GPU SMITH
COMPUTE ENGINEERING

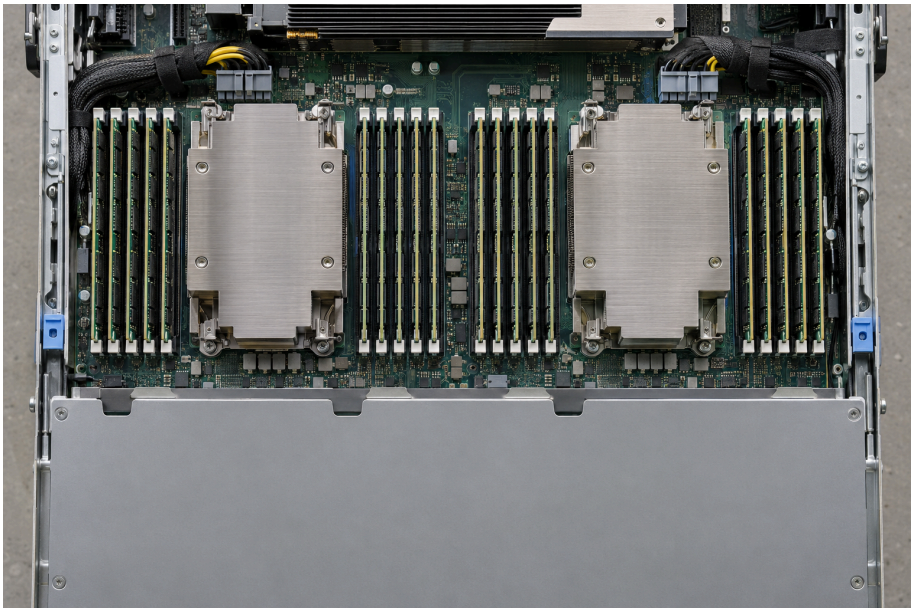
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

AMD PACE Agent Pilot Gate: Measure CPU Tool Work First

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-11-08 · amd pace agent pilot · amd pace · pace agent · langgraph deterministic replay · agentic ai cpu bottleneck · agent tool profiling · cpu gpu ratio planning · amd epyc inference · private ai infrastructure · pilot checklist

Original article: https://gpusmith.com/articles/en/amd-pace-agent-pilot-gate-cpu-tool-work



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes: What PACE Agent Adds
4. Implementation Considerations and Process Changes
5. Pilot Gate: Measurement Protocol and Decision Rules
6. Data Analysis and Evidence
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

On September 3, 2026, AMD announced that its **AMD PACE** project, previously an inference engine for **AMD EPYC** processors, had become an agentic orchestrator built on **LangGraph**, and AMD labels the project open-source research and development. The new **PACE Agent** package records an agent run, replays it, profiles registered non-model operations and can send model calls to local or remote OpenAI-compatible endpoints (github.com). This report converts that announcement into a pilot gate for private-AI platform engineers who must decide whether the package is mature enough for a laboratory trial and whether CPU-side work is worth optimizing before buying GPU capacity.

The maturity evidence is thin. The agent package declares version **0.1.0** (github.com), is installed from the repository and not from PyPI (github.com), and sits in a repository whose main branch showed **8 commits** as of November 8, 2026. The README says the built-in tools are synthetic CPU patterns, that unregistered work stays unattributed (github.com), and that `bench tools` does not suppress a speedup when parity fails. As of November 8, 2026, the public sources reviewed did not show an independent production evaluation of PACE Agent.

The performance evidence is workload dependent. AMD reports end-to-end gains of **1.02x to 2.20x** on WebVoyager, rising with page weight (www.amd.com). Back-calculating with Amdahl's law (www3.cs.stonybrook.edu), the sped-up operation occupied roughly 2% of baseline time on the lightest pages and roughly 62% on the heaviest, a derived figure and not an AMD measurement. Independent studies disagree: one reports tool processing up to **90.6%** of latency in a retrieval-heavy workflow (arxiv.org), while another reports tools at **2% to 29%** of runtime on coding and data benchmarks (arxiv.org). Tail shares rise sharply, from 32% at the median to 85% at the 99th percentile in one production-platform study driven by synthetic queries (arxiv.org).

The recommended gate has five parts. First, choose one bounded, non-sensitive workflow with behavioral and parity oracles. Second, treat the recorded trajectory as confidential, since the repository classifies it that way (github.com) and redaction is off by default (github.com). Third, build a baseline with characterization runs, warmups and interleaved samples (google.github.io), and report P95 and P99 only with enough samples. Fourth, accept an optimized candidate only after `optimize` passes parity and a live comparison confirms the projection, because the projection is not a measured full replay. Fifth, compute return on investment from measured or declared inputs: engineering time avoided plus capacity value, minus pilot, review and compute cost. If CPU tool work is small, drop the optimization; if it is large, there is a reproducible case for it before adding GPUs.

