026 · Version 1.0

Version 1.0 · Published June 2026

Contents

Executive Summary

Three assumptions drive most current decisions about applying LLMs in production: that evaluating LLM outputs requires an LLM judge, that pro-tier models earn their premium, and that a single frontier model can dominate the field. We tested all three across 30 evaluations and more than 20,000 individual tests in Q1 2026. The data falsified all three.

Assumption 1 — Evaluating LLM outputs requires an LLM judge.

Most automated eval frameworks (OpenAI Evals, LangSmith, Braintrust, Arize, Weave) default to LLM-as-judge as their primary scoring path: pass the output to a stronger LLM with a rubric, get a 1–5 score back. It's flexible, easy to set up, and the path of least resistance in framework documentation. It's also expensive, slow, stochastic, and prone to documented biases (self-preference, position, verbosity).

Of the 30 evaluations in this study, exactly one used LLM-as-judge as the primary scoring method. The other twenty-nine used deterministic or near-deterministic methods: structural validation (does the output parse? does it have the right shape?), golden-set comparison (does it match a pre-labeled correct answer?), embedding similarity (does it preserve meaning?), fact and keyword coverage (do the required facts appear?), and behavioral checks (did the model produce output? how fast?).

This is not a claim that LLM-as-judge is broken or useless — it has a real place when the dimension being evaluated is irreducibly subjective (instruction-spirit adherence, tone, conversational naturalness). It is a claim that **the cheaper and more reproducible methods give the necessary meaning to usefully evaluate LLM operations**. The 30:1 ratio in this study demonstrates that an eval suite can be effectively built bottom-up from deterministic methods, with LLM-as-judge reserved for the narrow set of dimensions nothing else can score — not used reflexively because it's the framework default.

Assumption 2 — Pro-tier models earn their premium.

Across 30 evals, lite-tier and default-tier models won the majority of tasks outright. The single most striking case in the study: on 4,140 tests of action extraction, Gemini Flash Lite — the cheapest model in the matrix — tied for first place against its own pro sibling and beat Claude Opus, GPT-5.4, and Claude Sonnet. The same lite-tier model also placed first on entity extraction across 675 tests.

Within-provider tier inversions occurred on multiple evals. Gemini Pro scored 16 percentage points worse than Gemini Flash Lite on relevance checking. Claude Haiku beat Claude Opus by 33% on knowledge-graph utilization. GPT-5-nano averaged longer response times than three higher-tier models on raw latency.

Tier labels are pricing labels, not capability labels. Defaulting to pro tier is statistically wrong on most production workloads — and expensive on top of being wrong.

Assumption 3 — A single frontier model can dominate the field.

Five different models won top rank on different evals in our suite. No model placed in the top three on every eval. The same Gemini Flash Lite that won extraction tasks finished dead last on content generation. The same GPT-5-mini that tied for first on conversational memory finished dead last on web-grounded queries.

Each of the three providers has measurably distinct strengths: Anthropic models excel at web-grounded reasoning and knowledge-graph context, Google models at structured extraction and speed, OpenAI models at fact-coverage-heavy generation and conversational continuity. **Multi-provider routing — using different models for different tasks — is increasingly the rational architecture for production AI systems. Single-vendor commitment ties an application to that vendor's strengths and its weaknesses.**

What we ran

30 evaluations across 9 production models (3 providers × 3 quality tiers), more than 20,000 individual tests, ~20 prompt variants. Each evaluation is a composite test of one capability scored on multiple dimensions — Note Generation, for example, scores six dimensions (fact preservation, entity coverage, Q&A consistency, organization, hallucination check, brevity) into one weighted total. Across the 30 evaluations we used **approximately 89 distinct scored dimensions**, ranging from one focused metric (latency, classification accuracy) to six layered sub-metrics. The "30 evals" headline is a count of eval processes, not of tasks.

The methodology categorizes scoring approaches into a lexicon: six categories applied in this study (Structural, Grounded, Semantic, Coverage, Behavioral, Judged), five additional categories named but not applied (Adversarial, Human Evaluation, Robustness, Safety/Toxicity, Code Execution), and seven cross-cutting techniques that combine categories (LLM-as-Q&A Probe, Multi-Step Composite Scoring, A/B Delta Scoring, Layered Pyramid Scoring, Threshold Sweep, Unique-Anchor Verification, N-Shot Stability Probe). Definitions, worked examples, and the per-eval inventory are in the body of the paper.

What this means

Production AI in 2026 cannot be designed responsibly without an in-house evaluation suite. Vendor benchmarks (MMLU, GPQA, HumanEval) measure capabilities that do not reliably transfer to specific production tasks like extraction, classification, retrieval, and structured output. The only way to know which model wins on a given task is to run that task. The data in this study makes that case empirically, across thirty tasks and eighty-nine scored dimensions.

The recommendation: treat eval infrastructure as foundational, on the same tier as logging, monitoring, and continuous integration. The cost of building it is real but bounded. The cost of *not* building it — paying premium prices for lite-tier work, missing systematic provider failure modes, locking into the

wrong model, defaulting to whatever the framework's quickstart suggested — compounds in production indefinitely.

1. A Lexicon for LLM Evaluation

1.1 Why a lexicon

Talking about LLM evaluation has a vocabulary problem. Teams use the same word ("eval") for at least six distinct kinds of work, and confuse the techniques used inside an eval with the categories the eval falls into. The result: discussions that sound like they're about methodology but are really about implementation choices, and decisions about evaluation infrastructure made without a clear picture of what's being chosen between.

