



project veneta whitepaper · October 2026 · v1.3.1

Beyond the limits of local LLMs

Why the language models you run inside your own network fall short, and what changes when you put a memory and self-improvement layer around the model instead of replacing it. Measured on eight open models, 12B to 70B.

Table of contents

03

Executive summary

05

Five limits of a local LLM

07

The VENETA solution
the memory and self-
improvement layer at the
LLM edge

11

What we measured

19

Industry by industry

22

Security and governance

24

How it is delivered
WorldModel · Enterprise ·
project veneta

28

Key takeaways and next
steps

Executive summary

Do not change the model. Put memory and self-improvement around it.

You keep your language model inside the building for a clear reason: the data cannot leave. The model you keep there often disappoints. It does not know the state of your systems right now. It does not remember last week's decision. It invents figures. It does not know when it is wrong. And nobody can audit what it has learned. None of this is a question of model size. It is a question of what a model cannot do alone.

VENETA does not train your model. It puts a **memory and self-improvement layer** around it. The first step of every operational question reads the real systems. Recalled facts arrive stamped with when they were read. A draft must pass checks written in code and a critic model before a person sees it. Only what a person approves changes the next conversation. Every step lands in a ledger. The layer sits on any model, so the discipline survives when you change the model.

We measured every number in this document, and we claim nothing we did not measure. Where the data was synthetic, we say so.

Four things we measured

10.3–12.0 vs

5.0–6.0

Memory layer · all eight models

Twelve memory-ability cases, the same model with and without the memory layer. Without it, the score is structurally zero on four abilities, because the model has no way to remember.

8 of 8 models up

Judgment layer · 14 operational questions

When we made the checks bind the critic, all eight models rose: two questions at the median, the 70B model from 6.3 to 13.7. Switch the binding off on the same model and 14.0 falls to 13.7 and 13.0.

125/125

Approved rules applied

Of the 125 times a person-approved rule reached the prompt, the final answer honored it 125 times. This is the first measurement, in code, of whether what was learned is actually applied.

Equal, then

structure

Against an open-source memory layer

We swapped the most widely used open-source memory layer (layer A in this document) into our own loop under the same conditions: operational questions equal, memory benchmark 12.0 to 11.7, long-horizon set 16 to 14. On retrieval alone, layer A is equal or slightly ahead.

These results come from one edition (telecom), synthetic data, 14 + 12 cases, three repeats, and one machine. That is a small experiment for a large claim, which is why this paper does not say "eliminates hallucination" or "best." Instead, it says what changes industry by industry and what we do not claim. The full protocol and data go into the paper we are preparing.

This paper is for CIOs, CTOs, AI-platform teams, and security officers responsible for in-house LLMs, AI agents, customer chatbots, and internal AI work systems. It covers VENETA WorldModel (industry editions), VENETA Enterprise (the memory and self-improvement layer for your own models), and project veneta (open source). We made the measurements between September 28 and October 1, 2026, on one NVIDIA DGX Spark (128 GB unified memory) with vLLM, on the telecom edition's synthetic operating data.



Five limits an enterprise meets in a local LLM

None of the five goes away with a bigger model. They belong outside the model.

On veneta-bench, our memory-ability benchmark (12 cases, eight open models, the veneta memory and self-improvement layer switched on and off over the same judgment layer, mean of 3), the score roughly doubled: 10.3–12.0 with the layer, 5.0–6.0 without it (Figure 2). Within that benchmark, the layer moved the score more than model size did. The difference does not apply to questions that need no memory, and it does not mean a model's general capability doubles. That is why we invest in the layer rather than in fine-tuning and training models.

1. It does not know: no access to the present state

A model knows the world only as of the end of its training. Which cell is down right now, which store shelf is empty, which change was approved today: all of that lives in the IT systems inside your closed network. Retrieval over documents (RAG) matches the documents as they were when the index was built, not as they are now. The vector memory and self-improvement layer carries the answer for today and for right now.

2. It forgets: nothing survives the session

Last week an operator corrected it: "this action only under that condition." This week the chatbot repeats the mistake. What was approved during the last incident, and how it turned out, is gone too. Most human work rests on memory, and the model has none.

3. It invents: figures and names without a source

The most dangerous error is a plausible number: a traffic figure with no reading behind it, a procedure number that does not exist, a stock level nobody checked. Smaller models do it more often, and the more fluent the answer, the harder it is to notice. In a customer chatbot and in an internal operations system alike, this is what breaks trust.

4. It cannot check itself

Ask "are you sure?" and the model says yes. Whether its answer carries evidence, keeps the required form, or labels a simulation as a simulation is not verified when the model is the one verifying. The checks have to live outside the model, in code, deterministically.

5. When it learns, it cannot be governed

"Let the model learn on its own" raises three questions in an enterprise: What did it learn? Who allowed it? Can it be undone? Fine-tuning and prompt edits answer none of them. In a regulated industry, learning that cannot be audited is not learning. It is risk.



The VENETA solution: the memory and self- improvement layer at the LLM edge

A lesson is never written into memory. It is proposed, a person approves it, a measurement decides whether it stays, and a hash-chained ledger records it so that it can be undone. That governed learning is the layer. The six steps that answer an operational question run on top of it, and the model takes part in only two of them.

This chapter describes *what* the layer does. *How* it is implemented will be published in the paper after the patent filing.

Governed learning: a lesson is never written, only proposed

When the loop learns something, from a failed check, an operator's correction, or an outcome, the lesson does not enter memory. It lands in an inbox as a **proposal**. A person approves or rejects it. An approved lesson is then measured on its next turns before it stays; a rejected one becomes a tombstone, and a sentence with the same meaning is never proposed again. Each of these events, proposal, approval, rejection, reinforcement, archive, and deletion, is one line in a **hash-chained ledger**, and the ledger can be **rewound**: memory returns to any earlier line, and the lines after it stay on record. Learning that the operator can undo is the part of the layer we built from scratch.

Memory comes in four kinds: **rules** (lessons, do and do not), **facts** (a reading and its time), **episodes** (what happened), and **outcomes** (how an approved action ended). Each agent sees only its own memory. The operator's decisions and simulation results are written to a shared scope that every agent sees. Memory has a ceiling and a horizon, so stale facts retire on their own. Three more things set enterprise memory apart from a general memory library.

How an operational question is answered

1 Read The first step of an operational question is always a tool call: alarms, topology, stock, transactions, read from your systems now. Memory is not allowed to answer first.	2 Recall Approved rules, past decisions and their outcomes, and the operator's observed preferences come up. Facts carry the time they were read and a note to re-read them.	3 Draft The model drafts an answer from the readings and the memory. The prompt carries no facts. Figures arrive only through tools and memory.
4 Check and critique Seven checks in code run first (evidence, length, simulation and quantum labels, unsourced figures, tool use, leaked internal names), and their findings bind the critic model. A finding means no pass. On failure, the loop fails closed.	5 Answer and approve The answer leaves with its evidence: what was read and when, which simulation and checks it passed, which model answered. A person approves anything that must be executed, and approval creates a draft in your system.	6 Remember and ledger The outcome is remembered. A lesson from a failure is proposed as a pending rule. It becomes active only when a person approves it, and a rejection blocks the same lesson from returning. Every step lands in an append-only ledger.