Introduction and Background

On September 3, 2026, AMD published a technical article stating that its **AMD PACE** (Platform Aware Compute Engine) project had moved beyond being an inference engine to become an agentic AI orchestrator built on LangGraph, and it labels the project open-source research and development (R&D) (www.amd.com). For a private-AI platform team that owns AMD EPYC or Instinct infrastructure, this raises two questions. Is the

new **PACE Agent** runtime mature enough for a laboratory pilot? And is the central-processing-unit (CPU) work around the model, meaning retrieval, serialization, context assembly, orchestration and tool input/output (I/O), a material bottleneck worth optimizing before anyone adds graphics-processing-unit (GPU) capacity?

This report answers the query "amd pace agent pilot" with a decision gate. It was prepared by GPU Smith, an independent private AI infrastructure engineering consultancy that advises on GPU procurement, workload sizing, inference optimization and air-gapped deployments. The report contains no private benchmark. Figures are drawn from the public sources cited inline, as reviewed on November 8, 2026, and derived figures are labeled as calculations.

Three things are kept apart here. PACE as a CPU inference engine carries AMD-attributed claims such as a 1.60x geometric-mean speedup over vLLM 0.17 (www.amd.com). PACE Agent, added on September 3, 2026 (github.com), is an orchestration and profiling layer. GPU procurement is a third matter that AMD's agent material does not try to settle. A pilot gate should judge the second item only, on the workload the organization actually runs.

The repository describes PACE Agent as a benchmark and optimizer for "the non-model layer" of agentic AI, covering five CPU buckets: retrieval, serialization, context, orchestration and tool-I/O (github.com). In this report, "CPU tool work" means those buckets, "model time" means time waiting on a model endpoint, and "unattributed" means CPU time outside any registered operation or supported serialization shim (github.com). The package metadata declares version 0.1.0, and the main branch listed eight commits in total, the latest on October 20, 2026 (github.com). That is a maturity signal the pilot decision should weigh.

Key Changes: What PACE Agent Adds

Change 1: a LangGraph runtime and profiler

The repository describes a LangGraph-based agent runtime for deterministic replay, profiling and optimization of non-model work (github.com). LangGraph models an agent as a graph in which "nodes do the work; edges decide what runs next" (docs.langchain.com), so an existing LangGraph codebase is the unit of adoption. The integration guide states that PACE Agent profiles the non-model layer of any compiled graph.

Change 2: record, replay and profiling

AMD says replay records "graph state, prompts, sampling parameters, tool inputs/outputs, and event ordering" and re-executes the graph against that trace (www.amd.com). The repository's format document is narrower: a trajectory holds tool calls with inputs, output and timing, the model tape, shim totals, run metrics and metadata (github.com). It does not describe attachments as separate items, so a pilot should open a real trajectory file instead of trusting either description. PACE Agent's replay also differs from LangGraph's time travel, which re-executes every node after a chosen checkpoint and skips only earlier nodes (docs.langchain.com).

Change 3: local or remote model calls

The repository states that PACE Agent supports local or remote OpenAI-compatible model endpoints (github.com), and vLLM documents a Chat Completions API in its OpenAI-compatible server (docs.vllm.ai). The agent README adds that the tool sends an unrecorded capability probe to choose between the

`/responses` and `/chat/completions` routes. Because the endpoint is a configuration value, the pilot question can be separated from the GPU question.

Change 4: parity-gated optimization with documented limits

The `optimize` command validates a full naive replay, re-times candidate-covered operations on recorded inputs and parity-checks every covered call. It reports an Amdahl-style projection and "does not measure a full candidate replay" (github.com). The built-in tools are "synthetic CPU patterns" and not production embedding, tokenization or reranking models, and `bench tools` "currently does not suppress a speedup or fail the command when parity is FAIL". These are documented properties of an R&D tool, and they drive the acceptance rules later in this report.

Implementation Considerations and Process Changes

Step 1: Decide which question the pilot answers

An agent run has four wall-time components: model time, registered non-model operations, unregistered non-model code, and waiting on external systems such as web pages. PACE Agent measures the second directly and the third only indirectly, since work outside registered operations "still shows up as node time and unattributed CPU" and is never guessed at (github.com). The pilot therefore answers one bounded question: for one workflow, how much wall time is registered CPU work, unattributed CPU work and model time? It does not settle GPU purchases, model quality or answer correctness. Treating a speedup on one operator as a statement about the whole system is the most common misreading of vendor material.