The lexicon below is drawn from how the 30 evaluations in this study actually scored their outputs. It is not exhaustive of the field, but it covers what we used and gives names to several patterns that recur across many of our evals. Six categories of scoring methodology, five additional categories the field uses that we did not apply here, and seven cross-cutting techniques that combine categories.

1.2 The six categories applied in this study

Each category answers a different question about the model's output. They are not mutually exclusive — most of our evals layer two to four of them.

Structural validation

The question: Does the output have the right shape?

Pure form and schema checks. No AI in the scorer. Includes JSON parseability, regex pattern presence, count validation (did the model extract the right *number* of items?), type or enum membership, format rules (lowercase, max-length, no spaces), and algorithmic assertions like topological-order checks.

Strengths: Perfectly deterministic, near-zero cost, instant feedback. Catches the most common failure modes — malformed output, wrong shape, hallucinated structure.

Limitations: Says nothing about whether the *content* is correct. A model can return perfectly formatted garbage.

Use it: As the first layer for any task with a defined output schema. If a downstream system depends on parsing the output, structural validation comes before anything else.

Grounded comparison

The question: Does the output match a pre-labeled correct answer?

Compare model output against a known-correct reference using exact, set-based, or fuzzy matching. The reference is determined before the eval runs. Includes boolean classification accuracy, binary detection (yes/no should-extract), set-based F1 with precision and recall, and categorical exact match.

Strengths: Deterministic once labels exist. Precise — measures actual correctness, not vague quality. Enables controlled prompt engineering: change one variable, measure the delta exactly. The upfront labeling work amortizes across unlimited eval runs.

Limitations: Requires labeling investment. Some tasks genuinely don't have a single right answer (open-ended generation). Ground truth can become stale if the task evolves.

Use it: Whenever you can define correctness before running the eval. Classification, extraction, binary decisions — most production LLM tasks have ground truth even if teams don't invest in creating it.

Embedding-based semantic similarity

The question: Does the output preserve meaning?

Use embedding models — which are deterministic for a given input — to measure semantic distance between texts. Cosine similarity in vector space. Not asking an LLM to opine. Includes compression-quality measurement (embed answers from original vs compressed context), cross-provider consistency (embed responses from different models to the same query), and embedding-similarity threshold matching for pattern detection.

Strengths: Deterministic (same text → same vector). Roughly 100× cheaper per call than LLM-as-judge generation. Composable into multi-step evaluation pipelines.

Limitations: Measures similarity, not correctness — two confidently wrong answers can be highly similar to each other. Embedding quality caps the eval quality.

Use it: When meaning preservation matters more than form. Compression, paraphrase tolerance, cross-model agreement, retrieval pattern matching.

Coverage verification

The question: Are the required facts in the output?

Define facts, entities, or keywords that *must* appear in a correct response. Check for their presence via substring matching or fuzzy text matching. A hybrid of grounded comparison and structural validation.

The strongest variant uses **unique-anchor verification**: plant fictional facts in the source data — names like "Joe's Plumbing", specific phone numbers, statistics that can't exist in any model's training set — and substring-match the response. False positive rate is essentially zero. If the anchor appears in the response, the model definitely retrieved the source data.

Strengths: Deterministic substring matching. Very precise for retrieval-augmented generation — directly measures whether retrieved information was incorporated. Catches hallucination by omission.

Limitations: Inclusion is not accuracy. A model can include the keyword in the wrong context. Doesn't measure response quality beyond fact presence.

Use it: Any RAG or generation task where you can pre-identify what should appear in the output. Especially powerful with unique anchors that can't be guessed.

Behavioral / functional testing

The question: Did the system *work*?

Not whether the output looks right — whether the system functioned at all and how. Includes success rate (did the model produce *anything*?), zero-output detection, latency profiling, end-to-end pipeline pass/fail, and convergence behavior (does recursive summarization terminate? in how many rounds?).

Note: Behavioral overlaps with Structural at the empty-output edge case but extends beyond it: latency, convergence, and end-to-end pipeline success are properties of producing output, not of the output text — Structural requires content to validate, Behavioral does not.

Strengths: Trivially simple to implement. Catches catastrophic failures that quality evals can't catch. Directly measures production-relevant behavior. Zero cost beyond the LLM call itself.

Limitations: Tells you nothing about quality. A model can be fast, always produce output, and still be wrong.

Use it: Always. This should be the baseline eval layer before any quality scoring. If the model doesn't produce output, or produces it too slowly, quality is irrelevant.

LLM-as-judge

The question: Is the output good, by a subjective rubric?

Use a (usually stronger) LLM to evaluate the output of another LLM, typically scoring on a 1–5 scale against criteria. Includes instruction-adherence scoring, conversation-quality grading, image-description quality assessment.

Strengths: Can evaluate subjective dimensions that other methods can't reach. Flexible — the judging prompt can score arbitrary quality criteria. Easy to set up.

Limitations: Stochastic — same input, different score on different runs. Expensive — every eval call costs as much as the original generation. Biased — documented self-preference bias (judges prefer their own provider's output), position bias, verbosity bias. Circular — the judge has its own blind spots, and the eval inherits them.

Use it: Only when the dimension being evaluated is genuinely subjective — instruction "spirit" adherence, tone, conversational naturalness. If you can define correctness objectively (format, facts, similarity), use a cheaper and more deterministic method. Your results will be more meaningful and more actionable.

In this study, exactly one of 30 evaluations used LLM-as-judge as the primary scoring method.

