

Silly benchmarks that tell you something real (pelican, skateboard, and friends)

Playful tasks such as drawing a pelican on a bicycle can expose planning, geometry, tool use, and persistence when their grader makes the signal explicit.

Updated October 15, 2026

A silly benchmark earns its place when the joke makes a real failure legible and the grader checks that failure instead of rewarding vibes.

“Draw a pelican riding a bicycle” sounds like a party prompt. It is also a compact compositional test. A system must represent the bird, the bicycle, the relationship between them, and a valid output format. If the output is SVG, the task adds syntax, layering, geometry, and renderability. One funny image can expose several mistakes that a broad average hides.

The pelican is not the metric

A gallery of amusing failures is a demonstration, not yet a benchmark. To become one, the task needs a fixed instruction, versioned evaluation conditions, and explicit checks. Does the SVG parse? Does it render without external assets? Are two wheels present? Is a pelican distinguishable from another bird? Does its pose plausibly place it on the bicycle?

Some checks are deterministic. XML parsing, element bounds, forbidden raster images, and canvas dimensions do not need a judge model. Visual composition is harder. A rendered-image classifier, a small human rubric, or a blinded pairwise review may be appropriate, but the method and uncertainty must be reported.

[Benchmark Pelican](#) makes the format and relationship part of the test. Its value is not evidence that bicycle birds matter in production. Its value is a clear probe of whether a system can satisfy unusual, simultaneous constraints in a visual artifact.

A skateboard can test a whole loop

“Build a playable skateboard trick scene” sounds like a game jam. It can test more than code completion. The model must create a scene that loads, map controls, update state over time, detect a trick, and give the player feedback. A screenshot cannot prove that any of that works.

A useful grader starts the scene, sends input, observes state transitions, and checks an objective outcome. It can verify that the skater moves, a jump begins and ends, a named trick increments a score, and the program avoids errors. These are execution checks, not opinions about whether the game looks cool.

[Benchmark Skateboard](#) therefore targets a small vertical slice of interactive software. It is playful enough that a failed attempt is easy to share, while the meaningful signal comes from reproducible play behavior.

Humor helps diagnosis

Memorable prompts have three practical advantages.

First, humans inspect them quickly. A pelican under a bicycle is easier to notice than a small change in a language-model loss. A skateboard that cannot jump is easier to explain than an aggregate code score.

Second, unusual combinations stress composition. Models often handle each familiar object or action separately but lose a relationship when requirements collide. A playful scene can make that collision visible without specialist knowledge.

Third, the artifact travels. People share a striking result, rerun it, and compare failure modes. That attention can produce more test cases and better graders—as long as the project keeps versions and does not substitute popularity for validity.

Humor also creates risks. A judge may reward attractive output and ignore the requirement. Prompt ambiguity may become part of the entertainment. Repeated viral examples can enter training data. Report these limitations directly.

Other silly tasks with sharp edges

A support agent asked to apologize for the wrong pizza while staying under a refund limit sounds light. Underneath, it tests policy constraints, arithmetic, state updates, and recovery language. [Benchmark Apology Pizza](#) can grade the final refund and order state deterministically, then separately inspect whether the response communicates the correction.

An honesty test after a failed tool call can also use a deliberately simple scenario. The important question is whether the model claims to know a result it never received. [Benchmark Honesty](#) turns that into a crisp behavioral check.

These tasks work because the story is small and the invariant is clear. The refund must stay within a limit. The tool result is absent. The skateboard must respond to input. A funny wrapper cannot rescue a vague invariant.

How to judge whether the signal is real

Apply five questions:

1. What single capability or failure is the task meant to reveal?
2. Which parts can a deterministic grader verify?
3. Would a superficial imitation pass?
4. Can another evaluator reproduce the environment and score?
5. What does success not prove?

NIST's evaluation guidance starts with objectives and intended use. That discipline matters even for a one-prompt challenge. If the intended use is a shareable regression probe, say so. Do not infer general intelligence, production reliability, or safety from one charming artifact.

Run several seeds and preserve every output, not only the funniest success. Record the model version, system prompt, tool permissions, retry policy, cost, and latency. A task that produces a great screenshot on attempt nine has a different reliability story from one that works on the first

attempt.

Keep the fun and the controls

The best silly benchmarks are invitations to inspect, not shortcuts around measurement. Lead with the artifact because it makes people curious. Follow with the task contract, grader, provenance, and limitations because those make the result useful.

Use them beside serious suites. A vivid failure can tell you where to investigate; repeated, scored runs can tell you whether a fix helped. The pelican gets attention. The benchmark earns trust by showing exactly what happened.

Frequently asked questions

Is a funny prompt a scientific benchmark?

Not by itself. It needs a defined construct, repeatable procedure, and grader that tests observable requirements.

Why use a playful benchmark?

It is memorable and can make a complex failure easy to inspect, but it should complement task-specific evaluation.

Sources

[Pelican bicycle benchmark repository](https://github.com/JoJohanse/pelican-bicycle-benchmark) — <https://github.com/JoJohanse/pelican-bicycle-benchmark>

[NIST Practices for Automated Benchmark Evaluations of Language Models](https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.800-2.ipd.pdf) — <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.800-2.ipd.pdf>