Step 2: Eligibility and dependency inventory

Table 1 consolidates the documented requirements into a ledger to complete before the first run.

Component	Documented value (as of November 8, 2026)	Pilot implication
PACE Agent package	Installed from the repository with pip install --no-build-isolation ./packages/pace_agent (github.com); the agent README says the package is not published on PyPI	Pin the commit hash, not a version number.
Base engine	Designed and tested for systems with AVX512 or higher support (github.com); AMD's datasheet cites full 512-bit data paths in the 9005 series core (www.amd.com)	Confirm AVX512 flags on the exact host.
PyTorch	The README pins a CPU build, torch==2.13.0+cpu	Needed for the base engine; verify whether the chosen workflow needs it.
Model endpoint	Any OpenAI-compatible endpoint	Choose it before baselining and keep it fixed.
Browser stack	Chrome for Testing or Chromium plus a matching driver; Chrome for Testing is a dedicated flavor without auto-update (developer.chrome.com), and Selenium Manager downloads and caches matching drivers (www.selenium.dev)	Skip unless production agents browse; live sites change between runs.

Component	Documented value (as of November 8, 2026)	Pilot implication
Code sandbox	The nsjail sandbox relies on Linux user namespaces, where a process can be unprivileged outside a namespace and privileged inside it (man7.org)	Needed only for code-execution agents; review privileges first.

A minimal pilot has few moving parts: a source install, a Python environment and a reachable endpoint. The browser and sandbox rows become real work only if the workflow needs them, and the browser row adds outbound traffic to driver and browser download hosts and to live sites that must be listed in the egress review. The sandbox row needs a security review, since seccomp system-call filtering is not itself a sandbox but a building block for one (www.kernel.org).

Step 3: Choose a bounded workflow and define oracles

Select one representative, bounded workflow free of sensitive data. The documentation warns that "an agent asked the same question twice does not do the same thing twice", and it calls a live-site browser agent the least repeatable workload the tool runs. Define two oracles before measuring. The **behavioral oracle** asks whether answers remain acceptable; PACE Agent can grade against an expected answer it withholds from the agent and reports strict and loose verdicts because "neither is quoted alone". The **parity oracle** asks whether an optimized tool returns the same output as the naive reference on recorded inputs. Because the built-ins are synthetic, spot-check outputs on the organization's own data types.

Public suites are small and concrete: WebVoyager has 643 tasks across 15 websites (arxiv.org), GAIA has 466 questions (arxiv.org), and the repository's GAIA web subset has 26 Level 1 and 64 Level 2 questions. They exercise the tool but do not proxy an internal workflow.

Step 4: Audit trajectory contents before recording

The trajectory format document classifies `trajectory.msgpack` as confidential and `report.md` and `result.json` as shareable. A redaction hook can scrub recorded values before persistence, and the default is none. The README calls a legacy inline `api_key` "accepted but unsafe" because the config is copied into the run directory (github.com). Independent guidance agrees on the direction: OWASP recommends least-privilege access (genai.owasp.org) and clear retention and deletion policies (genai.owasp.org), and the U.S. National Institute of Standards and Technology (NIST) advises minimizing collection and retention of personally identifiable information (PII) to what is strictly necessary (nvlpubs.nist.gov). *Table 2* is a starting classification matrix; the handling column is this report's recommendation, not a vendor requirement.

Artifact	What it can contain	Documented classification	Recommended pilot handling
trajectory.msgpack	Tool inputs, outputs, timing and the model tape	Confidential	Encrypted volume, stated retention date, fixtures only.
result.json, report.md	Aggregated results and rendered report	Shareable	Review for task text before sharing.

Artifact	What it can contain	Documented classification	Recommended pilot handling
trace.json, env.json, config.json, tasks.json	Per-run trace, environment data, config copy and tasks that ran	Not classified in the README	Treat as internal; strip endpoints and secrets.
Counter output	Cycles, IPC and cache metrics for a separate whole-process replay	Not classified	Low sensitivity; record host identity in the manifest.
LangGraph checkpoints (if already used)	A snapshot of graph state at each super-step (docs.langchain.com)	Encryption is optional (docs.langchain.com)	Separate pilot threads; plan pruning, since checkpoints accumulate (docs.langchain.com).

