Real Unreal Bench: a benchmark of large multi-system changes in complete Unreal Engine games

Superposition Lab. 2026-11-05. Version v0.

Abstract

We measure whether a model, working alone with command-line tools, can make a large multi-system change in a complete Unreal Engine 5.7 game so that the running game behaves as the request asks. Version v0 has 15 tasks in 10 games, chosen from a pool of over 900 tasks; the reference changes for them range from 5 to 818 game files, generated and re-saved assets included. Nine models each made one run per task, in one harness, on Linux VMs with an L4 GPU, under a limit of 300 turns and the time and cost caps of each task. Deterministic probes of the built and played game decide pass or fail. Model judges are recorded and never decide. For every task, a scripted no-op failed at least one required check and the reference passed them all. The best model passed 9 of 15 tasks and the median model 6. With one run per model and task, the intervals are wide: the leader's is 35.7 to 80.2%, and it overlaps every other model's.

1. Introduction

A game change often spans C++, maps, materials and other assets, and it isn't finished until the game plays correctly. Most published game-development benchmarks test smaller units of that work, or use a language model to decide whether the result is right.

v0 asks a narrower and more expensive question. Given one complete game and one request, does the model's final commit build, start and behave as asked when a fresh copy is played by a script? We report what that shows. Section 9 lists this version's limits.

We sell data and environments for training models, so we have a commercial interest in this benchmark's standing. Our team wrote the harness, chose the tasks and ran the models.

2. Related work

Three benchmarks are closest. Their scores can't be compared with ours or with each other: the tasks, models, harnesses and pass rules all differ.

	GameEngineBench [1]	**CraftBench-UE [2]**	**GameDevBench [3]**	**This benchmark (v0)**
Engine	Unreal Engine 5, native C++	Unreal Engine 5.8, Windows	Godot 4	Unreal Engine 5.7, Linux
Tasks	110, in 9 open-source projects	70: 33 C++, 25 Blueprint or native asset, 12 Python editor scripting	333: 154 tasks and 179 variants, from 57 video and 31 web tutorials	15, in 10 private games
What decides a pass	An LLM judge's verdict, after hidden Play-in-Editor tests	Build, Play-in-Editor tests and asset checks; no LLM decides	Tests in Godot's scripting framework	Build, start and probes of the played game; no LLM decides

	GameEngineBench [1]	CraftBench-UE [2]	GameDevBench [3]	This benchmark (v0)
Best published score	55.5% pass@1 (Claude Fable 5, max effort), then GPT-5.5 xhigh 29.1%, Claude Opus 4.8 max 23.6%, Gemini 3.1 Pro 18.2%	C++ 203/231 (87.9%), Blueprint 110/175 (62.9%), Python 66/84 (78.6%), in the best configuration	53.8% pass@1 (Gemini 3 Pro with screenshots and video), then GPT-5.4 52.0%	60.0% (Gemini 4 Argon), then Claude Opus 5.5, Claude Sonnet 5.5 and GPT-6 Astra 53.3%, one run per task

GameEngineBench [1] uses real open-source projects, and 31 of its 110 tasks are unsolved by all twelve configurations. Its authors note that no test suite can capture every valid implementation, which is why its success rests on a judge's verdict and that judge's calibration. They also note that models ran through different wrappers, so the score differences partly reflect those wrappers. The projects are public, which leaves room for contamination. It covers C++ only.

CraftBench-UE [2] is fully deterministic and is the only one of the three to cover Blueprint and editor work. It finds that C++ beats Blueprint by 30.0 and 42.9 points on paired tasks. Its authors state that the tasks were selected for explicit, checkable outcomes rather than sampled representatively, that integrating many features into a complete game isn't measured, and that each run has one attempt on one operating system through one agent runtime. They name long-horizon projects and creative requirements as future work. They disclose that their organisation develops one of the two editor tool sets it compares. Its C++ track, at 87.9%, leaves little room above it.