1.3 Where Q&A fits — a technique, not a category

A common pattern in LLM evaluation is to use a model itself to ask questions about source material and to answer questions about output, then compare the answers. This is sometimes called "LLM-as-Q&A" or "questioning-based evaluation," and it deserves a name.

But it isn't a separate category. It's a technique that uses two of the categories above:

- Q&A as **grounded comparison**: Generate questions about the source. Get a reference answer from the source. Get an answer from the candidate output. Compare via semantic similarity (Category 3) or substring match (Category 4).
- Q&A as **coverage probe**: Ask whether the response can answer specific factual questions ("what was the Q3 revenue?"). Used in summarization to detect lost specifics.

The category framework holds; Q&A is a technique that operationalizes grounded and semantic evaluation through a question-asking interface. We call out the distinction because conflating it with the categories causes confusion in design discussions.

1.4 Worked example: layered scoring on Action Extraction

The Action Extraction eval (Eval 1.7 in this study, 4,140 individual tests) is a good example of how categories layer together inside a single eval.

Input: a natural-language user message like *"Create a new project called Website Redesign and add a task called Update homepage"*.

The model's output is expected to be structured JSON listing the actions to take.

The eval scores the output in four sequential layers:

Layer	Category	Question
1	Structural	Does the output parse as JSON?
2	Structural	Does it have the right action *count*?

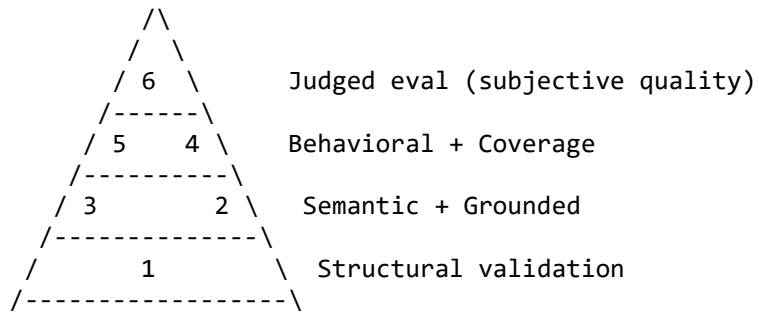
3	Grounded	Do the action *types* match the expected vocabulary?
4	Grounded	Do the action *names* match the labeled targets?

A model that passes Layer 1 but fails Layer 2 is over-extracting (producing extra invented actions). A model that passes Layers 1–2 but fails Layer 3 has the right *number* of actions but the wrong *kinds*. A model that passes Layers 1–3 but fails Layer 4 has the right structure but extracted the wrong specifics.

Each layer reveals a different failure mode. A single quality score (LLM-as-judge: 3.2 / 5) tells you almost nothing about what to fix. The four-layer score tells you exactly where the model struggles and what to change in the prompt.

1.5 The eval pyramid

The six categories form a pyramid, sorted by cost and reproducibility:



The bottom layers run first, are cheapest, and catch the most common failures. The top layer is reserved for the dimensions nothing else can score.

The most common mistake in eval design is starting at Layer 6 because it's general-purpose. Most production AI tasks have correctness criteria that can be defined more cheaply at Layers 1–5.

1.6 Five additional categories the field uses that we did not apply in this study

The six categories above cover what we used because they gave us the meaning we sought when measuring internal processes for our software. The field has at least five more that any complete eval program should consider.

Category	What it is	Why it matters
Adversarial / Red-Team	Deliberately try to break the model — prompt injections, jailbreaks, hostile inputs, edge cases designed to elicit failure.	Different goal from quality evaluation: this asks "what does it take to *make* the model fail?" Important for systems facing untrusted user input.
Human Evaluation	Domain experts score outputs by rubric or rank pairwise. Often the calibration ground truth that automated evals are validated against.	Slow and expensive, but irreplaceable for genuinely subjective quality and for calibrating LLM-as-judge against human standards.
Robustness / Perturbation	Same model, *perturbed* input — typos, paraphrasing, capitalization changes. Does the output stay stable?	Distinct from cross-provider consistency, which holds the input fixed. This holds the model fixed and varies the input. Catches brittle prompts.
Safety / Toxicity / Bias	Specialized classifiers detecting harmful, biased, or NSFW outputs. Often regulated separately from quality.	Required for any user-facing application, especially in regulated industries. Different scoring tools, different remediation paths.
Code Execution / Runtime	The output is an executable artifact (code, SQL, HTML, CSS, JS, regex). Don't grade the *text* — *run it* and check the result.	Catches errors that text-based evaluation can't see. For website builders specifically: validate HTML structure, CSS parseability, JS syntax, and verify generated code actually executes.

These are not the only additional categories — the field uses more — but they are the five we believe most teams should consider including in their eval programs alongside the six we applied.

1.7 Cross-cutting techniques (combining categories)

Categories answer "what are we measuring?" Techniques answer "how do we operationalize that measurement?" The same patterns recur across many evals; recognizing them as named techniques makes designing new evals faster.