Classification follows what the workflow handles, not the file name, so synthetic or public fixtures remove most of the burden. Privacy guidance points the same way. NIST's AI Risk Management Framework says privacy values such as anonymity, confidentiality and control should guide design choices (nvlpubs.nist.gov), SP 800-122 describes de-identifying records when full records are not needed (nvlpubs.nist.gov), and OWASP lists tokenization to preprocess and sanitize sensitive information (genai.owasp.org). For the sandbox host, NIST's container guidance recommends validating that the runtime operates under least-privilege and least-access principles (nvlpubs.nist.gov) and logging activity to provide a clear audit trail (nvlpubs.nist.gov). At the end of the pilot, delete recordings under a written sanitization step; NIST defines sanitization as rendering access to target data infeasible for a given level of effort (nvlpubs.nist.gov). Serialization deserves one more note. PACE Agent uses a typed msgpack codec that fails loudly on unsupported types, and loading an older pickle file needs an explicit opt-in because unpickling executes arbitrary code, consistent with Python's instruction never to unpickle data that could have come from an untrusted source (docs.python.org). LangGraph's own checkpointer uses ormsgpack and JSON by default and documents an optional fallback to pickle for objects the msgpack encoder cannot handle (docs.langchain.com). Keep that opt-in off and treat any outside trajectory as untrusted.

Pilot Gate: Measurement Protocol and Decision Rules

Record a baseline that can be compared later

The baseline manifest matters more than its speed. Record the repository commit, Python and PyTorch versions, model identifier and endpoint, sampling settings, tool fixtures, task list, CPU model, core count, NUMA layout, operating system and kernel. AMD's own test notes show the expected detail: a dual-socket EPYC 9755 host with 1,536 GB of memory (24 DIMMs at 6,400 MT/s) (www.amd.com).

Sampling needs care because determinism at the endpoint belongs to the endpoint. vLLM does not guarantee reproducible results by default (docs.vllm.ai), and OpenAI describes its `seed` parameter as a best effort that does not guarantee determinism (developers.openai.com). Even vLLM's reproducibility settings apply only on the same hardware and the same vLLM version (docs.vllm.ai). Characterization addresses this: with `--`

`characterize-runs N` the tool runs the agent N times live and records a representative run (github.com). The cost is real, since characterization costs N live runs per task; for the 90-task GAIA web suite at N equal to 10 that is 900 browser sessions. A first pilot can use five runs on a handful of tasks.

The remaining protocol follows standard benchmarking practice. The tool's default is 20 interleaved samples after three warmups (github.com). Google's benchmark library explains why: warmup stabilizes caching effects (google.github.io), random interleaving reduces the effect of system-state changes (google.github.io), and pinning the process to a fixed CPU reduces variance (google.github.io). On a dual-socket host, pin workers within one NUMA node before comparing runs.

Measure registered work, unattributed work and model time

Build a time budget whose lines sum to end-to-end wall time.

- **Model time:** time inside endpoint calls, measured client-side and cross-checked against server metrics.
- **Registered CPU time:** time inside registered operations, per operation and per bucket.
- **Unattributed CPU time:** node time minus registered time, which the tool keeps visible.
- **Waiting:** network, disk and external service time, including page loads.
- **Residual:** wall time minus the other four lines; a large residual means missed instrumentation.

Hardware counters are optional and coarse. Telemetry is opt-in, and Linux perf can show a derived instructions-per-cycle metric (perfwiki.github.io), but its manual warns that measuring cycles and instructions outside a group can make the ratio less accurate (man7.org). Use counters to ask whether the whole process is stall-heavy, not which tool is slow. Repeat counter runs: `perf stat` can repeat a command and print the average and standard deviation, up to 100 repetitions (man7.org).

Instrument the pilot, then check the instrumentation

Cross-check PACE Agent's timings against an independent trace. OpenTelemetry defines a span as a unit of work or operation (opentelemetry.io) and allows spans to nest through a parent span ID (opentelemetry.io), which maps naturally onto graph nodes and tool calls. The W3C Trace Context recommendation defines a portable header describing a request's position in its trace graph (www.w3.org), which lets client-side spans be correlated with a remote endpoint's own logs. Instrumentation has a cost: Python's documentation says the pure-Python profiler is significantly slower than the C implementation and unsuitable for long-running profiling (docs.python.org), so use `time.perf_counter` for coarse timers (docs.python.org) and compare a profiled run with an unprofiled one. For NUMA placement, the Linux manual describes a local memory node that is faster to access than remote nodes (man7.org), and notes that forcing CPU affinity is useful only in certain applications (man7.org); pin only to remove a measured source of variance, and say so in the manifest. Google's benchmark guidance likewise says variance reduction is operating-system and architecture dependent (google.github.io).