GameDevBench [3] is the largest, with deterministic tests, and it shows that visual feedback raises scores (GPT-5.4 from 41.1% to 52.0%). Its projects come from tutorials on a different engine. Its authors say that human validation took under 10 minutes per task and that agents are evaluated through code generation alone.

Where v0 stands against them. We use complete private games, a deterministic pass rule and Linux, and we use one harness for every model. All three have more tasks, and CraftBench-UE measures Blueprint as its own track, which we don't.

3. The tasks

3.1 Summary

v0 has 15 tasks, all rated `hard` in the pool. Each is one written request, in one complete, private, licensed, human-made game, with a start commit and a reference: the change recorded for that request in the game's delivered data. We describe them as large multi-system changes, by the size of the reference and the checks. The counts include generated and re-saved assets: of the 206, 663 and 818 game files of Tasks 12, 11 and 10, 13, 23 and 23 are code.

Table 1. The tasks. Tasks are numbered as on the results page, by kind. Files and systems use the selection's own count (§3.2), which includes generated and re-saved assets. "Required" is the count of required checks, build and start included. The limits are each task's; all ran at 300 turns.

Task	Kind	Game files	Systems	Required	Minutes	Cost cap
1	bug fix	6	config, materials, maps, code	7	180	$40
2	bug fix	20	assets, materials, maps, code	7	480	$150
3	bug fix	6	maps, code	10	180	$40

Task	Kind	Game files	Systems	Required	Minutes	Cost cap
4	change	66	assets, materials, maps, code	9	180	$40
5	change	12	assets, code	8	180	$40
6	correction	17	maps, assets, materials, code	9	180	$40
7	correction	5	config, code	8	180	$40
8	correction	20	materials, code	6	240	$40
9	correction	6	assets, code	7	180	$40
10	feature	818	config, assets, materials, maps, code	7	480	$150
11	feature	663	config, assets, materials, maps, code	8	480	$150
12	feature	206	config, assets, materials, maps, code	8	480	$150
13	feature	6	config, code	8	240	$40
14	feature	37	materials, code	5	300	$60
15	feature	7	code	7	180	$40

Counts come from the game's history. The start-to-reference difference as delivered can be larger: the audit of passed runs read Task 7's as 43 files.

3.2 How the tasks were chosen

Chosen from the pool of live game tasks, by these rules:

1. The game passed a one-time Linux check: it builds and plays on Linux.
2. The task is rated hard, with at least 3 required probes.
3. The reference changes game files (`Source/`, `Content/`, `Config/`, `Plugins/`) in one or more systems (code, maps, materials, Blueprints, other assets, config).
4. Bug fixes, corrections and features are all represented.

One task picked by these rules was dropped as a task error: its start doesn't compile on Linux, so its no-op, its reference and every model's run failed the same way, and it counts for no model. The next task by the same rules, Task 7, took its place.

Task 1's request gained one sentence on 2026-11-04. Its checks didn't change, and the runs that count are all on that version.

3.3 What the set does not cover

The 15 tasks are not a sample of game development. They are hard-rated tasks in games that build on Linux, picked by the rules above. Their distribution across engine systems follows what those rules let through.

4. Environment and harness

The machine. Each run has its own Google Cloud VM, a `g2-standard-8` with one NVIDIA L4, running Linux and Epic's Unreal Engine 5.7.4 image pinned by digest, in a container as an unprivileged user with no way to become root. Rendering is offscreen Vulkan. The agent works offline. VMs came from 13 US

zones at first. GPU shortages then stopped many runs, so 19 zones in Canada, Europe and Asia were added on 2026-11-03 and tried after the US ones. The model call is made from a job outside the VM, so the VM's location doesn't add model latency.

