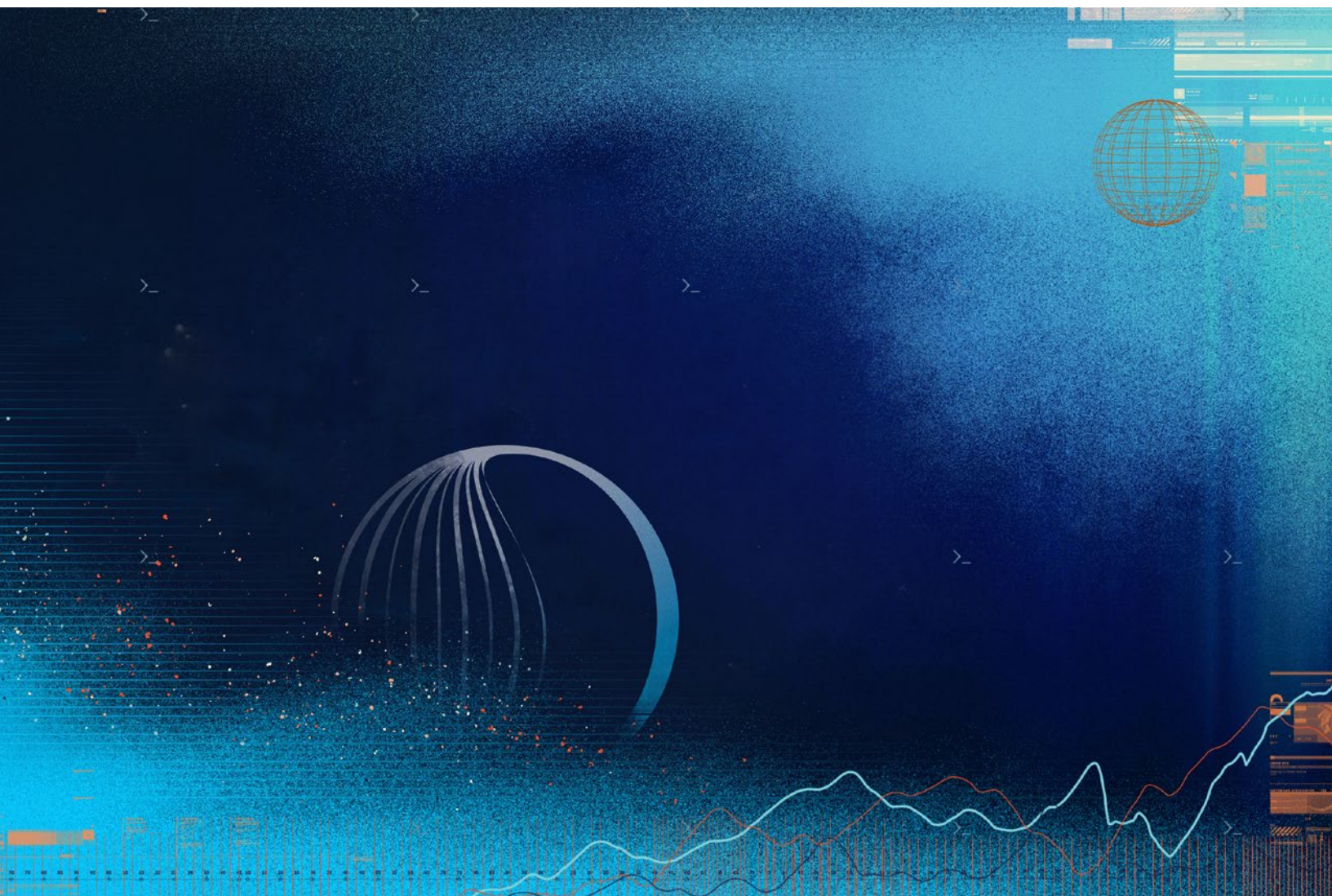




The CPU Behind the Agents: Benchmarking Intel Xeon 6 for Agentic AI Workloads

An independent Terminal Bench 2 evaluation of
AWS m8i (Intel Xeon 6), m7i (4th Gen Intel Xeon
Scalable), and m8a (5th Gen AMD EPYC)

AUTHORS

Jonathan Fellows

Performance Validation Engineer | Signal65

Russ Fellows

VP, Labs | Signal65

IN PARTNERSHIP WITH



OCTOBER 2026

Executive Summary

Agentic AI shifts meaningful work onto the CPU. Between model responses, autonomous agents run shell commands, manipulate files, compile code, execute tests, and manage containers. At fleet scale, these events determine how fast tasks complete, how many agents a server can sustain, and what that work costs. All of this work either runs on the CPU or is managed by services that do.

Signal65 independently tested multiple public cloud instance sizes and CPU types using an agentic record and replay framework. By using both Intel Xeon and AMD Turin CPU types and two different instance sizes, we were able to measure the impact of these CPU choices on the performance, and more importantly the cost, of agentic workloads.



The instance choices included AWS m8i (Intel Xeon 6, Granite Rapids), m7i (4th Gen Intel Xeon Scalable, Sapphire Rapids), and m8a (5th Gen AMD EPYC, Turin) using an agentic record and replay tool. The testing used Terminal Bench 2 with recorded inference responses to eliminate LLM variance between runs, emphasizing on-instance execution: the agent's commands, containerized (sandboxed) task environments, and orchestration. Each test was repeated three times at different concurrency levels with up to 100 processes tested.