Per-tool distributions and the tail

For each registered operation report n, median, P95 and P99 with a confidence interval. The tool declines to report P95 or P99 below 20 samples because a percentile from a handful of samples is not meaningful (github.com). NIST describes bootstrap resampling as typically using 500 to 1,000 subsamples (www.itl.nist.gov) and cautions that it is not appropriate for all distributions and statistics (www.itl.nist.gov), a reason to distrust extreme-tail intervals from small samples. The tail matters because rare latency episodes can come to dominate overall service performance at scale (research.google), and because steps compound: if each of 100 parallel servers has a one-second 99th-percentile latency, 63% of requests exceed one second (cacm.acm.org). An agent with dozens of sequential tool calls faces similar arithmetic. When generating load, measure from the scheduled start time, as the MLPerf Inference rules do (raw.githubusercontent.com). Rigor also matters for comparisons: a survey of Java benchmarking papers found that only 4 of 50 reported confidence intervals (dri.es). Report distributions, not means: Google's site reliability engineering book warns that designing around the mean of a quantity is tempting but hides the tail (sre.google), recommends bucketing request counts by latency to build a histogram (sre.google), and notes that the 99th percentile of one backend can become the median response of the frontend (sre.google). Performance data are rarely normal; a widely cited benchmarking study describes multi-modal distributions heavily skewed to the right (spcl.inf.ethz.ch) and tells analysts to check whether the mean or median is a useful summary (spcl.inf.ethz.ch). If several workers record latencies, Prometheus documentation warns that pre-computed quantiles cannot be aggregated across instances (prometheus.io) and that a histogram quantile's error is limited by the width of its bucket (prometheus.io). The HdrHistogram project offers a fixed-cost structure with built-in coordinated-omission correction (hdrhistogram.github.io). Repetition design matters as well: if measurements are not independent and identically distributed, variance and interval estimates are biased (kar.kent.ac.uk), and the original tail-latency paper concedes that eliminating all latency variability in large systems is impractical (cacm.acm.org).

Test candidates only when parity passes

`bench tools` is a screening instrument, not an acceptance gate: it does not suppress a speedup when parity fails (github.com), whereas `optimize` gates its projection on parity and treats zero provider coverage as invalid. A missing provider implementation raises a skip, never a silent Python fallback. An acceptance sequence follows.

1. **Screen with `bench tools`.** Discard any row whose parity is not PASS.
2. **Run `optimize` on the recorded baseline.** Require a valid replay contract, non-zero coverage and parity on every covered call.
3. **Read coverage before speedup.** A large operator gain on a small covered fraction yields a small end-to-end gain.
4. **Verify with a live run.** The projection is not a measured replay, so run the real graph with the candidate enabled on the same tasks, with interleaved repetitions.
5. **Re-run the behavioral oracle.** Accept only if scores hold within a tolerance declared beforehand.

Replay is strict by design: it is only trustworthy if it fails loudly. Browser agents cannot be re-timed from a tape, so browser pilots depend more on step 4.

Compare three options without crediting EPYC for everything

The decision reduces to three options:

- **Leave orchestration unchanged:** correct when registered plus unattributed CPU time is a small share of wall time, because Amdahl's limit caps any CPU optimization.
- **Optimize the CPU tools:** correct when that share is material, parity holds and the projected gain exceeds the cost of owning an R&D-status dependency.
- **Move or route model calls:** correct when model time dominates; the endpoint is then the variable to test.

Attribution keeps the comparison honest. A gain observed after changing the tool implementation, host, endpoint and sampling settings together cannot be assigned to any of them, so change one factor per experiment. AMD's footnotes state that results may vary based on configuration, usage, software version and optimizations (www.amd.com), and its engine and agent claims used different baselines and dates, so they should not be mixed.

Promotion, rollback and the reproducibility package

Promote from pilot to a longer evaluation only if every one of the following holds:

- **Behavior:** the behavioral oracle is unchanged within the declared tolerance.
- **Parity:** parity passes on every covered call.
- **Gain:** the live end-to-end gain has a confidence interval that excludes zero.
- **Attribution:** unattributed time is understood and the residual is small.
- **Handling:** the Table 2 rules were followed.

Rollback is simple: set the implementation back to naive, which restores the baseline. Archive the commit hash, lockfile, secret-free config, task list, fixtures, raw per-sample timings, reports and the decision memo. MLPerf's rules illustrate the standard: fixed random seeds (raw.githubusercontent.com) and a penalty that counteracts the higher variance of runs with few samples (raw.githubusercontent.com).

Data Analysis and Evidence

What AMD's agent numbers show

AMD's article reports one end-to-end result for the agent runtime: a WebVoyager evaluation with Qwen3-VL-8B-Instruct on a dual-socket EPYC 9755 server with a Radeon AI PRO R9700S GPU, giving end-to-end gains of 1.02x to 2.20x (www.amd.com). AMD defines page weight as the cost of observing a page, normalized to the lightest task (www.amd.com). *Table 3* reproduces AMD's published figures and adds a derived column that is this report's own calculation: if the whole end-to-end gain came from the sped-up operation, its share of baseline time is f = (1 - 1/E) / (1 - 1/s), where E is the end-to-end speedup and s the operator speedup.