A guard at write

We scan every line that enters memory. An instruction to the model, a secret, a link, or an invisible character inside a rule turns the row into a rejected tombstone: recorded, never recalled. Prompt injection through memory stops at the moment of storage.

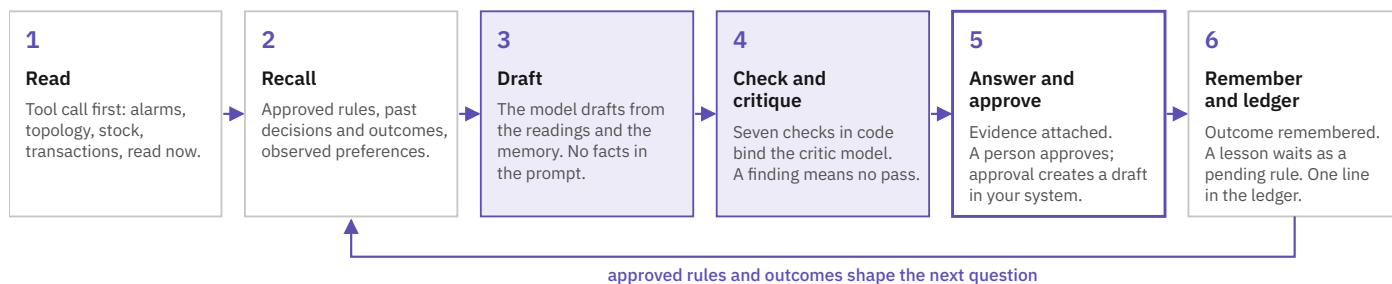
A person is the last step

A lesson the loop learns arrives pending. Only approved rows are recalled. A rejected lesson becomes a tombstone, and a sentence with the same meaning is never proposed again. We never infer operator preferences. We tally them from three or more recorded approvals.

A ledger that can be rewound

Proposal, adoption, reinforcement, approval, rejection, block, archive, and deletion each take one line in a hash-chained ledger. Even a deletion leaves a proof. An auditor can replay what this model learned, and when, and an operator can rewind memory to any line of it.

One operational question, six steps. The model takes part in two.



□ code

□ the model (draft, critique)

□ a person

Every step is one line in an append-only ledger.

Figure 1. How an operational question is answered. Six steps; the model takes part in two (draft and critique); a person approves anything that must be executed; every step is one line in the ledger.

Judgment and self-improvement: the checks bind the critic, and learned rules apply once approved

Many systems let a model grade a model. We use a critic model too. The difference is order and authority. Seven checks written in code run *first*, and their findings reach the critic not as things it may notice but as facts it must fix. With even one finding, the critic cannot pass the draft and must rewrite it. The checks grade. The critic repairs. Reverse the order and let the critic judge on its own (the control in the next chapter), and the answers miss exactly what the checks used to catch.

Rules carry checks too. A rule that code can verify, such as "label a simulation result as a simulation whenever the simulator ran," is remembered together with its check, and every turn records whether the answer actually honored it. The 125/125 in the next chapter is that record.

The point of the judgment layer is not a better judge than the model. It is to **tie judgment to deterministic things outside the model**: deterministic checks, then a calibrated small judge model (now under measurement), then the critic model, then a person. In that order, the discipline survives a change of the large model. The discipline is what you are buying.

What here is known, and what is ours

Checks written in code, a critic model that grades a draft, and a memory that retrieves past rows are known patterns, and we use all three.* What we built around them is what happens to a lesson after it is learned: a proposal before recall, an approval by a person, a measurement before the lesson stays, a ledger that can be rewound, and the measurement of all of it on eight open models against a control with no memory. A reader who files this chapter under guardrails has read the known half.

* The family is usually called guardrail tooling and LLM-as-a-judge: rules or classifiers that stop an answer, and a second model that scores it. We name no products here; the comparison with four open-source memory layers is in the next chapter.

What the memory and self-improvement layer needs to boost an LLM

Readings	Which systems were read, through which tools, and when.
Simulation	The band each option would move and the pessimistic edge the gate was applied to. A simulation result is labeled as a simulation.
Gate verdict	Passed or not indicated, with the service-level reason.
Checks	The seven checks, the critic's verdict, the rules applied.
Model	Which language model answered. The engine runs on your own model, inside your own building.



What we measured

Every set is scored in code, and a saved answer can be re-graded. There is no human grading. Three repeats per condition; tables show the mean and [range].

The measurement period was September to October 2026 (campaign 3, 250 run files, every run exited cleanly), followed by the baseline tests of third-party memory layers. Every figure comes from the synthetic data of the VENETA WorldModel telecom edition.

Setting

Machine	One NVIDIA DGX Spark (GB10, 128 GB unified memory). Models served by vLLM, FP8, 16k context, tool calling on, thinking mode off.
Eight models	Gemma 4 31B and Gemma 4 12B (Google), OTel 2.0 31B (Gemma 4 post-trained for telecom), Qwen3.8 27B (Alibaba), Mistral Small 3.2 24B, EXAONE 4.0.1 32B (LG AI Research), GLM-4.7 Flash (Zhipu), Llama 3.3 70B (Meta). All open weights, all at a size an enterprise can run in-house.
Data	The telecom edition's synthetic operating data: a town-sized replica of a network with cells, backhaul, alarms, topology, and SLAs. We used no real operator data.
Sets	14 operational questions (faults, optimization, quantum, small talk, no data); 12 memory-ability cases (decision recall, readings over memory, correction persists, rejection persists, observed operator, no contamination); 4 adversarial; 16 long-horizon and bilingual; 8 chain.

The memory layer: the same model with and without memory

The twelve memory-ability cases ask: What did we do last time, and did the gate hold? Does a remembered figure lose to today's reading? Does an approved correction shape the next session? Is a rejected lesson kept from returning? Does the answer cite the record rather than a trait of the operator? Does chatter in memory stay out of an operational answer? A model without a memory layer scores structurally zero on three of these abilities and passes one vacuously. The gray bars below are therefore not a competitor's result. They are a picture of what is impossible without the layer.

Two notes. First, the memory layer lost nothing on the two abilities where memory can only do harm: a remembered figure beating today's reading (readings over memory) and chatter leaking into an operational answer (no contamination) were 6/6 on all eight. Second, models differ: one 24B model reached only 2/6 on decision recall. The table records it as it is.