Testing found the following highlights:



Clear generational differences: Compared with prior-generation Intel m7i instances, newer m8i instances completed tests 6.9 – 11.0% faster and at 2.2 – 6.6% lower compute cost.

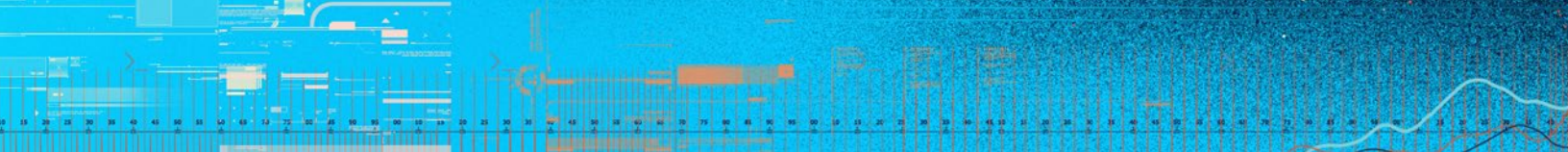


Intel and AMD have similar economics: When comparing price-performance, m8i test cost was at parity or better at four of five tested concurrency levels, with an approximately 5.5% cost advantage at concurrency 20, m8i's most cost-efficient tested operating point. m8a instances completed tests faster at an approximately 15% higher hourly rate; performance details are in the competitive results section.



Strong scale-up utilization when orchestration supplies enough work: On m8i.16xlarge, as our testing raised the number of concurrent processes, average CPU utilization increased to approximately 54%, with peaks near 100%, compared to average utilization rates of less than 10% using the standard Terminal Bench single-process runs.

For this replay workload, m8i is the clear upgrade for existing m7i deployments and offers competitive economics against m8a, while m8a remains the latency leader. The scale-up result also shows that larger systems can support much more concurrent agent work, but only when the orchestration layer supplies enough independent work.

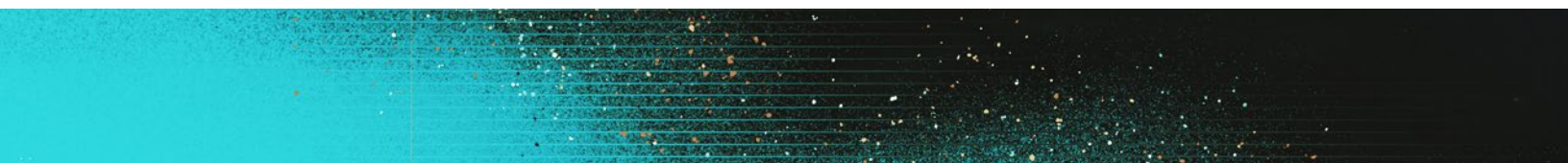


Why the CPU Matters for Agentic AI Workloads

Conventional LLM interactions often consist primarily of inference workloads gated only by a GPU: a prompt goes to an LLM running on accelerators, and tokens come back. Agentic tasks work differently: between every burst of model output, the agent performs multiple actions. It runs commands, reads and writes files, compiles code, executes tests, parses results, and decides what to do next. In agentic workloads of the kind Terminal Bench 2 measures, the model's "thinking" alternates continuously with work performed in a real execution environment, all running on general-purpose CPU cores.

This changes the infrastructure math in several ways. First, CPU execution sits directly in the latency path between model turns. Second, the CPU and orchestration layer determine how many concurrent agentic tasks a server can sustain. Finally, other system services compete for the same resources, which may impact agentic performance as well. The Signal65 tests measure replayed agent execution and orchestration during normal operations.

The test results show that concurrency changes CPU utilization. Across the smaller 2xlarge (8 vCPU) configurations, the median utilization increased from approximately 35% CPU to roughly 65% as concurrency rose from 5 to 25 concurrent tasks. Moreover, to compare larger hardware configurations, the workload itself must be scalable to levels high enough to use the additional CPU resources, while still reporting the efficiency of all workloads.



What We Tested

The tool Signal65 chose is a well-known, open-source tool named "Terminal Bench 2" (aka TB2) that runs realistic, multi-step agentic tasks: building software from source, training models, path tracing, data processing, and similar terminal-driven work, executed by an autonomous agent inside a containerized environment. Each task requires the agent to plan, issue shell commands, react to their output, and iterate until the task completes or times out. TB2 is a benchmark of the full agentic loop rather than of any single component.

The challenge in using agentic workloads for hardware comparison is that the specific model, or LLM, has a significant impact on tests. Typically, LLM calls introduce variance in latency and behavior that can overwhelm hardware differences. The record-replay file we used keeps the model responses the same, so each instance receives the same recorded decisions and tool calls. Replay did not alter the responses or their timing from the recorded values.

This design makes TB2 replay a reliable tool for measuring agentic execution under a fixed response sequence, capturing how quickly each instance executes the same scheduled replay workload, along with the CPU utilization and time required to complete this work.

How We Tested

Signal65 conducted all testing independently, on infrastructure we provisioned and controlled, using a reproduction harness developed by Signal65 and the replay file from the original test package, which we verified to be byte-identical (by MD5 checksum) to the source copy before any run. Testing used three persistent AWS instances in us-west-2, one per SKU. We used both smaller instances, known in AWS as 2xlarge (8 vCPU) size, and larger 16xlarge instances with 64 vCPUs.