Page weight class	Normalized page weight	Operator speedup (AMD)	End-to-end speedup (AMD)	Implied share of baseline time (derived)
Very light	1x	7.2x	1.02x	about 2.3%
Light	2.6x	8.8x	1.04x	about 4.3%
Medium	5.9x	10.0x	1.08x	about 8.2%
Heavy	15.8x	9.1x	1.21x	about 19.5%
Very heavy	68.2x	7.9x	2.20x	about 62.5%

Read this way, the data illustrate Amdahl's logic and not a general speedup. Operator gains of 7.2x to 10.0x produced end-to-end gains of 2% to 21% for four of five classes because the operation occupied roughly 2% to 20% of baseline time under the stated assumption. Only the very heavy class, where observation dominates, reaches 2.20x. The derived shares assume the operator speedup is the only source of improvement, so they are a consistency check and not measurements. Three conclusions follow. The "1.02x to 2.20x" claim is a range across page weights, and its low end sits within the noise a pilot must resolve. The result concerns serializing a page's accessibility tree, so it says little about retrieval or code execution. And the result comes from one vendor test on one configuration, so it needs reproduction on the target workflow.

What independent studies say about CPU work in agent runs

AMD's article is not independent evidence. Preprints on agent latency are the closest comparators, and they disagree by workload. *Table 4* summarizes five; each is an arXiv preprint used its own hardware and agents.

Study (arXiv)	Workload and setup	Reported tool or CPU share	Reported model share	Caveat
CPU-centric analysis, v2 (2511.00739)	Haystack RAG, Toolformer, ChemCrow, LangChain, SWE-Agent	Up to 90.6% of latency (arxiv.org); SWE-Agent Bash/Python 64.7%, 78.7%, 43.8%	LLM 0.5 s or less in the RAG case	Some workloads are CPU-heavy, others LLM-heavy (arxiv.org)
CPU-centric analysis, v3 (2511.00739)	Two hardware systems	Retrieval 83%, 81%, 82% in RAG (arxiv.org); SWE-Agent 38% and 25%, up to 65% on the faster GPU system (arxiv.org)	Toolformer LLM share falls from 88% to 77% (arxiv.org)	Numbers differ from v2; name the version
Agentic workload characteristics (2605.26297)	Five benchmarks including GAIA, two H100 GPUs	Tools 2% to 29% of runtime (arxiv.org)	LLM 71% to 98%	Tool latency is workload dependent (arxiv.org)
Sutradhara (2601.12967)	Synthetic queries on a production agent platform	Median 32% of first-token latency, 61% at P90, 85% at P99 (arxiv.org)	Remainder	Queries are synthetic

Study (arXiv)	Workload and setup	Reported tool or CPU share	Reported model share	Caveat
CPU sensitivity of agents (2609.19947)	Remote LLM APIs, local tool containers	Local retrieval QA 62% of average latency (arxiv.org)	Varies by task	More CPU cores do not always accelerate agents (arxiv.org)

The table supports no universal statement. Retrieval-heavy and code-execution agents show tool-dominated latency in some studies, and tool-calling workloads show model-dominated latency in others. One study reads its results as showing that a faster GPU system can shift the bottleneck to the CPU when tool latency is comparable to model latency (arxiv.org), which is the condition a pilot should test. The tail finding matters too: the tool share rises from 32% at the median to 85% at P99, so per-tool P99 belongs in the report.

Ratio heuristics and the ROI formula

A multi-GPU LLM serving study (arXiv 2603.22774) reports that adequate CPU resources reduced time-to-first-token by 1.47x to 7.11x without adding GPUs (arxiv.org) and gives 4 CPU cores per GPU as an example of the simple heuristics used for allocation; it also warns that ratios below 4-8 cores per GPU risk measurable slowdown for LLM inference workloads. That concerns model serving and not agent tools. No independent source found during research publishes a validated agent-specific CPU-to-GPU ratio, so the ratio is an output of the pilot, not an input.

The return on a pilot can be written before it starts:

ROI = engineering time avoided + capacity value, minus (pilot labor + review labor + compute cost)

Engineering time avoided is labor saved by replay in debugging and regression testing, estimated from defects reproduced per month and hours per defect. Capacity value is extra concurrent runs on the same hardware at the organization's own cost of capacity, credited only if the live gain has a confidence interval that excludes zero. Pilot labor covers source install, fixtures and characterization; review labor covers data classification and security review; compute cost includes live runs. Label declared inputs as assumptions and copy none from a vendor table.