The harness. Every model runs in the same loop, our own game agent, not its maker's agent. It gets the same system prompt (`game-agent/3`) and the same nine tools (`game-agent-tools/1`):

Tool	What it does
`shell`	Runs bash on the VM
`read_file`, `write_file`, `edit_file`, `list_files`	Work on the project's files
`build`	Compiles the editor target
`run_game`	Plays the game, with scripted inputs if given, and returns screenshots, player positions and log errors
`editor_python`	Runs a script in the editor's `unreal` Python module, for maps and assets
`finish`	Ends the run

The prompt gives the request, the map QA plays, and each task's milestones in words: what must build, what must start, and that automatic measurements and reviewers will look at the result. It doesn't give the probes' selectors or thresholds. The reference isn't on the VM during a model's run. The agent's work is saved as a checkpoint after each turn.

The limits, checked before every model call.

- **Turns:** 300, for every model and task.
- **Time:** the task's, 180 to 480 minutes (Table 1). One tool call may take at most 30 minutes.
- **Cost:** the task's model budget, $40 on 10 tasks, $60 on one and $150 on four.
- **Output per turn:** 64,000 tokens for every model. A model's thinking counts inside the cap. See §9.

A run ends when the agent calls `finish` or answers without a tool, at one of the limits, or when a model call keeps failing. It is then graded on its last checkpoint, whatever the reason it ended.

Runs lost to our side. A run that fails for a reason of ours (no VM, a VM lost, a download cut, a grade that failed on the machine) is run again and doesn't count for its model. A run that fails for the model's own work counts as a fail. A run hit by a provider outage is left out rather than counted, and isn't re-run automatically. Any such run has to be started by hand, and until then it is missing and scores 0 in the results. On the 15 tasks at 300 turns, 24 model runs were lost to a VM or to capacity (12 each) and all 24 ran again. 135 runs count.

5. Grading

Only the end state counts. When a run stops, its container is removed and two fresh ones are made on the same VM. The first holds the agent's commit alone: the project at its final checkpoint with our probe plugin put back. It is built and played, and its readings are taken. The second holds trusted copies only, the start and the reference, built and played to set the thresholds. The agent's code never runs beside them. If a grade fails on the machine, it moves to a fresh VM. The agent's own account of its work isn't read for a pass.

Deterministic probes decide. Each task's required checks are a build, a start with no crash, and 3 to 8 probes. A probe is one number taken from the game: a measure of the pawn, camera, actors, widgets, components, assets, light, grading, spawns or the log, during an idle or scripted play, or in a headless editor pass. A play samples every 0.5 seconds by default, and some required probes sample every 0.1, 0.2 or 0.25 seconds. A threshold takes one of three forms: a literal; calibrated between the start's and the reference's values measured in that grade; or a multiple of the start's value (`start_times`), on 2 of the 84 required probes, both on Task 7. A task passes only when every required check passes.

Judges never gate. Eight of the 15 tasks also carry model judges (of screenshots, of the diff, of the agent's actions). They are recorded as quality signals, are always non-required and can't change a pass. Their cost is counted apart from the model's.

The numbers we report.

- **Pass rate:** tasks passed over tasks scored. A task missing a counted run scores as a fail for that model. It is the only number models are ranked on.
- **Required probes passed:** the share of a task's required probes passed, averaged over tasks, the build and the start not counted. A run that does nothing doesn't score 0 on this, because the start already passes some probes (Table 2, which counts the build and the start too).
- **Adjusted partial credit:** counts only what the start fails, `(passed − start passes) / (required − start passes)`, floored at 0.
- **Cost and turns:** the model's own USD per counted run and median turns. VM cost isn't included. We report turns rather than minutes because the script's records don't separate a run's working time from its grading for every model.

6. Validity audit

Every task was checked by two scripted agents that call no model and go through the same start, checkpoint and grade as a model's run. The **no-op** changes nothing, so the start is graded. The **oracle** writes the reference's changes onto the start. A task stays only if the no-op fails at least one required check and the oracle passes every required one. Scripted runs never count for a model. The audit of the first tasks ran after their baselines started, since a baseline takes up to five hours.