Each instance ran the same software environment: Harbor 0.20.0, the Terminus-2 agent, and the local replay mechanism. The test matrix covered concurrency levels of 5, 10, 15, 20, and 25, with each test scheduling 20 trials (10 selected tasks × 2 attempts). The entire set of 27 tests was repeated three times, producing 81 measured results.

Reported figures are the median of the three per-run means. Every test completed 20 trials using the Harbor test harness. Our methodology relies upon three independent executions and then uses the median value rather than averaging, where an outlier could skew test results.

Generational Results: m8i (Xeon 6) vs. m7i (4th Gen Xeon)

At every tested concurrency, the 2xlarge m8i instances completed the work faster than m7i and at lower total cost.

Concurrency	m7i wall time (s)	m8i wall time (s)	m8i faster by	m7i test cost	m8i test cost	m8i cheaper by
5	1,832.5	1,691.5	+7.7%	\$0.2052	\$0.1989	+3.1%
10	1,271.5	1,131.5	+11.0%	\$0.1424	\$0.1331	+6.6%
15	1,091	1,011	+7.3%	\$0.1222	\$0.1189	+2.7%
20	1,068	971	+9.1%	\$0.1196	\$0.1142	+4.5%
25	1,048	976	+6.9%	\$0.1174	\$0.1148	+2.2%

Table 1: Intel generation comparison, m8i.2xlarge vs. m7i.2xlarge. Values are the median of three per-run means; positive percentages favor m8i. Costs are for the full 20-trial test.

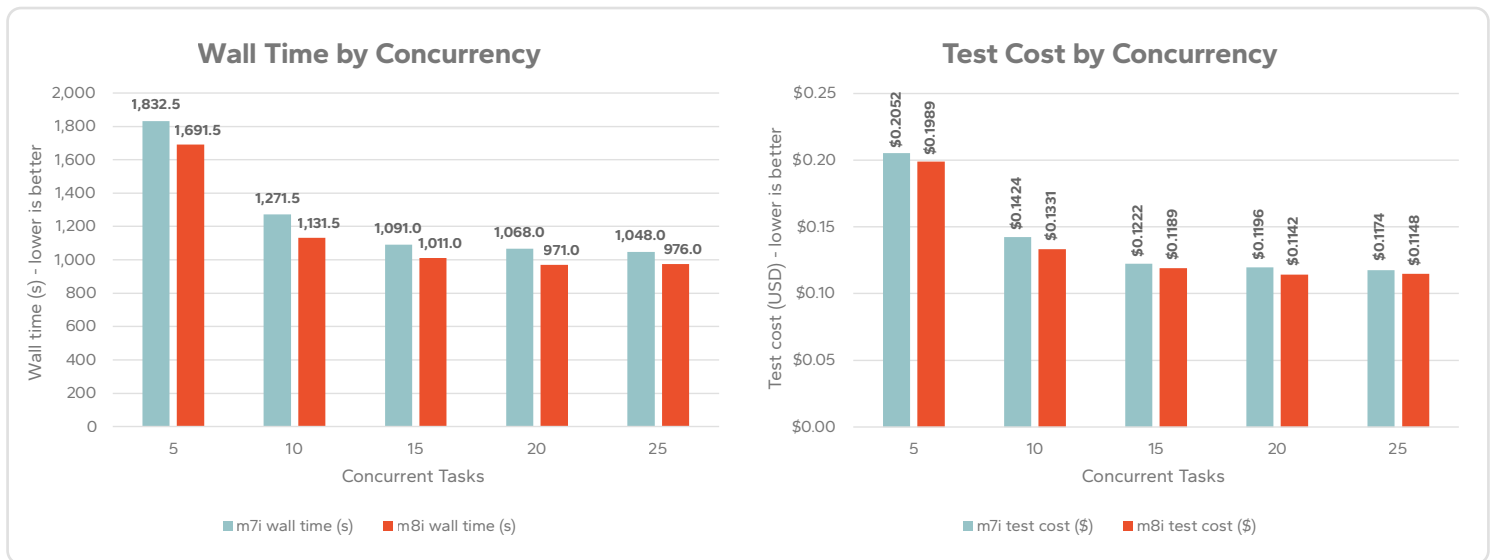


Figure 1: Values are 3-run medians from Table 1 (median of three per-run means). Test cost is the full 20-trial test at AWS on-demand rates (\$0.4234/hr m8i, \$0.4032/hr m7i).

As Table 1 and Figure 1 show, the largest measured advantage occurs at concurrency 10, where m8i is 11.0% faster and costs 6.6% less; at concurrency 20, m8i is 9.1% faster with 4.5% lower cost. Speed gains drop off sharply above about 20 concurrent processes, and we limited our range to 25 processes.

The tests show m8i’s performance gains make up for its 5% hourly-price premium at every tested concurrency. The conclusion is that m8i provides faster throughput and meaningful price-performance benefits throughout the tested workload range.

Competitive Context: Xeon 6 vs. AMD EPYC Turin

The 2xlarge m8i-versus-m8a comparison is mixed. On wall time, m8a is consistently faster for this workload, completing the 20-trial tests 8.3 – 15.8% faster than m8i under the reporting method used in the tables.