Technique	Categories it combines	Pattern	Canonical example
LLM-as-Q&A Probe	Grounded + Semantic	Generate questions, answer from reference and from output, compare answers.	Note Generation Q&A consistency sub-metric
Multi-Step Composite Scoring	4+ categories	Layer multiple sub-metrics into one weighted total. Reveals *where* a model fails, not just *that* it fails.	Note Generation 6-metric scorer (fact 30%, entity 20%, Q&A 20%, organization 10%, no-external 15%, no-commentary 5%)
A/B Delta Scoring	Any category, applied twice	Run the same eval with two configurations; report the delta. Isolates the causal contribution of one variable.	Web Grounding — search-on minus search-off coverage, per query category
Layered / Pyramid Scoring	Cat 1 → 2 → 3 in sequence	Run cheap layers first; only run expensive layers if cheap layers pass. Each layer reveals a different failure mode.	Action Extraction 4-layer scoring (JSON valid → count → types → pass)
Threshold Sweep	Semantic, typically	Don't pick a threshold a priori; sweep across multiple values, pick optimal data-driven.	Semantic pattern-match routing — cosine threshold swept 0.60 to 0.80
Unique-Anchor Verification	Specialized Coverage	Plant fictional facts in source data (names, numbers, statistics that	External Source Capture verification

		can't exist in training data). Substring-match output for them. False-positive rate is near-zero.	setup
N-Shot Stability Probe	Any category, repeated	Same input × N runs. Measure variance. High variance = unpredictable production behavior.	Temperature consistency probes for non-zero temp models

Most production-grade evals use two to four of these techniques together. Designing a new eval is largely the work of picking the right combination of categories and techniques for the task.

---

2. What We Ran

2.1 The model matrix

Nine production models were tested across three providers and three quality tiers. The matrix is the standard way to compare price-tier behavior across the major providers.

Tier	OpenAI	Google	Anthropic
Lite	gpt-5-nano	gemini-3.1-flash-lite-preview	claude-haiku-4-5
Default	gpt-5-mini	gemini-3-flash-preview	claude-sonnet-4-6
Pro	gpt-5.4	gemini-3.1-pro-preview	claude-opus-4-6

Where prompt variations were tested (most evals had 3–5 prompt variants), each variant ran across all 9 models. Combined with sample counts of 10–30 per condition, individual evals ranged from 30 tests at the smallest to 4,140 at the largest.

2.2 The 30 evaluations

The 30 evaluations are organized into five functional groupings, listed here with test counts. Per-eval methodology and headline results appear in §3 and the appendix.

Part	Theme	Evals	Tests
1	Core pipeline (extraction, classification, summarization, speed)	10	11,808
2	Agent and project system (task classification, replan, conditional logic, end-to-end)	7	5,717
3	Core capabilities (compression, entity extraction, knowledge graph, retrieval, files)	7	1,836
4	Project-system quality (instruction injection, tree consistency)	2	285
5	Gap evals (thread continuation, web grounding, provider consistency)	4	464
Total		**30**	**20,110**

The 20,110 figure is the sum of individual model × prompt × sample combinations. Quality scoring within each eval typically used multiple sub-metrics combined into a weighted total — averaging three sub-metrics per eval, with a range of one to six. Across the 30 evals, approximately 89 distinct quality dimensions were scored. The "30 evals" headline is a count of eval *processes*, not of tasks.

2.3 Methodology distribution across the 30 evals

Two views are useful for understanding which methodologies do the work in this suite. The first counts every category that appears anywhere in an eval's scoring stack. The second counts only the *primary* scorer — the method that produces the headline number.

View 1 — any usage:

Category	# of evals	Share of suite
Structural	11	37%
Grounded	11	37%
Coverage	10	33%
Behavioral	10	33%
Semantic	8	27%
Judged	2	7%

View 2 — primary scorer:

Category	# of evals	Share of suite
Grounded	8	27%
Coverage	7	23%
Semantic	6	20%
Structural	4	13%
Behavioral	3	10%
Hybrid	1	3%
Judged	1	3%

The primary-scorer view is the more striking distribution. **Judged — the method most teams default to — drives the headline number on exactly one of 30 evals.** Structural is also a small share at the primary level — it does enormous work as a *gating* layer (catches malformed output before quality scoring), but rarely produces the headline metric because "passes JSON" doesn't tell you if the content is right.

The deterministic methods — Structural, Grounded, Coverage, Behavioral — collectively drive 73% of the primary scoring in this suite. That ratio is the empirical heart of the methodological argument: cheap, reproducible methods carry most of the weight.

---

3. What We Found

3.1 Model performance summary

Five different models won top rank on different evals. The full per-eval winners are in the appendix. Two patterns stand out:

Lite-tier wins are common. Gemini Flash Lite placed first on action extraction (97.6% across 4,140 tests, tied with Gemini Pro), entity extraction (F1 0.811 across 675 tests), and conditional logic (97.8%). Claude Haiku won knowledge-graph utilization (4.80 / 5, beating Claude Opus by 33%).

Pro-tier wins are narrow. Claude Opus and Gemini Pro finished mid-pack on most evals. GPT-5.4 won content generation fact coverage (0.852, the only clean pro-tier win in the suite), and tied for #1 on action extraction. The pro tier earned its premium on creative content generation and certain complex-reasoning tasks. Outside that narrow set, lite and default models were competitive or winning.

Provider profiles are distinct. Each provider has a measurable specialty:

- **Anthropic**: web-grounded reasoning (Sonnet #1 with the largest +9.3% search boost), knowledge-graph utilization (Haiku #1, Sonnet #2), thread continuation (Sonnet tied #1).
- **Google**: structured extraction (Flash Lite won action and entity extraction), speed (Flash Lite averaged 1,937ms — 2× faster than the next-fastest model).
- **OpenAI**: fact-coverage-heavy generation (GPT-5.4 #1 on content generation), conversational continuity (GPT-5-mini won confirmation response, tied #1 on thread continuation).