Implications and Future Directions

CPU-to-GPU ratio planning and agentic workload sizing for private inference

Searches such as "agentic AI CPU GPU ratio" and "private inference CPU to GPU ratio planning" cannot be answered from public sources with one number. Two things are established. Host CPU headroom affects GPU serving: in the multi-GPU serving study, adequate CPU resources cut time-to-first-token by 1.47x to 7.11x. And agent-side tool work is a large share of latency in some workloads and a small one in others, with reported shares ranging from 2% to 90.6%. A sizing model should carry the tool-time share as a measured parameter. If registered plus unattributed CPU time is 10% of wall time, eliminating it entirely yields at most about 1.11x end to end; at 50% the ceiling is 2.0x. Both figures follow from Amdahl's law, and the original 1967 argument already treated a 40% housekeeping fraction as the limiter (www3.cs.stonybrook.edu).

The same arithmetic guards against over-buying. Faster GPUs shrink model time and raise the CPU share of what remains, so agent sizing should be reviewed whenever the model endpoint changes.

EPYC-based agentic inference servers: what is established

AMD positions some high-frequency EPYC 9005 products for CPU-based AI inference (www.amd.com), and two-socket servers offer up to 160 PCIe Gen 5 lanes (www.amd.com), which matters for attaching accelerators. The PACE vLLM plugin activates only on AMD EPYC with AVX512 support (www.amd.com), and AMD says it retains 92% to 98% of standalone PACE efficiency (www.amd.com). Those facts concern model serving on CPUs. They do not show that an EPYC host is the best place for an agent's tool layer, and the public sources reviewed did not show an independent production evaluation of PACE Agent. As of November 8, 2026, the evidence is vendor documentation, a vendor article and a repository with a short public history.

An on-premises agent pilot checklist

Each item can be marked done, not done or not applicable, with an owner and a date.

- **Scope:** one workflow, one endpoint, one host, one commit hash.
- **Eligibility:** AVX512 flags confirmed, source install of the agent package verified with `pace-agent check`.
- **Data:** fixtures only, or a documented classification and redaction hook.
- **Secrets:** keys supplied through an environment variable name, never inline.
- **Isolation:** if the agent executes code, nsjail is installed and its privileges reviewed.
- **Network:** egress list documented for the endpoint, dataset hosts and live sites.
- **Oracles:** behavioral tolerance and per-operation parity checks defined.
- **Baseline:** characterization verdict written into the memo.
- **Statistics:** at least 20 samples per tool, 100 where P99 is reported, with bootstrap intervals.
- **Attribution:** registered, unattributed, model, waiting and residual time in one waterfall.
- **Candidates:** parity gate passed, coverage reported, live comparison completed.
- **Rollback and package:** naive tier restorable in one change; manifest, raw samples and memo archived.

Agents that execute code deserve an explicit isolation review. User namespaces grant privileges only inside the namespace (man7.org), and system-call filtering is a building block for a sandbox and not a sandbox by itself (www.kernel.org), so review mounts, network policy and resource limits together. A security owner, not the benchmarking team, should sign off on these settings.

Signals that would raise confidence

The agent work is early-stage. Three signals would justify more than a laboratory pilot:

- **Releases:** tagged releases of the agent package with a change log.
- **Independent evidence:** measurements on workloads beyond the two benchmarks AMD used.
- **Data policy:** a documented policy for trajectory encryption and retention.

Until then, treat PACE Agent as a measurement instrument. A budget showing that CPU tool work is small is as valuable as one showing it is large, because it removes an optimization from the backlog. The decision rules above can be run by an internal performance team.

Frequently Asked Questions (FAQs)

What is the AMD PACE agent pilot, and who should run it?

It is a time-boxed laboratory test of PACE Agent on one bounded LangGraph workflow, for platform and performance teams that own AMD EPYC or Instinct infrastructure. Its output is a time budget and a go or no-go decision, not a purchase recommendation.

Is PACE Agent production ready?

AMD labels PACE an R&D project, the agent package is version 0.1.0, and it is installed from source. Those facts point to a laboratory pilot.

Is an agent workload CPU-bound or GPU-bound, and how can CPU tool work be measured?

It depends on the workload: reported tool shares run from 2% to 29% in one study and up to 90.6% in another. Profile a representative run, record registered time per bucket, compute unattributed time as node time minus registered time, add model and waiting time and check that the residual is small.

Why not rely on `bench tools` alone?

It does not fail when parity fails. Use `optimize` and a live comparison for acceptance.

What would invalidate a pilot result?

A diverged replay, zero coverage or failed parity; a changed endpoint or sampling setting between baseline and candidate; fewer than 20 samples for a percentile claim; and unlabeled assumptions in the ROI.