AWS prices m8a approximately 15% above m8i (\$0.4869 versus \$0.4234 per hour). The table therefore reports total compute cost for each 20-trial test alongside wall time:

Concurrency	m8i wall time (s)	m8a wall time (s)	m8a faster by	m8i test cost	m8a test cost	Test-cost result
5	1,691.5	1,483.5	12.3%	\$0.1989	\$0.2006	≈ parity
10	1,131.5	952.5	15.8%	\$0.1331	\$0.1288	m8a ~3% lower
15	1,011	869.5	14.0%	\$0.1189	\$0.1176	≈ parity
20	971	890.5	8.3%	\$0.1142	\$0.1204	m8i ~5.5% lower
25	976	850	12.9%	\$0.1148	\$0.1150	≈ parity

Table 2: Current-generation cross-vendor comparison, m8i.2xlarge vs. m8a.2xlarge. Values are the median of three per-run means. Costs are for the full 20-trial test.

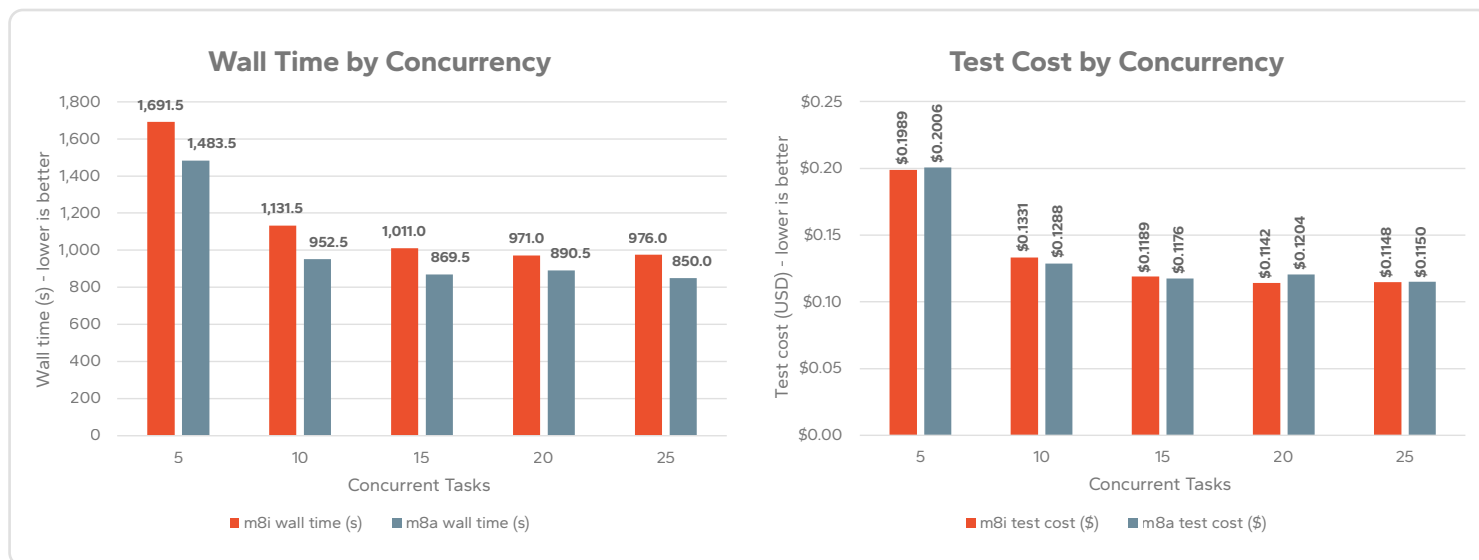


Figure 2: Values are 3-run medians from Table 2 (median of three per-run means). Test cost is the full 20-trial test at AWS on-demand rates (\$0.4234/hr m8i, \$0.4869/hr m8a).

As shown in Table 2 and Figure 2, m8a has 3.2% lower cost at concurrency 10 and 1.1% lower cost at concurrency 15. m8i has 5.5% lower cost at concurrency 20, with differences at concurrency 5 and 25 below 1%. These results do not identify a single recommended operating point. At a 15% hourly-price premium, m8a needs to finish approximately 13.0% sooner to break even, meaning that when evaluating price-performance,

there are minor advantages for m8i and m8a across the tested concurrency levels. As a cross-check, other ways of summarizing the data (medians of all tests combined and per-run means) give the same winner in every case, but narrow the concurrency-20 cost advantage to as little as 0.8% and the smallest generational cost advantage to approximately 0.1%.

Across the 2xlarge configurations, m8a shows higher performance, and thus lower latency. This benchmark does not test AMX-assisted models, small language models, or CPU-side inference.

Scale-Up Results

Testing larger 16xlarge instance sizes required modifying the test harness to increase parallelism. As stated previously, the default TB2 test configurations were limited to running a single container workload instance. By changing the test harness to enable using multiple containers, Signal65 was able to scale up the workload to allow more accurate testing and utilization rates of larger cloud instance types.

- **Low-concurrency tasks do not benefit from larger instances:** On the 64-vCPU instances, the single-process Harbor configuration running 20 trials averaged roughly 3 – 8% CPU utilization, leaving most of the system idle.
- **Increased parallelism with larger instance types:** Signal65 created a Python/SSH runner that launches 100 independent Harbor processes as one parallel batch – 10 selected TB2 tasks × 10 copies, with one trial per process. It kept the existing task definitions and replay setup, changing only how the work was launched and how much work was run. This raised reported utilization to ~51 – 57% on average, with peaks near 100%.
- **Economics:** To judge the potential amount of work larger instances were capable of, we had to improve the TB2 test harness to enable significantly higher concurrency. These results showed that on the same 16xlarge hardware, the improved harness delivered approximately 3× more throughput per dollar than the unaltered TB2 harness. Even so, the 16xlarge configurations returned roughly half the throughput per dollar of the 2xlarge instances at their best concurrency.