The full headline ranking by eval is in Appendix A.

3.2 Counter-intuitive results — the lite-beats-pro pattern

Several results in this suite directly violate the assumption that price tier predicts quality. The four most striking:

Action Extraction (Eval 1.7, 4,140 tests, Structural scoring):

Rank	Model	Pass Rate	Tier

1 (tie)	gemini-3.1-flash-lite-preview	97.6%	Lite
1 (tie)	gemini-3.1-pro-preview	97.6%	Pro
4	claude-opus-4-6	95.7%	Pro
5	gpt-5.4	95.4%	Pro
9	gpt-5-nano	88.5%	Lite

The cheapest model in the matrix tied for first place at 6.7× the speed of its own pro sibling — across 4,140 tests with deterministic scoring. The pro-tier alternatives all finished behind it.

Entity Extraction F1 (Eval 3.3, 675 tests, Grounded scoring):

The same Gemini Flash Lite placed first on entity F1 (0.811), beating Claude Opus, Sonnet, GPT-5.4, GPT-5-mini, and Gemini Pro. Two extraction tasks, two first-place finishes for the cheapest model in the suite. Not a one-off.

Relevance Check (Eval 1.6, 675 tests):

Within the Google family, the lite model beat the pro model by 16 percentage points on a binary classification task (89.3% vs 73.3%). The default tier (Gemini Flash) also underperformed its own lite variant (74.7% vs 89.3%). Same provider, same architecture family — the "more capable" model performed *worse* on the simpler task. Pro models seem to over-think binary decisions, second-guessing the obvious answer.

Knowledge Graph Utilization (Eval 3.4, 134 tests):

Within the Claude family, the lite model beat both the default and pro variants:

Rank	Model	Graph Score	Graph Usage Rate
1	claude-haiku-4-5	4.80	93.3%
2	claude-sonnet-4-6	4.20	80.0%
3	claude-opus-4-6	3.60	80.0%

Going up the Claude tier *reduced* knowledge-graph usage. Opus, the most expensive model in the entire matrix, was beaten by its cheapest sibling by a 33% relative margin on a 5-point scale.

The synthesis: tier labels are pricing labels, not capability labels. Each provider's lite-default-pro gradient is monotonic in price; it is not monotonic in task performance. Within-provider tier inversions are not edge cases — they appeared on multiple evals across all three providers in this study.

3.3 Where all models failed equally — and what prompt engineering did about it

The Q1 batch surfaced one task where every model in the matrix clustered at the bottom: **entity relationship extraction**, the relationship-accuracy column of Eval 3.3.

On the original prompt, the best model (Gemini Flash Lite) scored 23.8% relationship accuracy. The worst (GPT-5.4) scored 10.0%. Median across all 9 models was approximately 14%. Pro models did not outperform lite — GPT-5.4 was last.

We took a single fixed lite-tier model — claude-haiku-4-5, which scored 15.4% on the original prompt — and ran three rounds of structured prompt iteration. Same model, same test set, same scoring methodology. Only the prompt changed.

Milestone	Best Prompt	Relationship Accuracy	Improvement vs original
Original	v2_step_by_step	0.238	baseline
Round 1	B: Two-Pass	0.324	+36%
Round 2	F: Count Calibration	0.463	+94%
Round 3	S: Quote Evidence	0.553	+132%

More than doubled relationship accuracy on a lite-tier model with no model change. The Round 3 production prompt (Quote Evidence) combined two-pass extraction structure (Round 1's contribution), count-calibration to prevent over-extraction (Round 2), and evidence grounding requiring direct quotes from source (Round 3).

A model swap from worst (GPT-5.4 at 10%) to best (Gemini Flash Lite at 23.8%) on the original prompt would have yielded +13.8 percentage points. Three rounds of prompt engineering on a single fixed model yielded +39.9 points — 2.9× the lift of any model swap on the original prompt.

The narrow conclusion: when every model clusters at low scores, prompt iteration against a fixed model is worth trying before concluding the task is unsolvable — the lift can be large enough to make an unusable task production-ready.

3.4 The best prompt for one model is sometimes not the best prompt for another

The same relationship-extraction work surfaced a second finding. Within Round 2, two prompts ran across five models on the same 25 test passages:

Prompt	gem-flash-lite	claude-haiku	claude-sonnet	gemini-3-flash	claude-opus
F: Count Calibration	0.492	0.446	0.483	0.433	0.449
I: Negative Examples	0.299	0.277	0.245	**0.454**	0.207

Prompt I — "Negative Examples" — is the worst prompt of the round for four of the five models tested, and the best prompt for gemini-3-flash. Same prompt text, same test set, same scoring. The relative ranking flips entirely depending on which model runs it.

Prompt and model are not independent dimensions to optimize. A prompt that wins on Claude Sonnet cannot be assumed to win on Gemini Flash; one that loses on most models may quietly be the best choice for a specific model. Production architectures that offer a model switcher — or that route different requests to different models — need to validate prompts per model, not pick one prompt and assume it travels.

3.5 What behavioral checks caught that quality scoring would have missed

Two findings in this study came from the simplest possible eval layer — Behavioral, the category that asks only "did the system function?" One (latency) is invisible to quality scoring entirely. The other (empty output) would surface in quality scores as catastrophic averages, but Behavioral catches it directly with a one-line check at zero cost and points at the failure mode instead of leaving you to diagnose it from the aggregate.

GPT-5-nano latency anomaly (Eval 1.10, 990 tests): pure latency profiling, no quality metric.