Table 2. The audit. Read from the runs' grades on 2026-11-05.

Task	Required checks	No-op failed	Oracle passed
1	7	3	7
2	7	4	7
3	10	4	10
4	9	3	9
5	8	4	8
6	9	3	9
7	8	1	8
8	6	3	6
9	7	2	7
10	7	4	7

Task	Required checks	No-op failed	Oracle passed
11	8	5	8
12	8	5	8
13	8	4	8
14	5	3	5
15	7	3	7

All 15 tasks pass the audit: the no-op passed 0 of them and the oracle every one. The task dropped in §3.2 failed it, as its start doesn't build.

Each no-op and oracle ran once to a graded result. An oracle's total score can sit below 1 because of non-required checks: a scripted run has no tool log for an actions judge to read, and its frames can miss what a judge looks for.

An earlier check, on two Unreal tasks ported to Linux for another purpose, found every probe read the same as on Windows to at least 15 significant digits.

7. Models and results

7.1 Models

Nine models, one run per model per task: Claude Opus 5.5, Claude Sonnet 5.5 and Claude Haiku 5.5 (Anthropic), GPT-6 Astra and GPT-6.1 Sol (OpenAI), Grok 4.7 (xAI), Kimi K3 (Moonshot AI), Qwen3.8 Max (Alibaba) and Gemini 4 Argon (Google). Claude Fable 5.1 is left out for cost. There are no human baselines in v0.

7.2 Results

Table 3. Per model. Models rank on pass rate alone, out of the scored tasks; the interval is the 95% Wilson interval. Partial credit never orders a model.

Model	Passed	Pass rate (95% interval)	Required probes passed	Adjusted partial credit	Model USD per task	Median turns
Gemini 4 Argon	9 of 15	60.0% (35.7–80.2%)	90.8%	77.2%	$15.86	129
Claude Opus 5.5	8 of 15	53.3% (30.1–75.2%)	89.4%	75.6%	$19.94	120
Claude Sonnet 5.5	8 of 15	53.3% (30.1–75.2%)	88.2%	76.1%	$9.31	100
GPT-6 Astra	8 of 15	53.3% (30.1–75.2%)	87.2%	76.4%	$5.38	42
GPT-6.1 Sol	6 of 15	40.0% (19.8–64.3%)	84.2%	77.9%	$1.07	53
Grok 4.7	4 of 15	26.7% (10.9–52.0%)	65.9%	48.2%	$5.25	176
Claude Haiku 5.5	4 of 15	26.7% (10.9–52.0%)	61.0%	46.1%	$2.58	88
Kimi K3	3 of 15	20.0% (7.0–45.2%)	65.5%	46.8%	$13.58	273
Qwen3.8 Max	3 of 15	20.0% (7.0–45.2%)	61.1%	44.8%	$7.07	122

Table 4. How the runs ended, per model. Counted runs.

Model	Called `finish` or stopped	Turn limit	Time limit	Cost limit	Output cap (§9)
Gemini 4 Argon	14	1	0	0	0
Claude Opus 5.5	14	0	0	1	0
Claude Sonnet 5.5	15	0	0	0	0
GPT-6 Astra	15	0	0	0	0
GPT-6.1 Sol	15	0	0	0	0
Grok 4.7	14	1	0	0	0
Claude Haiku 5.5	10	1	0	0	4
Kimi K3	8	7	0	0	0
Qwen3.8 Max	8	2	5	0	0

Table 5. Per task. P passed, F failed.