Considerations and Limitations

By providing the full set of results, Signal65 is being transparent about where Intel Xeon 6 instances have advantages, and where results are mixed.

Using the TB2 test replay harness enabled the test sequence to remain consistent and repeatable, providing a fair execution environment for tests. This includes agentic tool calls and a reproducible, consistent test environment. The tradeoff is that replay does not reproduce live, end-to-end agent workloads, where inference latency and model behavior re-enter the loop.

Three of the ten tasks have recorded trajectories that end mid-task and generate strict replay-cache misses. This happens the same way on all three SKUs. All 81 measured 2xlarge results therefore carry a Failed status under the harness's strict test guidelines, even though Harbor exited cleanly and all scheduled trials ran. The source-package baseline also identifies four never-passing tasks, including these three. Wall-time and test-cost comparisons remain complete-batch comparisons, not successful-task economics.

Pricing uses the AWS on-demand rates documented in the source package. The test-cost conclusions are specific to those rates. Reserved, Savings Plan, spot, and negotiated pricing (as well as other clouds' relative pricing) will change the arithmetic.

Determinism scope. Replay fixes the LLM-response sequence, but it does not eliminate kernel scheduling, container-runtime, filesystem, or external dependency variation. Results should therefore be read with the run-to-run variation and the cross-check results noted above in mind.

Metric scope. The cost conclusions in this report are based on batch wall-clock time and total compute cost for the scheduled replay workload under concurrent load. Per-task CPU-time measured in isolation is a different metric and ranks the instances differently: in both Signal65's measurements and the source package's per-task data, which agree closely in absolute terms, m8a consumes less CPU-time per task than m8i in isolation. Batch economics under concurrent load and isolated per-task efficiency answer different buyer questions; this report focuses on batch economics.

Scope. Results reflect one benchmark (TB2 replay), two instance sizes (2xlarge at 8 vCPU and 16xlarge at 64 vCPU), one region, and the specific agent stack described. The 16xlarge scale-up results use a separate multi-process execution method. Different task mixes, sizes, frameworks, or orchestration designs may change the size of the differences.

Conclusion: The Processor Under the Agents

Agentic AI changes the nature of tasks and shifts many tasks from being gated by GPU performance to being CPU-limited. When the workload is no longer a stream of tokens but a loop of decisions and actions, the CPU becomes an important component in the AI platform, controlling how fast agents act, how many tasks can be active, and what each unit of completed work costs.

Signal65's testing shows that Intel's current-generation Xeon 6 m8i instances are 7 – 11% faster than previous m7i instances and reduce costs by 2.2 – 6.6% across our tested concurrency levels. Comparing Intel m8i to AMD m8a, the two instances delivered similar price-performance.

The scale-up results add an important caveat. Larger systems can run substantially more concurrent agent work when orchestration supplies enough independent work, but the smaller-instance configurations in this dataset delivered better compute economics per scheduled trial.

For organizations planning agent deployments, the evidence supports three conclusions: m8i is a stronger choice than m7i; m8a and m8i deliver similar price-performance; and large instances need substantially more parallelism to achieve good price-performance, particularly compared to smaller instances for agentic tasks.

Appendix A: Methodology Detail

Test window and runs

All testing finished by September 8, 2026. We ran the full 2xlarge test set three separate times between July 31 and August 5, 2026, on the same three instances, without rebuilding the software between runs. The 16xlarge scale-up tests ran from August 28 to September 8, 2026. Each run had 27 tests, for 81 tests in total.

Test plan

At concurrency 5, 10, 15, and 20, each instance type ran two tests per run; at concurrency 25, it ran one. Each test scheduled 20 trials (10 selected TB2 tasks $\times$ 2 attempts). Within each run, we first averaged the two tests at the same concurrency, then reported the median of the three runs. As a cross-check, we also looked at the median and mean of all tests combined across runs.

Replay data and software

Before run 1, we used an MD5 checksum to confirm the replay file (978 entries) exactly matched the source copy. Each instance ran Harbor 0.20.0 and the Terminus-2 agent, with the same timeout fix used in the original testing. The replay proxy ran locally on each instance, with added delay turned off and strict mode on, so any request not found in the recording is flagged as a miss. Each instance passed a short check test before measurement began. The replay file was recorded with an older Harbor version than 0.20.0; this does not affect the cross-instance comparison, because the file matched the source exactly and strict mode flags any request that does not match the recording.

Metrics

Wall time is the total seconds needed to finish all 20 trials in a test. Test cost is wall time multiplied by the AWS on-demand hourly rate. CPU utilization is the instance's average CPU use over the run. Dividing test cost by 20 gives cost per scheduled trial, and the percent differences stay the same. Neither figure is cost per successfully completed task.