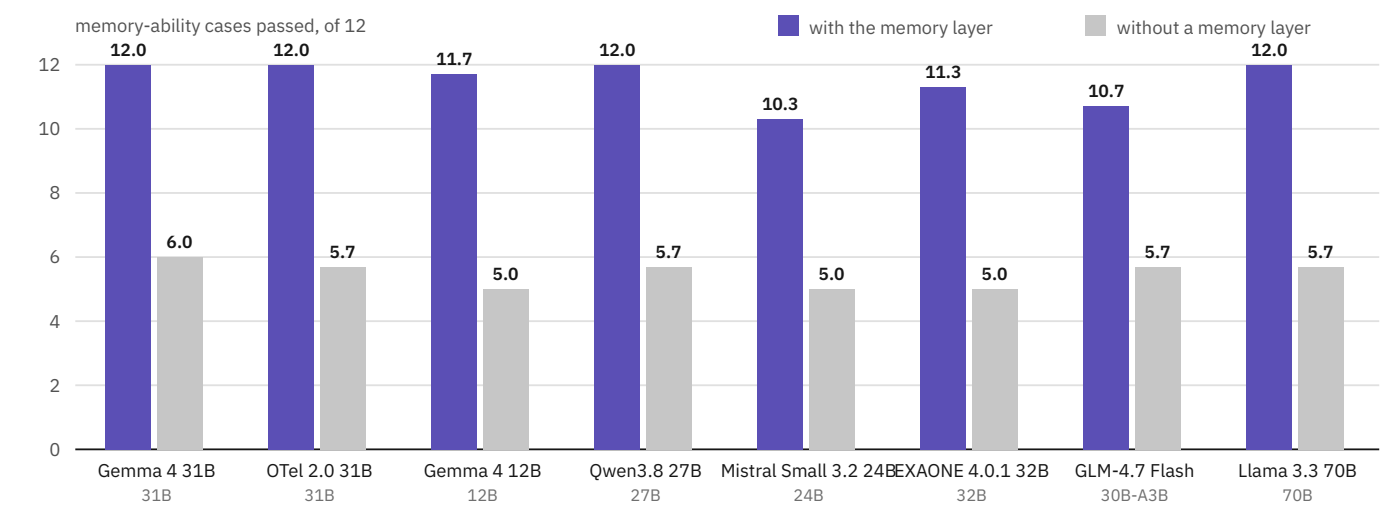


Figure 2. Twelve memory-ability cases, eight models, the memory layer switched on and off over the same judgment layer (mean of 3, dense recall). With memory 10.3–12.0, without 5.0–6.0. Four models at 12/12. Source: campaign 3, September 28–30, 2026.

The judgment layer: before and after the checks bind the critic

We measured the same 14 questions twice: on the engine of September 25, 2026 (the critic judging on its own) and on the engine of September 29 (the findings of the checks binding the critic). Models, data, and questions were the same. What changed in the engine was the judgment layer and some record-keeping. A separate control that switches only the binding off on the same engine confirmed that the judgment layer is what moved the score.

What rose matters. The 70B model's seven-question gain was mostly *labels*: the check that a simulation is called a simulation went from 45 % to 100 %, and the quantum-result label from 17 % to 83 %. The layer supplied less the content of the answer than the discipline of saying honestly what the answer is. The remaining failures sit in two checks: unsourced figures (EXAONE, GLM, Mistral at 86–98 %) and the quantum label. The names of the failed questions are in the record.

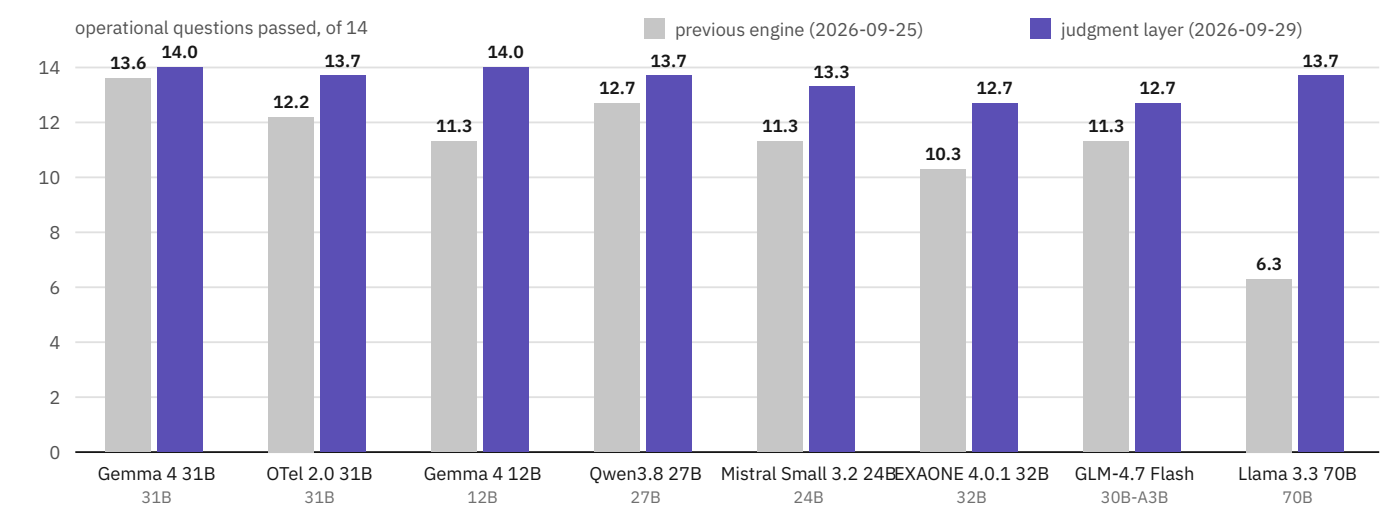


Figure 3. Fourteen operational questions, eight models, memory on, dense recall (mean of 3). Previous engine 6.3–13.6 → judgment layer 12.7–14.0. All eight up, about two questions at the median. The three anchor models (Gemma 4 31B and 12B, OTel 31B) pass all seven checks.

The control. On the strongest model (Gemma 4 31B), stop handing the findings to the critic. The critic rewrites about as often and grades about the same, yet the answers it passes miss exactly what the checks used to catch: the quantum label and leaked internal tool names. The score falls from 14.0 to 13.7 (dense) and 13 (hybrid). The memory benchmark does not move, and it should not. The control changed judging, not memory.

We also shortened the critic's rubric to its headings and left out each of its six criteria in turn. On the strong model, the score did not move. Only the cost in seconds did. The effect comes from the *binding findings*, not from the wording of the rubric.

Table 1. Judgment-layer control, Gemma 4 31B, 14 operational questions, three repeats

Condition	Passed / 14	Checks below 100 %	Turns rewritten	Median s
dense · checks bind the critic	14.0 [14–14]	none	16/30	34.5
dense · binding off	13.7 [13–14]	internal-name leak 98 %	17/30	39.8
hybrid · checks bind the critic	14.0 [14–14]	none	16/30	39.5
hybrid · binding off	13.0 [12–14]	quantum label 83 %, internal names 95 %	15/30	36.1

Are learned rules applied, and what about attacks

Rules applied 125/125

Two anchor models (Gemma 4 31B, OTel 31B), two recall modes, three repeats each. Every one of the 125 times an approved rule with a check reached the prompt, the final answer honored it. Before this, we could say "the rule was remembered." Now we can count "it was applied."