Rank	Model	Avg ms	Tier
1	gemini-3.1-flash-lite	1,937	Lite
2	claude-haiku	3,684	Lite
3	gemini-3-flash	6,842	Default

4	gpt-5.4	7,932	Pro
5	claude-sonnet	10,634	Default
6	**gpt-5-nano**	**11,631**	**Lite (anomaly)**
7	claude-opus	11,955	Pro
8	gpt-5-mini	14,756	Default
9	gemini-3.1-pro	16,223	Pro

GPT-5-nano is OpenAI's "lite" tier — supposedly the fast, cheap option. It is slower than GPT-5.4 (their own pro tier) and slower than two default-tier models from other providers. The lite-default-pro labels promise a speed gradient that does not hold within OpenAI's lineup. Quality scoring would never have caught this. Only Behavioral profiling does.

Summarization zero-output (Eval 1.9, 378 tests): gpt-5-nano returned empty text on 47.6% of inputs overall — and 100% of short inputs (12 of 12 in the small-summarization bucket). gpt-5-mini was also affected (19% overall, 67% on short inputs). gpt-5.4 had zero empties. The pattern is a clean model-size gradient on a single OpenAI failure mode: smaller models silently produce no output when the input is short enough that the compression task degenerates.

The generalizable lesson: any production eval suite should run a Behavioral layer across the full model matrix, regardless of which quality scorer it uses for the headline number. The cheapest possible signal layer catches failure modes invisible to every other category.

---

4. Big-Picture Conclusions About LLMs

The conclusions below are meta-level claims about LLMs as a production category, drawn from the data in this study. Each is anchored to specific eval evidence.

BP-1: There is no "one model to rule them all"

The implicit industry hope (and fear) has been that one frontier model would dominate every task. The §3.1 data is incompatible with that picture. No model placed top-3 on every eval. The same Gemini Flash Lite that won extraction tasks finished dead last on content generation. The same GPT-5-mini that tied for first on conversational memory finished dead last on web-grounded queries.

The takeaway is structural rather than competitive: each of these models is good at different things because they were trained against different objectives, on different data, with different alignment choices. The differences won't disappear with a better frontier model — they're the result of the providers making different bets.

BP-2: Capability is task-shaped, not tier-shaped

The lite / default / pro tiering scheme is a pricing gradient, not a capability gradient. Within-provider tier inversions occurred on multiple evals across all three providers in this study. Lite-tier models won outright on five of the largest evals. Pro-tier models won fewer evals than would be expected if price predicted quality. The right unit of model selection is *task profile × model strength*, not *budget × tier*.

BP-3: Models have specialty profiles — treat them like specialists, not generalists

The provider-level summary in §3.1 names broad strengths. Going one level deeper, the per-model specialties from this study:

- **Gemini Flash Lite**: extraction specialist. Wins entity extraction, action extraction; loses badly on creative content. Use for structured-output tasks where speed and accuracy matter and creativity doesn't.
- **Claude Sonnet**: research specialist. Wins web grounding and replan; ties for first on thread continuation. Use for long-form reasoning, multi-turn conversation, search-augmented research.
- **Claude Haiku**: knowledge-graph specialist. Wins graph utilization by a wide margin. Use for any task pulling structured context (graphs, hierarchies, taxonomies) into reasoning.
- **GPT-5.4**: content specialist. Wins fact coverage on content generation. Use when the model is *writing* something with required facts to include.
- **GPT-5-mini**: conversational-memory specialist. Wins confirmation response; ties for first on thread continuation. Use for multi-turn flows where remembering prior context is the bottleneck.

The implication for production architectures: stop describing models as "good" or "bad" — the verdict depends on the task. Build a routing layer that picks per-task.

BP-4: The intelligence-per-dollar curve is steep, and recent

Two years ago the sentiment was "use the most expensive model you can afford and hope it's smart enough." The Q1 2026 data shows that sentiment is now empirically wrong on most production tasks. Lite-tier models from Google and Anthropic — at one-fifth to one-tenth the cost of pro tier — are competitive or winning on action extraction, entity extraction, knowledge-graph utilization, relevance check, conditional logic, task classification, and note generation.

This is a competitive-dynamics observation more than a technical one. Lite-tier pricing is aggressive because the providers are competing for market share. The pricing may converge as the market matures — but for now, paying for pro tier on every request is leaving money on the table.

BP-5: Vendor benchmarks (MMLU, GPQA, HumanEval) are weak predictors of task performance

If public benchmark rankings predicted task rankings, GPT-5.4 and Claude Opus would be winning everything in this study. They aren't. Gemini Flash Lite — a model that does *not* top public benchmarks — wins two of the largest evals. The benchmarks measure capabilities (reasoning, math, knowledge breadth) that don't reliably transfer to specific production tasks like extraction, classification, retrieval, and structured output.

The only way to know which model wins a given task is to run that task. This is the strongest single argument for in-house eval suites.

BP-6: "More information" is not always better

Web search hurt accuracy on product pricing in this study (a -0.8% boost when search was enabled) and added 0% value on technical-API queries. The framing "if I add more context, the model will do better" is not always true. Stale or low-quality external data can actively confuse a model whose training data was already correct.

The same logic applies to retrieval-augmented generation, knowledge-graph injection, and tool use. Whether to provide context is itself a decision that needs eval data. "Always inject context X" is a worse architecture than "inject context X when the eval says it helps for this task category."