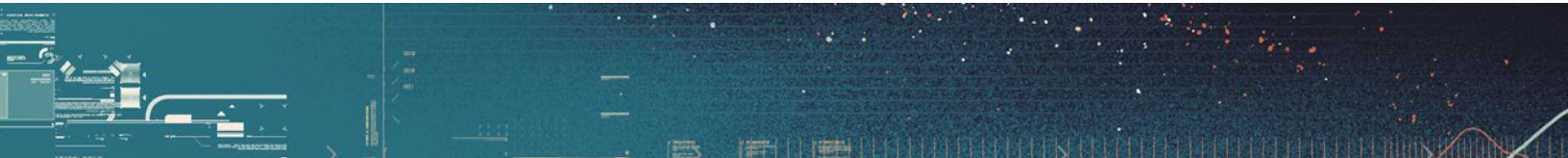
Scale-up harness

For the scale-up tests, we wrote a Python script that runs over SSH and starts 100 Harbor processes at once for each repetition: 10 selected tasks $\times$ 10 copies, with one trial per process. We kept the same task definitions, Terminus-2 timing fix, and replay proxy; only how the work was launched and how much work was run changed. The script recorded total batch time, CPU utilization, trial result counts, process exit codes, and replay-cache misses.

Scale-up test runs

We tested the same three instance types at the 16xlarge size (64 vCPU). Results are the median of three repetitions per instance type, each ending with a clean Harbor exit code. Some repetitions had to be rerun, with containers cleaned up in between, to get three valid results. Throughput is 100 scheduled trials divided by total batch time; it is not the rate of successfully completed tasks.

This comparison uses AWS compute cost and scheduled-trial throughput. Because the 2xlarge instances were tested on their own, the results compare small and large instances; they do not show how a cluster of small instances would perform.



Appendix B: Detailed Results

The tables in this appendix are calculated from the raw per-test measurements. They provide the per-run detail behind the summary figures in Tables 1 and 2, the 16xlarge scale-up measurements, and the utilization and per-vCPU comparisons referenced in the body of this report.

Instance	Vendor / CPU	vCPU	AWS On-Demand \$/hr
m8i.2xlarge	Intel Xeon 6 (Granite Rapids)	8	\$0.4234
m7i.2xlarge	4th Gen Intel Xeon (Sapphire Rapids)	8	\$0.4032
m8a.2xlarge	AMD EPYC Turin (Zen 5)	8	\$0.4869

Table 3: Instances under test. Hourly pricing as documented in the source benchmark package (AWS on-demand); see Considerations.

A note on the median rows. Throughout this appendix, a “3-run median” is the median of the three per-run values of the underlying quantity (wall time or cost), with the two tests at each concurrency averaged within each run first. The median percentage difference is then computed from those two median values. It is therefore not the median of the three per-run percentage differences, and the two can differ by a point or more when one run is an outlier.

B.1 Per-Run Detail: m8i.2xlarge vs. m7i.2xlarge

Run	Conc.	m7i wall (s)	m8i wall (s)	m8i faster by	m7i test cost	m8i test cost	m8i cheaper by
1	5	1,806.5	1,757.0	+2.7%	\$0.2023	\$0.2066	-2.1%
1	10	1,271.5	1,134.5	+10.8%	\$0.1424	\$0.1334	+6.3%
1	15	1,092.0	1,011.0	+7.4%	\$0.1223	\$0.1189	+2.8%
1	20	1,051.5	1,030.0	+2.0%	\$0.1178	\$0.1211	-2.9%
1	25	1,010.0	1,024.0	-1.4%	\$0.1131	\$0.1204	-6.5%
2	5	1,832.5	1,691.5	+7.7%	\$0.2052	\$0.1989	+3.1%
2	10	1,285.0	1,131.5	+11.9%	\$0.1439	\$0.1331	+7.5%

Run	Conc.	m7i wall (s)	m8i wall (s)	m8i faster by	m7i test cost	m8i test cost	m8i cheaper by
2	15	1,073.0	1,013.5	+5.5%	\$0.1202	\$0.1192	+0.8%
2	20	1,090.0	943.0	+13.5%	\$0.1221	\$0.1109	+9.2%
2	25	1,048.0	971.0	+7.3%	\$0.1174	\$0.1142	+2.7%
3	5	1,878.0	1,642.0	+12.6%	\$0.2103	\$0.1931	+8.2%
3	10	1,190.0	1,101.0	+7.5%	\$0.1333	\$0.1295	+2.9%
3	15	1,091.0	1,010.0	+7.4%	\$0.1222	\$0.1188	+2.8%
3	20	1,068.0	971.0	+9.1%	\$0.1196	\$0.1142	+4.5%
3	25	1,064.0	976.0	+8.3%	\$0.1192	\$0.1148	+3.7%
Median	5	1,832.5	1,691.5	+7.7%	\$0.2052	\$0.1989	+3.1%
Median	10	1,271.5	1,131.5	+11.0%	\$0.1424	\$0.1331	+6.6%
Median	15	1,091.0	1,011.0	+7.3%	\$0.1222	\$0.1189	+2.7%
Median	20	1,068.0	971.0	+9.1%	\$0.1196	\$0.1142	+4.5%
Median	25	1,048.0	976.0	+6.9%	\$0.1174	\$0.1148	+2.2%

Table 4: All three independent runs of the 2xlarge Intel generation comparison, plus the 3-run median rows that appear as Table 1 in the body. Positive percentages favor m8i. Concurrency 5 – 20 average two tests per run; concurrency 25 is a single test.

m8i wins 14 of the 15 per-run wall-time comparisons and 12 of 15 on test cost. The single wall-time exception is run 1 at concurrency 25, where m7i finished 1.4% sooner; the same run also produced the narrowest m8i margin, 2.0% at concurrency 20. Neither result repeated in runs 2 or 3. Outliers like the concurrency-25 result are why this report uses the median of three runs rather than a single run or a mean. The spread between runs is real: at concurrency 20, the per-run m8i wall-time advantage ranges from +2.0% to +13.5%, and at concurrency 5 from +2.7% to +12.6%.