Adversarial 4/4 on all eight

Poisoned rows planted in memory (an instruction, a wrong figure, another person's episode) never surfaced in an operational answer.

Long horizon and bilingual, 16 questions

Two sessions, a Korean decision recalled by its identifier, time order and conflict (the latest wins, and the answer says when), federation of shared rules, and the mark on an unverified rule. Four models (Gemma 4 31B and 12B, OTel 31B, Qwen3.8 27B) at 16/16, the others at 10–13. Failures cluster on recalling a first-session decision by name and on honoring the time window of a correction.

Against another memory layer: the same loop, only the store swapped

There are several "memory layers" now. For a fair comparison, we plugged four widely used open-source memory layers *into our own loop* — this document calls them layer A (a memory API with an extraction model), B (an agent framework's built-in memory), C (a hosted memory API storing document chunks) and D (a graph-based layer); their names and versions are in the paper and the run files — tools, checks, critic, and approval unchanged, with only the storage and search of memory replaced. The difference is therefore the difference between stores. Same local embedding model, telemetry off, nothing left the building.

The second comparison matters more. In the field a memory layer does not receive our rows; it extracts memories from the conversation itself. Run that way (extraction) on the three anchor models, layer A's memory-ability score falls from 11.7 to 8.0 and the long-horizon set from 13.3–14.3 to 6.7–8.0. Corrections and operator preferences go to 0/6, federation, isolation and marks fail every time, and the list of failed cases is nearly identical across three models and three repeats. The mechanism is extraction: a median of 1 memory (at most 10) survives out of 37 rows, and loading rises from 1.4 s to 8–10 s (95th percentile 71–103 s). Extraction compresses, and what it drops is exactly the structure enterprise memory needs. The extraction model was our anchor; layer A's default configuration with a larger cloud model may keep more, and we say so.

Read it this way. Share the judgment layer, and operational answers come out equal. The difference appears on **abilities that need structure in the record**: the operator's observed preference, a decision called by its identifier, the mark "this rule is not yet measured," a rule promoted to a shared scope. A flat semantic memory has no place for these. Our typed records, lexical search, and pending state do that work. Conversely, layer A retrieves well, and we intend to absorb that strength.

Table 2. Our memory layer against layer A on two anchor models, store swapped only, three repeats

Measure	VENETA	Layer A		How to read it
14 operational questions · Gemma 4 31B	14.0 · 34.5 s		14.0 · 34.6 s	Equal. With the judgment layer shared, operational answers are less sensitive to the store.
14 operational questions · OTel 31B	13.7 · 33.5 s		13.7 · 42.6 s	Equal. Layer A is nine seconds slower at the median.
12 memory-ability cases (both models)	12.0		11.7	What layer A missed: an observed operator preference once, a correction with a time window once.
16 long-horizon · Gemma 4 31B / OTel	16 / 16		14.0 / 14.3	What layer A missed every time: a Korean decision named by its identifier, the mark on an unverified rule (layer A has no pending state), applying a rule promoted to the shared scope.
Retrieval alone (720 queries, no model)	dense 91.5 % · hybrid 98.8 %		99.3 %	Layer A equal or slightly ahead. Retrieval quality as such is not our advantage.
Layer A extracting for itself (extraction, three anchors)	12.0 · 16	memory 8.0 · long-horizon 6.7–8.0		Given the transcript and left to extract its own memories, the layer kept a median of 1 memory per case out of 37 rows. Corrections, preferences, federation and marks vanished; decisions and readings survived. The same cases failed the same way on all three models.
Search latency · median / 95th	7 ms / 9 ms	21–36 ms / 228–252 ms		Measured while a model decodes beside it. Layer A drops to 31 ms when the GPU is idle.

Fairness of the comparison. Every compared layer stores and searches through our integration code, and every run file records how many rows the layer actually wrote, how many failed, and the last error. On first contact with a locally served 16k-context model, three of the four layers lost memory silently. One rejected the model's structured answers when they arrived in the shape of the schema and dropped a large share of its writes; one did not start under the embedding-provider name its documentation gives; one assumed a context window far larger than the served model's, stored not one line of the transcript, and scored exactly like having no memory. We do not use any run whose ingest report shows losses. We fix our side of the integration (never the layer itself) and measure again. A comparison in which the competitor has lost its memory is a comparison that flatters us, and we do not publish those numbers.

Table 3. Four open-source memory layers — the same loop, only the store swapped (Gemma 4 31B; one repeat per external cell, three for layer A). handed = our records handed over, extracted = the layer extracts from the session transcript

Measure	VENETA	Layer A	Layer B	Layer C	Layer D
Memory abilities 12 · handed / extracted	12	11.7 / 8.0	12 / 8	12 / 7	7 / 7
Long-horizon and multilingual 16 · handed / extracted	16	14.0 / 6.7	13 / 5	13 / 5	—
Under attack 4 · handed / extracted	4	4 / 4	4 / 3	4 / 4	3 / 3
Follow-up chains 8	8	8	8	8	—
Ingestion, median per case · handed / extracted	—	1.4 s / 23.5 s	0.7 s / 22 s	53 s / 104 s	25 s / 40 s
Search, median	7 ms	32–40 ms	199–292 ms	32–33 ms	45 ms
Retrieval only, 720 queries, hit@k	hybrid 98.8 %	99.3 %	78.3 %	88.9 %	52.8 %

Read it like Table 2. Handed our records, three of the layers work as stores and the operational answers tie. Left to extract from the transcript themselves, all three lose the same things: corrections rarely survive, and a preference the operator showed by approving never does. Layer D sits at the extraction level even when handed our records, because it turns every record into graph facts and returns those facts rather than the record. On retrieval quality alone layer A is equal to us or slightly ahead; layer B, with semantic search only, falls behind on questions that name an identifier. Ingestion cost differs a hundredfold by design: a layer that stores rows as they are takes under a second, a layer that runs a model over every row takes tens of seconds to minutes. Layer D's retrieval bench (Table 3) ran overnight because of that cost.