BP-7: Prompts and models are not independent dimensions

Per §3.4, the same prompt can be best for one model and worst for another on identical test data. Prompt engineering and model selection cannot be treated as separable optimization passes. When migrating prompts across models — or when offering users a model switcher — prompts need to be re-evaluated per model and sometimes re-engineered. The implication for routing architectures: a "best prompt" library is really a best (prompt, model) pair library.

BP-8: Eval-driven model selection is non-negotiable

This is the synthesis of everything above. Given that no single model wins (BP-1), tier doesn't predict capability (BP-2), models have specialty profiles (BP-3), pro tier rarely earns its premium (BP-4), vendor benchmarks don't transfer (BP-5), adding context doesn't always help (BP-6), and prompts don't transfer

cleanly between models (BP-7) — there is no defensible way to design a production AI system without an in-house eval suite.

The cost of building such a suite is real but bounded. The cost of *not* building it — paying premium prices for lite-tier work, missing systematic provider failure modes, locking into the wrong model, defaulting to whatever the framework's quick start suggested — compounds in production indefinitely.

---

5. Recommendations

The recommendations below follow from the conclusions above. They are addressed to teams building or maintaining production AI systems.

5.1 Build the eval pyramid bottom-up

Start with Structural validation (does it parse?) and Behavioral checks (did it produce output? how fast?). These are zero-cost and catch the loudest failures. Add Grounded comparison once you have labeled data for the task. Add Coverage verification when retrieval or fact preservation matters. Add Semantic similarity when meaning preservation matters and exact wording doesn't. Reserve LLM-as-judge for the dimensions that genuinely require subjective judgment.

The most common eval-design mistake is starting at LLM-as-judge because it's the framework default. Start at the bottom of the pyramid and only escalate when you need to.

5.2 Invest in ground truth where you can

Ground truth has compounding returns. Twenty-five hand-labeled passages for entity extraction in this study powered 675 tests across 9 models, plus three rounds of prompt engineering across 1,775 additional tests — all with precisely comparable scores. LLM-as-judge cannot deliver this precision because its scores vary between runs.

The labeling is upfront work. The amortized per-test cost is negligible. For any task where correctness can be defined before the eval runs, the investment pays off across the lifetime of the system.

5.3 Route per-task, not per-provider

Pick the model based on task profile and the empirical winner from your own evals — not based on which provider you have the closest relationship with, and not based on which tier feels safe. The data in this study makes this case repeatedly: a single model commitment leaves quality on the table on most tasks.

Multi-provider routing infrastructure (a thin layer that selects the model per request) is now standard for serious production systems. The cost of building it is small relative to the routing-quality gains it enables.

5.4 Consider the categories we did not apply

The five additional categories listed in §1.6 should be considered for future batches based on deployment context, not applied uniformly. The prioritization that follows from this study:

- If the system faces untrusted user input, Adversarial / Red-Team is the highest-priority gap to close — quality scoring tells you nothing about resilience.
- If the system has subjective quality dimensions where reasonable evaluators could disagree, Human Evaluation is the calibration gold standard and worth the cost periodically.
- If the system generates executable artifacts (code, SQL, HTML, CSS, JS), Code Execution catches errors invisible to text-based scoring.
- Robustness and Safety follow as the deployment context demands.

The lexicon names these categories explicitly so they can be planned for, not stumbled into.

Appendix A — The 30 evaluations at a glance

#	Eval	Tests	Best model	Primary methodology
1.1	Memory Tag Extraction	900	gpt-5-mini + chain_of_thought	Grounded
1.2	Personalization Update	450	gemini-flash-lite	Semantic
1.3	Tag Hierarchy	1,575	gpt-5 + librarian prompt	Grounded
1.4	Note Generation	1,350	gpt-5-mini + v2_structured	Coverage
1.5	AI Search Tag Extraction	900	gemini-flash-lite + chain_of_thought	Grounded
1.6	Relevance Check	675	gemini-flash +	Grounded

			zero_shot/2shot (perfect 1.0)	
1.7	Action Extraction	4,140	gemini-flash-lite (97.6%, tied with gemini-pro)	Structural
1.8	Personalization Extraction	450	All 9 pass; decision_focused prompt best	Semantic
1.9	Summarization	378	All 9 pass	Coverage
1.10	Speed Testing	990	gemini-flash-lite (1,937ms avg)	Behavioral
2.1	Task Classification	1,800	All tested pass	Grounded
2.2	Evaluate + Replan	1,485	claude-sonnet (97.5%)	Grounded
2.3	Conditional Logic	1,350	gemini-flash-lite (97.8%)	Grounded
2.4	Action Ordering	20	N/A (deterministic algorithm)	Structural
2.5	Agent End-to-End	117	All score 4.0/4.0; Sonnet fastest	Behavioral
2.6	Content Generation	405	gpt-5.4 (0.852 fact coverage)	Coverage
2.7	Confirmation Response	540	gpt-5-mini (0.991)	Coverage
3.1	Recursive Summarization	540	All 9 pass	Coverage
3.2	Context Compression	180	Claude models dominate	Semantic
3.3	Entity Extraction	675	gemini-flash-lite (F1 0.811); relationship	Grounded

			accuracy improved 23.8% → 55.3% via prompt engineering	
3.4	Knowledge Graph Context	134	claude-haiku (4.80, 93.3% usage)	Coverage
3.5	Semantic Pattern-Match Routing	105	N/A (embedding pattern test)	Semantic
3.6	External Source Capture	134	claude-sonnet (0.870 keyword coverage)	Coverage
3.7	File Upload Extraction	68	All pass	Structural
4.1	Project Instructions Injection	270	gemini-flash/pro, claude-opus (5.0/5.0)	Judged
4.3	Tree Consistency	15	N/A (deterministic)	Structural
5.1	Thread Continuation	30	gpt-5-mini & claude-sonnet tied 0.883	Coverage
5.2	Scripted Response Matching	74	N/A (embedding pattern test)	Semantic
5.3	Web Grounding	270	claude-sonnet (0.973, +9.3% boost)	Coverage
5.4	Provider Consistency	90	gemini-flash + claude-sonnet (0.891 pairwise)	Semantic