Conclusion

PACE Agent gives an AMD EPYC owner a LangGraph-compatible way to record a run, replay its CPU-side operations, profile registered work and test parity-checked candidates. On the public record reviewed on November 8, 2026, it does not prove that CPU tool work is a bottleneck for any given organization. The vendor's evaluation shows end-to-end gains that depend heavily on page weight, and independent studies disagree about whether tools or models dominate agent latency.

The pilot gate keeps the decision where the evidence is. Define the workflow and oracles first. Audit what the recording will contain. Build a baseline with a manifest, characterization and interleaved repetitions. Report registered, unattributed, model and waiting time with per-tool P50, P95 and P99 and confidence intervals. Accept a candidate only when parity passes and a live comparison confirms the projection. If CPU tool work is small, the pilot has still removed an uncertainty from capacity planning. If it is large, the organization will hold a measured, reproducible case for optimizing it before adding GPU capacity.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://github.com/amd/AMD-PACE
2. https://github.com/amd/AMD-PACE/blob/main/packages/pace_agent/pyproject.toml
3. https://github.com/amd/AMD-PACE/blob/main/packages/pace_agent/README.md
4. https://www.amd.com/en/developer/resources/technical-articles/2026/agentic-flow-with-amd-pace.html
5. https://www3.cs.stonybrook.edu/~rezaul/Spring-2019/CSE613/reading/Amdahl-1967.pdf
6. https://arxiv.org/html/2511.00739v2
7. https://arxiv.org/html/2605.26297
8. https://arxiv.org/html/2601.12967v3
9. https://github.com/amd/AMD-PACE/blob/main/packages/pace_agent/docs/trajectory-format.md
10. https://google.github.io/benchmark/user_guide.html
11. https://www.amd.com/en/developer/resources/technical-articles/2026/amd-pace---high-performance-platform-aware-compute-engine.html
12. https://github.com/amd/AMD-PACE/commits/main
13. https://docs.langchain.com/oss/python/langgraph/graph-api
14. https://docs.langchain.com/oss/python/langgraph/use-time-travel
15. https://docs.vllm.ai/en/latest/serving/online_serving/openai_compatible_server
16. https://github.com/amd/AMD-PACE/blob/main/packages/pace_agent/docs/agent-integration.md
17. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/datasheets/amd-epyc-9005-series-processor-datasheet.pdf
18. https://developer.chrome.com/blog/chrome-for-testing
19. https://www.selenium.dev/documentation/selenium_manager
20. https://man7.org/linux/man-pages/man7/user_namespaces.7.html
21. https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html
22. https://arxiv.org/html/2401.13919v4
23. https://arxiv.org/abs/2311.12983
24. https://genai.owasp.org/llmrisk/llm022025-sensitive-information-disclosure
25. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf
26. https://docs.langchain.com/oss/python/langgraph/checkpointers
27. https://docs.langchain.com/oss/python/langgraph/persistence
28. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf
29. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf
30. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-88r2.pdf
31. https://docs.python.org/3/library/pickle.html
32. https://docs.vllm.ai/en/latest/usage/reproducibility
33. https://developers.openai.com/api/reference/python/resources/chat/subresources/completions/methods/create

34. https://google.github.io/benchmark/reducing_variance.html
35. https://perfwiki.github.io/main/tutorial
36. https://man7.org/linux/man-pages/man1/perf-stat.1.html
37. https://opentelemetry.io/docs/concepts/signals/traces
38. https://www.w3.org/TR/trace-context-1
39. https://docs.python.org/3/library/profile.html
40. https://docs.python.org/3/library/time.html
41. https://man7.org/linux/man-pages/man7/numa.7.html
42. https://man7.org/linux/man-pages/man1/taskset.1.html
43. https://github.com/amd/AMD-PACE/blob/main/packages/pace_agent/docs/benchmark-methodology.md
44. https://www.itl.nist.gov/div898/handbook/eda/section3/bootplot.htm
45. https://research.google/pubs/pub40801
46. https://cacm.acm.org/research/the-tail-at-scale
47. https://raw.githubusercontent.com/mlcommons/inference_policies/master/inference_rules.adoc
48. https://dri.es/files/oopsla07-georges.pdf
49. https://sre.google/sre-book/monitoring-distributed-systems
50. https://spcl.inf.ethz.ch/Publications/.pdf/hoefler-scientific-benchmarking.pdf
51. https://prometheus.io/docs/practices/histograms
52. https://hdrhistogram.github.io/HdrHistogram
53. https://kar.kent.ac.uk/id/document/3127886
54. https://arxiv.org/html/2511.00739v3
55. https://arxiv.org/html/2609.19947v2
56. https://arxiv.org/abs/2609.19947
57. https://arxiv.org/abs/2603.22774
58. https://www.amd.com/en/developer/resources/technical-articles/2026/amd-pace-integrates-with-vllm.html

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.