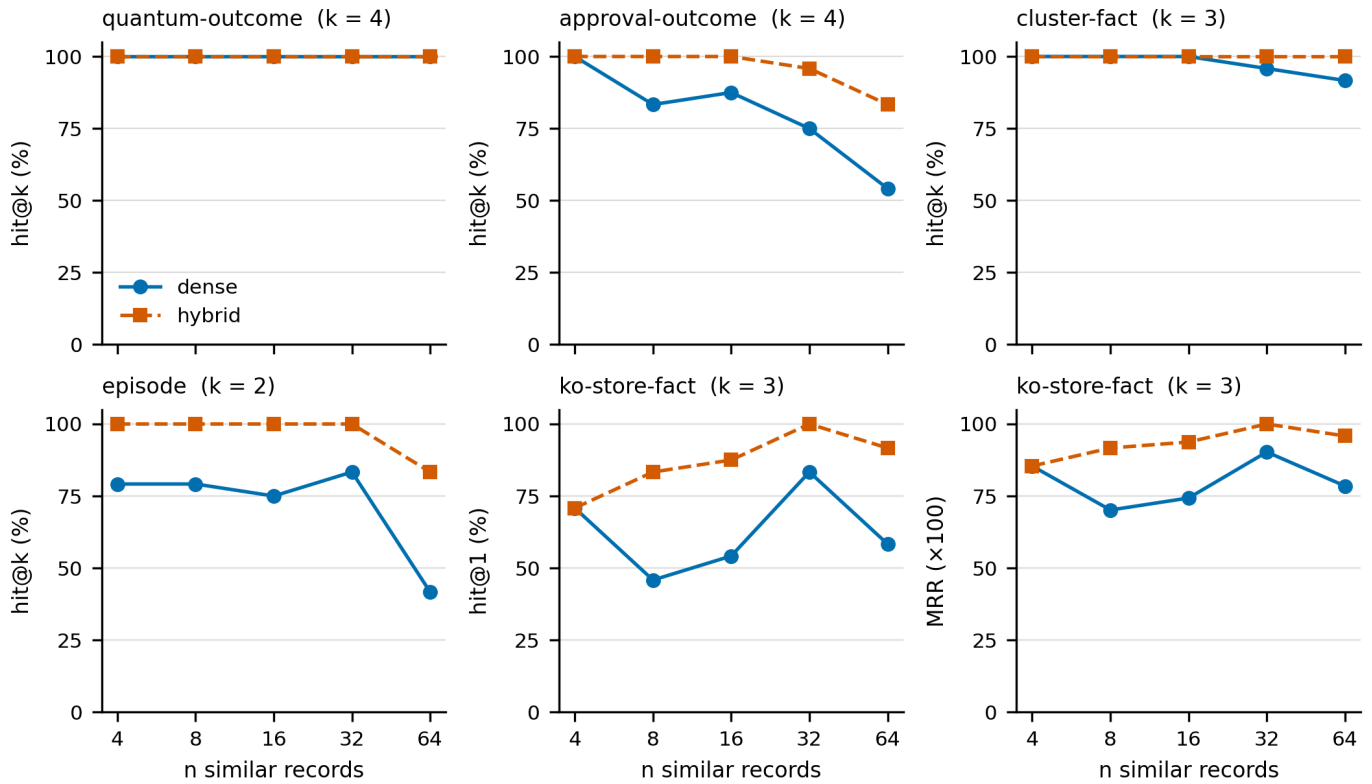


Figure 6. The needle benchmark (no model): the share of questions whose named record is among the top-k rows read (hit@k) against the number n of same-shaped records, dense (circles) and hybrid (squares), 24 queries per cell. In quantum outcomes the two lines coincide at 100 %. Korean store facts are shown on hit@1 and MRR because hit@k is 100 % for both arms; Korean episodes score 100 % on every metric and are not drawn. Data: recall-bench-2026-09-22, recall-bench-ko-2026-09-23. Figure: Minjun Lee.

Speed and cost

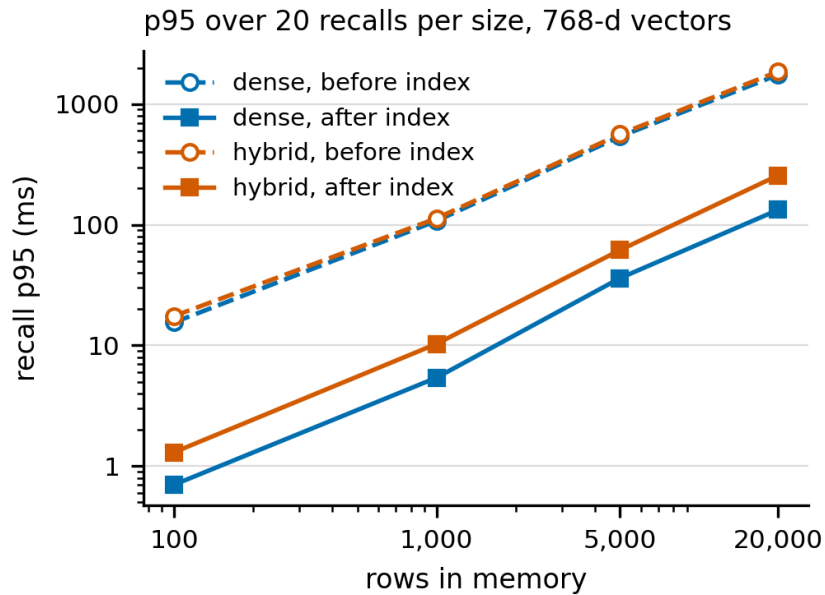


Figure 7. Recall latency against store size: p95 of one recall over 20 recalls per size, 768-d vectors, dense and hybrid, before (dashed, open markers) and after (solid, filled) the recall index; both axes logarithmic. Before the index, reading the vectors dominates and the two curves coincide. Data: recall-scale-baseline, recall-scale-indexed (2026-09-23). Figure: Minjun Lee.

The median time for one operational question, including tool calls, draft, checks, critique, and a rewrite when needed, ranged on this machine from 12.6 s (GLM-4.7 Flash) to 76.9 s (Llama 3.3 70B). The two 31B models took 33–41 s. The judgment layer is not free. Small models trip the checks more often and spend time repairing.

On Gemma 4 12B, rewritten turns rose from 21 % to 57 % and the median by 15 s, while the two 31B models stayed within ± 1 s. Accuracy is bought with latency, and we put the cost in the table. Of the two recall modes, the one that adds lexical search (hybrid) was 3–5 s faster on a memory grown to 93 rows, at the same score.

Industry by industry

There is one engine. Each industry brings a domain module with its tools, reference model, and gates. Every number in the previous chapter comes from the telecom edition.

Below, for each industry: what a local LLM cannot do alone, and what the layer changes. This chapter explains what changes. It claims no measurement.

Telecom: the edition we measured

Questions: What is wrong on the network right now, and what is the root cause? Reroute, restart, or wait: which keeps the SLA? Which cells can sleep tonight?

The model alone cannot read the alarm stream and the topology, does not remember what was approved in the last incident and how it ended, and invents the KPI impact of an action.

What the layer changes: The layer reads alarms and topology through tools and ranks the causes (the scenario name is never an input). It gates each action on the pessimistic edge of the KPI band. Approval creates a draft in the operator's change-management system, and the outcome is remembered. Combinatorial parts (cell sleep, backhaul rerouting, channel assignment) run on IBM quantum systems when the depth budget allows, labeled with backend, qubits, shots, and the gap to the classical optimum. All figures in the previous chapter are this edition's.