Task	Opus	Sonnet	Haiku	Astra	Sol	Grok	Kimi	Qwen	Gemini	Passed
1	F	F	F	F	F	P	F	F	P	2
2	F	F	P	P	F	F	F	F	F	2
3	F	P	F	P	P	F	F	F	F	3
4	F	F	F	P	P	F	F	F	P	3
5	P	F	F	F	F	P	F	P	P	4
6	F	F	F	F	F	F	F	F	F	0
7	F	F	P	F	P	F	F	P	F	3
8	P	P	F	P	F	F	P	F	P	5
9	P	P	P	P	P	P	F	F	F	6
10	F	F	F	F	F	F	F	F	F	0
11	P	P	F	F	F	F	F	F	P	3
12	P	P	F	P	F	F	F	F	P	4
13	P	P	F	P	P	F	F	F	P	5
14	P	P	F	F	F	P	P	F	P	5
15	P	P	P	P	P	F	P	P	P	8

8. Analysis

Ranking. Every model's interval overlaps the leader's, so none can be ranked apart from it: Gemini 4 Argon's 35.7 to 80.2% reaches Kimi K3's and Qwen3.8 Max's 7.0 to 45.2%. With one run per model and 15 tasks, even the widest gap, 6 tasks, is within the intervals.

Cost and turns. GPT-6.1 Sol ($1.07 per task, 6 passed), GPT-6 Astra ($5.38, 8) and Gemini 4 Argon ($15.86, 9) are the models no other model beats on both pass rate and cost. Rank correlation of pass rate with median turns across the 9 models is $\rho = -0.36$.

Where models fail. Pass rate by kind: feature 25 of 54 runs (46%), correction 14 of 36 (39%), change 7 of 18 (39%), bug fix 7 of 27 (26%), over few tasks each. 82 counted runs failed. 27 of them (33%) gained no check over the start. 60 (73%) ended by the agent's own `finish` rather than a limit. Median adjusted partial credit of the failed runs is 42%, and 67% among those that gained at least one check.

Which tasks separate models. No task is passed by all 9. Two are passed by none: Tasks 6 and 10. On Task 6, 7 of 9 runs passed 7 of 9 required checks. On Task 7, 3 of 9 passed, and 5 of the 6 that failed missed one required check. On Task 10, four runs passed 6 of 7, four passed 4 of 7 and one passed 3 of 7.

9. Limitations

- **One run per model per task.** With 15 tasks, the intervals are wide, and models a task or two apart aren't ranked apart.
- **Per-turn output cap.** A turn's output is capped at 64,000 tokens for every model, its thinking included. A run the cap ends counts as a failure. Table 4 shows the count.
- **Cost.** Cost is the model's own calls at list prices, priced from the records, without the VMs.
- **Time limits.** A slow provider gets fewer turns in the same minutes.
- **Spread across games.** Four tasks come from one game and two each from two others; the other 7 games have one task each.
- **One harness.** All models use the same tools and prompt. We say nothing about a model in its maker's own agent.
- **No human baseline.** v0 has no developer's result to compare with.
- **Conflict of interest.** We sell in this area and wrote everything here (§1).

10. What's next

These are candidates for later versions, not commitments.

- Run mutants and a red team on the tasks, and publish how many each check caught.
- Change the size rule to count hand-written work, leaving out generated and re-saved assets.
- Add tasks written under the later task rules, and rate difficulty from results rather than by label.
- More runs per model, so intervals can separate models. Human baselines.

References

[1] B. La, S. Chang, B. Kim, J. Bae, A. A. Beg, S. Chang, G. Gonzalez-Pumariega, K. Goyal. *GameEngineBench: Evaluating Coding Agents on Real C++ Runtime Environments.* arXiv:2607.03525, 2026. https://arxiv.org/abs/2607.03525

[2] S. Wu, K. Calderone, A. Tsen. *CraftBench-UE: Deterministic Evaluation for Coding Agents in Unreal Engine.* arXiv:2609.23142, 2026. https://arxiv.org/abs/2609.23142

[3] W. Chi, Y. Fang, A. Yayavaram, S. Yayavaram, S. Karten, Q. A. Wei, R. Chen, A. Wang, V. Chen, A. Talwalkar, C. Donahue. *GameDevBench: Evaluating Agentic Capabilities Through Game Development.* arXiv:2602.11103, 2026; ICML 2026 poster. https://arxiv.org/abs/2602.11103