B.2 Per-Run Detail: m8i.2xlarge vs. m8a.2xlarge

Run	Conc.	m8i wall (s)	m8a wall (s)	m8a faster by	m8i test cost	m8a test cost	m8a cheaper by
1	5	1,757.0	1,498.0	+14.7%	\$0.2066	\$0.2026	+1.9%
1	10	1,134.5	1,000.0	+11.9%	\$0.1334	\$0.1352	-1.4%
1	15	1,011.0	880.5	+12.9%	\$0.1189	\$0.1191	-0.2%
1	20	1,030.0	897.0	+12.9%	\$0.1211	\$0.1213	-0.2%
1	25	1,024.0	815.0	+20.4%	\$0.1204	\$0.1102	+8.5%
2	5	1,691.5	1,479.5	+12.5%	\$0.1989	\$0.2001	-0.6%
2	10	1,131.5	933.5	+17.5%	\$0.1331	\$0.1263	+5.1%

Run	Conc.	m8i wall (s)	m8a wall (s)	m8a faster by	m8i test cost	m8a test cost	m8a cheaper by
2	15	1,013.5	869.5	+14.2%	\$0.1192	\$0.1176	+1.3%
2	20	943.0	823.0	+12.7%	\$0.1109	\$0.1113	-0.4%
2	25	971.0	948.0	+2.4%	\$0.1142	\$0.1282	-12.3%
3	5	1,642.0	1,483.5	+9.7%	\$0.1931	\$0.2006	-3.9%
3	10	1,101.0	952.5	+13.5%	\$0.1295	\$0.1288	+0.5%
3	15	1,010.0	819.5	+18.9%	\$0.1188	\$0.1108	+6.7%
3	20	971.0	890.5	+8.3%	\$0.1142	\$0.1204	-5.5%
3	25	976.0	850.0	+12.9%	\$0.1148	\$0.1150	-0.2%
Median	5	1,691.5	1,483.5	+12.3%	\$0.1989	\$0.2006	-0.9%
Median	10	1,131.5	952.5	+15.8%	\$0.1331	\$0.1288	+3.2%
Median	15	1,011.0	869.5	+14.0%	\$0.1189	\$0.1176	+1.1%
Median	20	971.0	890.5	+8.3%	\$0.1142	\$0.1204	-5.5%
Median	25	976.0	850.0	+12.9%	\$0.1148	\$0.1150	-0.2%

Table 5: All three independent runs of the 2xlarge current-generation cross-vendor comparison, plus the 3-run median rows that appear as Table 2 in the body. Positive percentages favor m8a. Test costs use the documented AWS on-demand rates of \$0.4234/hr (m8i.2xlarge) and \$0.4869/hr (m8a.2xlarge).

The per-run detail shows how differently speed and cost behave. On wall time, m8a is faster in all 15 per-run comparisons, with a margin between +2.4% and +20.4%. On test cost, the result flips repeatedly: m8a is cheaper in 6 of 15 comparisons and m8i in 9, and the per-run spread at a single concurrency can exceed 20 points (concurrency 25 runs from +8.5% for m8a in run 1 to -12.3% in run 2). At a 15.0% hourly-price premium, m8a must finish about 13.0% sooner to break even, and the measured wall-time advantage falls on both sides of that line. This is the math behind the body's conclusion that the two SKUs are about even on price-performance for this workload despite a consistent latency advantage for m8a.

B.3 16xlarge Single-Process Results

The 16xlarge instances were first measured with the same unmodified concurrency 5 – 25 matrix used on the 2xlarge systems. Because Harbor's single-threaded orchestrator cannot load 64 vCPUs through that harness, these figures are included for completeness and should not be read as a full-load CPU comparison like the 2xlarge results. These results led to the multi-process design described in Appendix B.4.

Conc.	m7i wall (s)	m8i wall (s)	m8a wall (s)	m7i test cost	m8i test cost	m8a test cost
5	1,853.5	1,795.5	1,710.5	\$1.6607	\$1.6892	\$1.8507
10	1,109.0	1,070.0	1,022.5	\$0.9937	\$1.0067	\$1.1063
15	767.5	828.0	786.0	\$0.6877	\$0.7790	\$0.8504
20	769.0	834.5	818.0	\$0.6890	\$0.7851	\$0.8850
25	657.0	834.0	819.0	\$0.5887	\$0.7846	\$0.8861

Table 6: 16xlarge single-process results, 3-run medians. Same concurrency matrix as the 2xlarge tests, not scaled up for the larger instance. Test cost is wall time at the 16xlarge on-demand rate.

The rankings here are inconsistent and do not agree with the 2xlarge results: m7i posts the lowest wall time at concurrencies 15, 20, and 25, and the per-run values behind those medians swing widely (the m8i concurrency-25 test, for example, was measured at 956 s, 834 s, and 635 s across the three runs). With average CPU utilization of only 3 – 8%, these tests are measuring orchestrator overhead rather than processor throughput, which is why the report bases its scale-up conclusions on the multi-process results in Table 7 instead.