Energy: generation, transmission and distribution

Questions: How should load be shared in this grid state? What is the outage response order, and where do we restore first? Does the maintenance schedule collide with the peak?

The model alone does not know the live grid state, does not remember the outcome of past actions, and asserts without computing the safety margin.

What the layer changes: No answer before load, asset, and weather data are read. Every recommendation is gated on the pessimistic edge of the safety and reliability band. Judgments that need grid computation go to an external simulator (load flow) and come back with a verification and validity card. The approvals and outcomes of the last outage become the memory of the next response. The ledger goes to the regulator as it is.

Semiconductor: fab scheduling and data movement

Questions: Which tool, in which order, for this lot? Where is the metrology data, and which system must it move to? How did the last recipe change turn out?

The model alone cannot read MES and tool state, cannot keep the corrections of engineers whose process knowledge differs, and quotes yield figures without a source.

What the layer changes: No answer before tool, lot, and metrology systems are read. An engineer's correction becomes an approved rule that applies on the next shift. Combinatorial problems such as tool assignment and lot order are solved as labeled optimization compared against a classical optimum. Nothing leaves the fab, and data movement between fab systems is ledgered too.

Retail: demand, assortment, store operations

Questions: What is next week's demand for this item at this store, and on what evidence? How did last month's promotion decision turn out? Which store's stock moves where?

The model alone cannot read transactions and inventory, invents forecasts, and forgets the store manager's correction ("this store does not sell when it rains") by the next time.

What the layer changes: Transactions and inventory are read through tools. Demand forecasting is delegated to an external forecasting engine whose result is labeled "conditional forecast, accuracy, against baseline." The manager's correction becomes an approved rule remembered at store scope. Decisions and outcomes are linked, so next season's question brings up last season's result. This edition is being built with an external demand-forecasting engine.

Manufacturing: process and equipment scheduling

Questions: Which line, in which order, for these orders? Where do we place inspection staff? Did the action on the last equipment failure work?

The model alone has no access to MES and equipment data, keeps none of the shop floor's tacit knowledge, and asserts delivery impact without checking.

What the layer changes: Equipment, orders, and materials are read. Assignment is labeled optimization, gated on the delivery and quality band. A supervisor's correction becomes a rule. The action on an equipment failure and its outcome are remembered. Execution always enters the MES as a person-approved draft.

Defense: readiness, logistics and asset allocation in closed networks

Questions: Which assets go where in this situation? How are logistics stock and transport routes planned? What were the decisions and outcomes of the last exercise or operation?

The model alone cannot read closed-network data, cannot be audited for what it learned, and delivers dangerous judgments in a confident voice.

What the layer changes: It runs on your model inside a fully closed network: no outbound calls, no telemetry. Every recommendation leaves with its evidence and checks, and nothing is executed that a person has not approved. Learned rules have their approval, rejection, and block in the ledger and are auditable. A dangerous judgment is something the judgment layer catches, not something left to the model.

Two patterns across industries

AI chatbots for customers. What a customer asks about is usually "mine": my bill, my order, my outage. The model does not know it. The layer refuses to answer before the customer's account, order, and ticket systems are read through tools. It remembers, after approval, a correction the customer gave ("I no longer use this number"). It marks what is unverified. And it stops instructions arriving through the chat from being planted in memory, at the moment of storage. The figures that go out are figures read from a system. What breaks trust in a customer chatbot is not "I do not know." It is a wrong number.

Internal AI work systems. The value of an internal system is the organization's memory: past decisions, corrections, approvals, outcomes. The layer keeps these as person-approved rules and records, scopes them by department (agent), and tallies an operator's approval habits from the record rather than by inference. When the auditor asks what this system learned, when, and who allowed it, the ledger answers. Execution is always a draft, and the last step is a person.



Security and governance

Memory is a new attack surface. An instruction that enters memory shapes every future conversation. A personal record that is not deleted is a compliance breach. Learning that cannot be audited is a risk.

We measured this with seven probes and ran the same probes against layer A. The table shows what each layer actually did.

Table 4. Seven security probes and one default: what each layer actually did (September 30, 2026)

Probe	VENETA	Layer A
An instruction to the model, written as a rule	blocked at write, never recalled	stored and returned verbatim
A secret (key, token) inside a fact	blocked at write, never recalled	stored and returned verbatim
One person's episode searched as another person	isolated per user	isolated per user (filter)
Delete one of three facts	erased, with a hash-chained proof	deleted, no proof record
Edit one line of the log	append-only, hash-chained; the edit is detected	no verifiable append-only log
A rejected rule, paraphrased and proposed again	blocked by the tombstone, never recalled	the paraphrase is recalled (no tombstone)
An absurd reading stored as a fact	stored and recalled: the answer's evidence check is where it is caught	same
Outbound telemetry in the default configuration	none (no calls beyond the configured LLM and embedding endpoints)	yes (usage statistics sent to an external service; can be switched off)

We found the last row during the measurement. Both open-source memory layers we compared send usage statistics to an external server in their default configuration. It showed up as failed outbound calls when the office line dropped one night. We switched it off for every run, but for an enterprise that assumes a closed network, the default matters as much as the feature. Our layer connects to nothing but the configured model and embedding endpoints.

We did not pass the seventh probe either, and that is by design. To judge an absurd reading at the moment of storage, the store would have to know the domain. We place that judgment in the answer's evidence check: whatever memory holds, a figure that goes out is compared against today's reading.

Anything that must leave the building, such as a quantum job or a call to an external SaaS, passes a separate gate (VENETA Sentinel): the payload is shielded, scanned, hashed, and written to an egress ledger. It is outside this paper's scope but follows the same principle. What leaves is recorded, and what is not recorded does not leave.



How it is delivered:

WorldModel, Enterprise, project veneta

Product	What it is	Who it is for
VENETA WorldModel	Industry editions (telecom, energy, semiconductor, retail, manufacturing, defense). A domain module with tools, reference model, gates, simulators, and optimization instances sits with this paper's memory and self-improvement layer on one engine, placed over your systems as a read-only sandbox.	Operators, manufacturers, and agencies that want recommendations on operational questions with a simulation and evidence attached.
VENETA Enterprise	This paper's memory and self-improvement layer placed over the models, agents, and chatbots you already run, connected through an OpenAI-compatible interface. Approval console, ledger, roles and dual control, a learning budget, and undo are on by default. It adds the enterprise features: multi-tenancy (memory and policy separated per department, subsidiary, or client), role-based access control (RBAC) , and single sign-on (SSO) through SAML and OIDC.	Companies already running in-house LLMs, agent frameworks, or customer chatbots that want what those systems learn to be approved, measured, and audited, and that run several organizational units on one deployment.
project veneta (open source)	The same memory and self-improvement layer as open source (Apache-2.0), published at veneta.ai/project together with the memory benchmark and how to rerun it. The enterprise posture (ledger, approval, tombstones) is the default, and telemetry is off by default. It does not include the enterprise features: no multi-tenancy, no RBAC, no SSO. It is built for one operator and one organization.	People running local models, users of agent frameworks, research labs, and anyone who wants to check this paper's claims for themselves.