Appendix B — Glossary of categories and techniques

Categories (what we measure):

- **Structural**: form/schema check on the output. JSON parseability, regex, count, type, format rules.
- **Grounded**: comparison against pre-labeled ground truth. F1, classification accuracy, set match.
- **Semantic**: embedding-based similarity. Cosine distance in vector space.
- **Coverage**: required facts / keywords appearing in output. Substring match.
- **Behavioral**: did the system *function*? Output produced, latency, convergence.
- **Judged**: LLM-as-judge scoring on subjective rubric.

Categories named but not applied in this study:

- **Adversarial / Red-Team**: deliberately break the model with hostile inputs.
- **Human Evaluation**: domain experts scoring outputs.
- **Robustness / Perturbation**: same model, perturbed inputs (typos, paraphrasing).
- **Safety / Toxicity / Bias**: specialized classifiers for harmful content.
- **Code Execution / Runtime**: output is run, not read.

Cross-cutting techniques (how we operationalize measurements):

- **LLM-as-Q&A Probe**: questions answered from reference vs output, compared.
- **Multi-Step Composite Scoring**: weighted total of multiple sub-metrics.
- **A/B Delta Scoring**: same eval × two configurations, report the delta.
- **Layered / Pyramid Scoring**: cheap-to-expensive in sequence.
- **Threshold Sweep**: data-driven cutoff selection.
- **Unique-Anchor Verification**: fictional facts as RAG verification probes.
- **N-Shot Stability Probe**: same input × N runs, measure variance.

Appendix C — Prompt × Model Interaction Data

Section 3.4 and BP-7 claim that the best prompt depends on which model runs it. The densest empirical evidence for that claim comes from the relationship-extraction work in Eval 3.3, which ran the same set of prompts across the same five models in two rounds. The full matrices appear below.

Round 1 — six prompts × five models (relationship accuracy)

Prompt	gem-flash-lite	claude-haiku	claude-sonnet	gemini-3-flash	claude-opus
v2 baseline	0.242	0.177	0.209	0.155	0.256
A: Explicit Schema	0.213	0.264	0.238	0.279	0.237
B: Two-Pass	0.356	**0.364**	**0.309**	**0.448**	0.250
C: Rel-First	0.376	0.318	0.266	0.218	0.269
D: Sentence	**0.405**	0.328	0.255	0.299	**0.273**
E: Few-Shot	0.364	0.310	0.296	0.228	0.269

Best prompt per model splits into two clusters: B (Two-Pass) wins for claude-haiku, claude-sonnet, and gemini-3-flash. D (Sentence) wins for gem-flash-lite and claude-opus. No single prompt wins for all five models.

Round 2 — six prompts × five models (relationship accuracy)

Prompt	gem-flash-lite	claude-haiku	claude-sonnet	gemini-3-flash	claude-opus
B: Control	0.362	0.332	0.295	0.381	0.239
F: Count Calibration	**0.492**	**0.446**	**0.483**	0.433	**0.449**
G: Quality Gate	0.340	0.408	0.418	0.419	0.389
H: Hybrid Sentence	0.440	0.357	0.303	0.387	0.285
I: Negative Examples	0.299	0.277	0.245	**0.454**	0.207
J: Verify Step	0.340	0.350	0.286	0.329	0.293

F (Count Calibration) wins for four of five models — the most consistent prompt in the round. I (Negative Examples) is the worst prompt for four of five models, and the best for gemini-3-flash. Section 3.4 highlights this row pair as the cleanest in-round flip in the entire study.

What this data is and isn’t

Two rounds, one eval, five models — this is the most concentrated empirical data on prompt × model interaction in the Q1 2026 study. Other evals tested prompt variants but did not run them against the full 5- or 9-model matrix in clean form, so equivalent matrices for other tasks would require new eval runs.

The point the data supports is narrow but defensible: prompt × model is a real interaction, not a fixed pairing. Production architectures that route across models, or offer users a model switcher, should treat each (prompt, model) pair as a separately-evaluated unit.

About the Author

David Hood — Master's in Industrial Engineering. 6 years at Texas Instruments as an IE. 11 years running an automation/SEO business. Backend/Python developer and systems thinker. Hands-on with every client engagement. Based in Dallas, TX.

How to Cite

Suggested citation:

Hood, D. (2026). Q1 2026 LLM Evaluation Study: Methodology and Findings. 42Robots. https://42robots.ai/papers/q1-2026-llm-evaluation/

BibTeX:

```
@techreport{hood2026q1llm,
  author = {Hood, David},
  title  = {Q1 2026 LLM Evaluation Study: Methodology and Findings},
  institution = {42Robots},
  year   = {2026},
  url    = {https://42robots.ai/papers/q1-2026-llm-evaluation/}
}
```

Contact

Website: 42robots.ai

LinkedIn: linkedin.com/in/robotzero

Email: david@42robots.ai

Available for AI consulting.