B.4 16xlarge Multi-Process (100-Process) Results

Metric (3-run median)	m7i.16xlarge	m8i.16xlarge	m8a.16xlarge
Elapsed wall time, 100-process job (s)	1,220	1,194	1,199
CPU, run average (%)	56.8	54.1	50.8
CPU, p95 (%)	85.8	82.7	78.8
CPU, peak (%)	99.9	100.0	98.7
Cost per 100-process job	\$1.0931	\$1.1233	\$1.2973
Seconds per scheduled trial	12.20	11.94	11.99
Cost per scheduled trial	\$0.01093	\$0.01123	\$0.01297

Table 7: 16xlarge multi-process design: 100 independent Harbor processes per batch (10 selected TB2 tasks × 10 copies, one trial per process), 3 repetitions per SKU, all with a clean Harbor exit and the full 810-cache-miss signature. Repetition 3 is the September 8, 2026 re-run that replaced the original disk-pressure-affected repetition.

Under the multi-process design the three SKUs finish within 2.2% of each other on elapsed time (1,194 – 1,220 s for the same 100 scheduled trials), and all three drive the CPU to a peak near 100%. The difference shows up in cost, where the m8a hourly premium is no longer offset by a wall-time advantage: m8a is 15.5% more expensive per batch than m8i while finishing 0.4% slower. Against m7i, m8i costs 2.8% more per batch while finishing 2.1% sooner. Average utilization of 50.8 – 56.8% also shows headroom remaining at 100 processes, so these figures show that the larger instances can be kept busy, not their maximum capacity.

B.5 CPU Utilization by Concurrency (2xlarge)

Concurrency	m7i CPU avg (%)	m8i CPU avg (%)	m8a CPU avg (%)
5	36.4	35.2	31.8
10	56.4	56.9	50.9
15	68.8	64.4	56.8
20	68.7	62.7	56.3
25	71.1	65.4	59.0

Table 8: Run-average CPU utilization on the 8-vCPU instances, 3-run medians, as reported by the harness.

Utilization roughly doubles across the tested concurrency range on all three SKUs and then flattens above concurrency 15, which is consistent with the diminishing wall-time gains observed at the top of the range. At every concurrency, m8a completes the same work at lower average utilization than either Intel SKU – 3 to 8 points below m8i – which matches its shorter wall times.

B.6 Per-vCPU Scaling: 2xlarge vs. 16xlarge

The final comparison adjusts for instance size. Each SKU is represented by its fastest single-process concurrency on 2xlarge – that is, its own best case rather than an arbitrary operating point – against its 16xlarge multi-process result from Table 7. Because AWS pricing scales directly with vCPU count within each SKU family, the cost ratio and the vCPU-hours ratio are identical.

SKU	2xl best conc.	2xl s/trial	2xl \$/trial	16xl s/trial	16xl \$/trial	cost ratio (16xl / 2xl)	2xl trials/hr/vCPU	16xl trials/hr/vCPU
m7i	25	52.40	\$0.00587	12.20	\$0.01093	1.86x	8.59	4.61
m8i	20	48.55	\$0.00571	11.94	\$0.01123	1.97x	9.27	4.71
m8a	25	42.50	\$0.00575	11.99	\$0.01297	2.26x	10.59	4.69

Table 9: Per-vCPU scaling. The 2xlarge side uses each SKU's fastest single-process concurrency from the 3-run medians in Tables 4 and 5; the 16xlarge side is the 100-process multi-process median from Table 7. Costs are per scheduled replay trial, not per successfully completed task.

Two results stand out. First, at its own best concurrency, each SKU lands within \$0.0002 per scheduled trial of the others on 2xlarge – \$0.00571 for m8i, \$0.00575 for m8a, \$0.00587 for m7i – a spread of under 3%, even though the underlying wall times differ by nearly 10 seconds per trial. Second, the 64-vCPU instances cost 1.86x to 2.26x more per scheduled trial than the 8-vCPU instances even with the improved multi-process harness, and deliver roughly half the per-vCPU throughput (4.6 – 4.7 trials/hr/vCPU versus 8.6 – 10.6). This is the measurement behind the report's conclusion that larger instances require substantially more parallelism than this workload supplied before their economics match those of smaller nodes.

Important Information About this Report

CONTRIBUTORS

Jonathan Fellows

Performance Validation Engineer | Signal65

Russ Fellows

VP, Labs | Signal65

PUBLISHER

Ryan Shrout

President and GM | Signal65

INQUIRIES

Contact us if you would like to discuss this report and Signal65 will respond promptly.

CITATIONS

This paper can be cited by accredited press and analysts, but must be cited in-context, displaying author's name, author's title, and "Signal65." Non-press and non-analysts must receive prior written permission by Signal65 for any citations.

LICENSING

This document, including any supporting materials, is owned by Signal65. This publication may not be reproduced, distributed, or shared in any form without the prior written permission of Signal65.

DISCLOSURES

Signal65 provides research, analysis, advising, and lab services to many high-tech companies, including those mentioned in this paper. Research of this document was commissioned by Intel Corporation.

ABOUT SIGNAL65

Signal65 exists to be a source of data in a world where technology markets and product landscapes create complex and distorted views of product truth. We strive to provide honest and comprehensive feedback and analysis for our clients in order for them to better understand their own competitive positioning and create optimal opportunities to market and message their devices and services.



IN PARTNERSHIP WITH



CONTACT INFORMATION

Signal65 | signal65.com