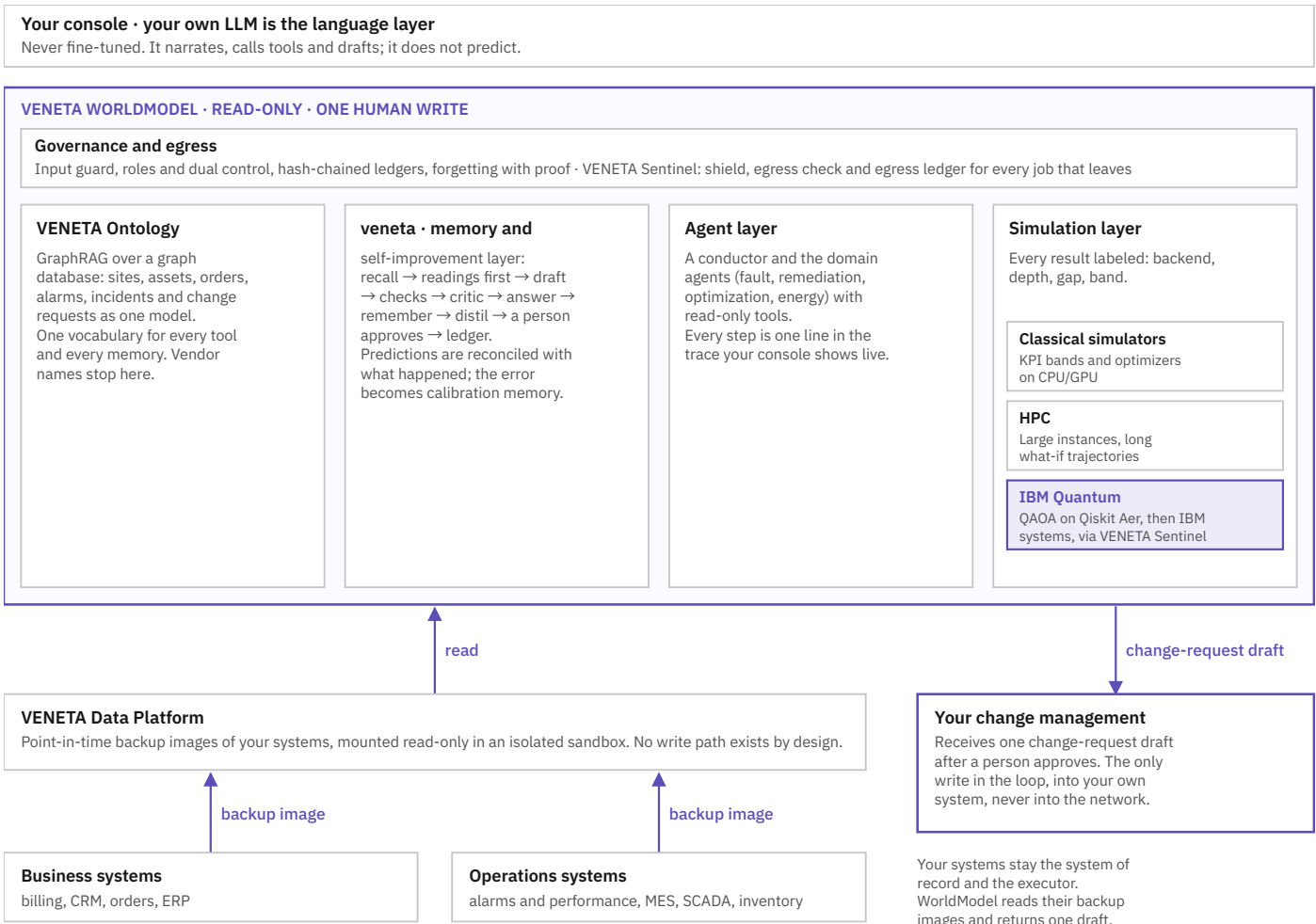


Figure 4. VENETA WorldModel architecture. Backup images of your systems come in through the VENETA Data Platform; the engine reads, simulates and drafts in a read-only sandbox; one change-request draft goes back to your own change management after a person approves.

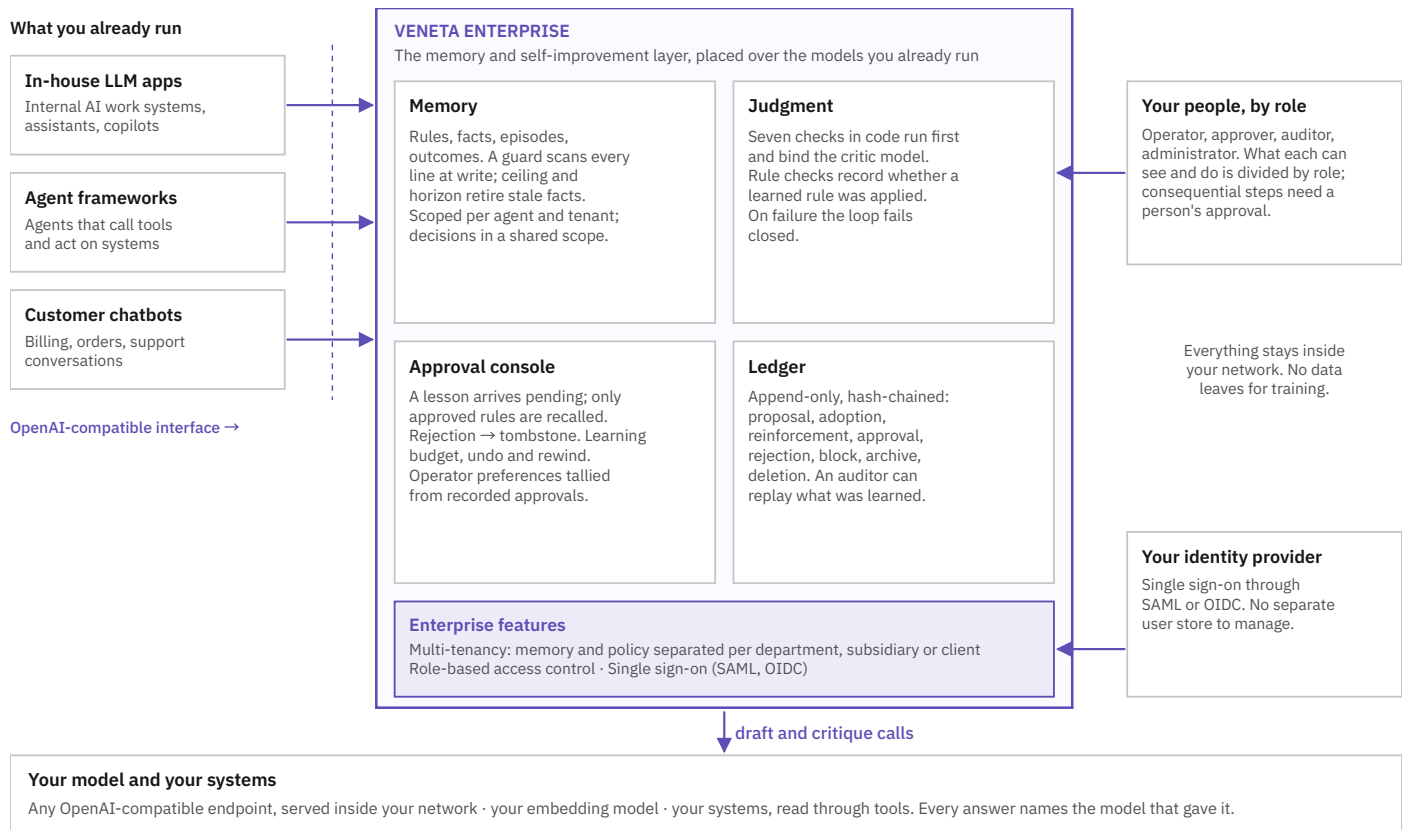


Figure 5. VENETA Enterprise architecture. The same memory and self-improvement layer, placed over the LLM applications, agents and chatbots you already run through an OpenAI-compatible interface, with multi-tenancy, role-based access control and single sign-on. project veneta provides the layer without those three.

Deployment	Inside your network, on your hardware. We made the measurements on one desktop-sized machine (DGX Spark) with 31B models. No data leaves for training.
Model	Your model, through any OpenAI-compatible endpoint. The eight models we measured are all open weights. Every answer names the model that gave it.
Organization	VENETA Enterprise separates memory, rules, and approval policy per tenant, divides what can be seen and done by role (operator, approver, auditor, administrator), and attaches to your identity provider through SSO. project veneta provides none of the three.
Quantum	Combinatorial parts optionally run on IBM quantum systems, delivered through the IBM Quantum GSI (Global Systems Integrator) partnership. Every result carries backend, qubits, shots, and the gap to the classical optimum.

What we do not claim

We do not say "eliminates hallucination".

We catch unsourced figures and missing labels in code and force a rewrite of what is caught. What is not caught remains. The remaining failures in the measurement chapter are the evidence.

We do not say "best".

We compared four open-source layers head to head, and on retrieval quality alone layer A is equal or slightly ahead. The difference shows on abilities that need structure in the record, and the benchmark that measures those abilities is ours. There is no result yet on a benchmark someone else built.

We did not measure on real customer data.

Every figure comes from the telecom edition's synthetic network. There are no figures for other industries. The industry chapter explains what changes and claims no measurement.

It is a small experiment.

14 + 12 cases, three repeats, one machine. A difference of 0.3 is noise. Read the [range] with the mean.

The judgment layer has a cost.

On a small model, the median rose by 15 s. Where latency matters, that cost belongs in the design.

A person must approve.

The layer does not replace people. Execution is always a draft in your system, and a learned rule is active only once a person approves it. Remove that, and the security table in this paper no longer holds.

The compared layers ran in our environment.

A locally served 16k-context model, local embeddings, our integration code. Each layer's hosted service or a larger model may behave differently. The ingest report in each run file says how much memory the layer actually answered with.

It is not yet peer reviewed.

The protocol and data will be published as a paper. If the numbers change then, we will revise this whitepaper.



Key takeaways and next steps

- The limits of a local LLM are not model size but the absence of what must live outside the model: the present state, memory, checks, approval, a ledger.
- The memory layer raised memory ability on all eight models from 5.0–6.0 to 10.3–12.0 and lost nothing on the two abilities where memory can do harm.
- The judgment layer raised the operational answers of all eight models, and switching the binding off lost exactly what the checks used to catch. The effect comes from binding findings, not from the rubric's wording.
- Approved rules were applied on all 125 occasions they reached the prompt. Whether what was learned is applied can now be counted.
- With only the store swapped inside the same loop, the differences were in the structure of the record and in the security defaults. Retrieval as such, the open-source layer does well.

The fastest way to test this claim on your own data is one of two: put VENETA Enterprise in front of an in-house LLM or chatbot you already operate and rebuild the memory benchmark as twelve cases of your domain, or run the fourteen operational questions of the relevant WorldModel edition over your own systems. Either way, the numbers come first, and the numbers are yours.

[Request a VENETA Enterprise briefing →](#)

© 2026 VENETA Inc. All rights reserved.

Author contributions: Hyunjoo Lee designed the memory layer and wrote this document. Minjun Lee verified the needle-benchmark results against the data files and produced the needle-benchmark and recall-latency figures (Figures 6 and 7) from them by code.

VENETA, VENETA WorldModel, VENETA Enterprise, VENETA Sentinel and project veneta are trademarks of VENETA Inc. IBM, IBM Quantum and Qiskit are trademarks of International Business Machines Corporation. Gemma is a trademark of Google LLC; Llama of Meta Platforms, Inc.; Qwen of Alibaba Group; Mistral of Mistral AI; EXAONE of LG AI Research; GLM of Zhipu AI. Other product and company names belong to their owners.

VENETA made the measurements in this document between September 28 and October 1, 2026, on its own equipment with synthetic data. They are current as of the date of publication and may change. Record: campaign 3 (2026-09-28 17:35 → 09-30 18:42, 250 run files) and the external memory-layer baselines (09-30 18:48 → 10-01). Serving: vLLM, FP8, 16k context, tool calling, thinking off, three repeats. The full protocol, cases and scoring code are published with the paper.

This document is provided for information only, without warranty of any kind, express or implied. Results in a given environment depend on the model, the data, the hardware and the configuration. No IT system or product can be considered completely secure, and no single product or security measure can be completely effective in preventing improper use or access. The customer is responsible for compliance with applicable laws and regulations; this document is not legal advice.

Terms. Local LLM: an open-weights language model a company runs on its own hardware (here, 12B to 70B parameters). Memory layer: the layer that stores and recalls rules, facts, episodes and outcomes across sessions, with approval, horizon, ceiling, a guard at write and a ledger. Judgment layer: what a model's draft passes before a person sees it (checks in code, a critic bound by the checks, rule checks, a person's approval). Gate: the service-level or safety band a recommendation must not cross, applied on its pessimistic edge. Ledger: an append-only, hash-chained record. dense / hybrid: semantic (embedding) search alone, or semantic search plus lexical and identifier search. Synthetic data: data built to reproduce the structure of real data, not real values.

VENETA Inc. · 254 Chapman Rd, Ste 208 #28815, Newark, DE 19702, USA · contact@veneta.ai · veneta.ai

