

GOVERNED CONTINUAL LEARNING

Learning from Use

A governed framework for continual improvement
of a self-hosted enterprise LLM assistant

AUTHOR

Ishan Mishra

Lead Applied Scientist, Neurologic
ishan.mishra@neurologica.com

PUBLISHED

September 2026

PLATFORM

Corporate GPT

ABSTRACT

Enterprise LLM assistants are typically deployed as static systems: the model, prompts and search index are fixed at release, while the organisation's vocabulary, data and working practices continue to change. Every interaction nevertheless produces evidence of success or failure, in the form of corrections, rephrasings, edits, and accepted or rejected actions, and this evidence is almost always discarded. We present a framework for continually improving Corporate GPT, Neurologic's self-hosted enterprise assistant, from its own usage. The framework captures a complete record of each interaction, derives outcome labels from

explicit and implicit user feedback, and routes each lesson to the cheapest part of the system able to absorb it: business definitions, search, agent instructions, user memory, or the model's weights. Because the model (Qwen3.8-Max) is hosted in-house, all five routes, including fine-tuning of the model itself, are available without data leaving the organisation. Every candidate change is evaluated against human-verified answers and requires human approval before release. A scoping rule ensures that no lesson reaches further than the access permissions of the evidence it came from. We describe the design, its grounding in recent research on preference optimisation, prompt evolution and continual adaptation, an evaluation protocol, and a phased implementation plan.

KEYWORDS

Continual learning

Learning from human feedback

Retrieval-augmented generation

Agent self-improvement

Parameter-efficient fine-tuning

AI governance

01

Introduction

The value of an enterprise assistant lies less in general capability than in its fit to one organisation: its definitions, its systems, its people and their preferences. Public benchmarks measure the former; only real usage reveals the latter. A question such as “how many active customers did we lose last quarter?” depends on what *active customer* means in a particular database, which legal entities are in scope, and how the asker expects the answer to be presented. None of this is known at deployment time, and much of it changes afterwards.

Corporate GPT already records every conversational turn, together with a detailed trace of how the answer was produced. When an answer fails, users signal it: they rephrase, retry, correct the assistant, rewrite its drafts, or abandon the session. When it succeeds, they copy the result, open its citations, or accept a proposed action. This behaviour is a continuous and essentially free supply of supervision, yet in most deployed assistants, including ours until now, none of it flows back into the system. The assistant is as good on its thousandth day as on its first.

WHEN AN ANSWER FAILS

Rephrase

Retry

Correct the assistant

Rewrite its drafts

Abandon the session

WHEN IT SUCCEEDS

Copy the result

Open its citations

Accept a proposed action

This paper asks three questions. First, which usage signals are reliable enough to learn from, and how should they be combined into labels? Second, where in a multi-agent assistant should each lesson be applied, given that model weights are only one of several places where behaviour is encoded? Third, how can continual learning be made safe in an enterprise, where access control, privacy and auditability are non-negotiable and where optimising for user approval is known to produce sycophancy, the tendency to tell users what they want to hear?

Our contributions are:

1

A **signal taxonomy and labelling scheme** that converts heterogeneous explicit and implicit feedback into outcome labels with calibrated confidence (Section 4).

2

An **improvement hierarchy** that routes each diagnosed failure to the cheapest adequate layer, from business definitions through search, agent instructions and memory to parameter-efficient fine-tuning of the self-hosted model (Section 5).

3

A **three-ring evaluation protocol** anchored on human-verified answers, with explicit guards against forgetting and sycophancy (Section 6).

4

A **scoping rule** that bounds every lesson by the access permissions of its evidence, together with supporting privacy and audit controls (Section 7).

5

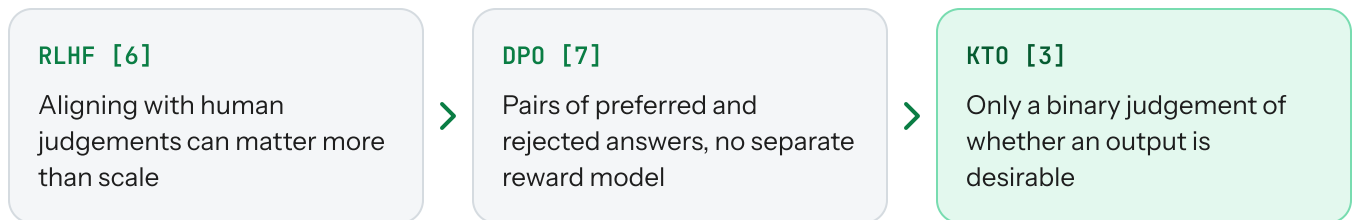
A **phased implementation plan** in which each phase delivers value independently (Section 8).

Section 2 reviews related work, Section 3 describes the system, and Sections 9 and 10 discuss limitations and conclude. A glossary of technical terms is provided in Appendix A.

Background and related work

2.1 Learning from human feedback

Reinforcement learning from human feedback (RLHF) established that aligning a model with human judgements can matter more than scale: a 1.3B-parameter InstructGPT model was preferred by human raters over the 175B-parameter GPT-3 [6]. Direct Preference Optimization (DPO) simplified this by learning directly from pairs of preferred and rejected answers, without a separate reward model [7]. Kahneman-Tversky Optimization (KTO) went further, requiring only a binary judgement of whether an output is desirable, and matched or exceeded pairwise methods at 1B to 30B parameters [3]. Binary signals are exactly what production systems collect in abundance, which makes KTO-style objectives a natural fit for learning from usage.



2.2 Learning from implicit signals and edits

Explicit ratings are sparse; implicit behaviour is plentiful. Gao et al. show that an LLM agent can infer a user’s latent preference from the edits they make to its outputs and apply that inferred preference to later prompts, reducing subsequent editing effort without any fine-tuning [4]. This result motivates treating edits to generated drafts as a first-class signal and storing inferred preferences in memory rather than in model weights.

2.3 Self-improving agents through context and prompt evolution

A growing body of work improves agents by changing their instructions and context rather than their parameters. GEPA has a model reflect in natural language on past attempts to evolve its prompts, outperforming GRPO, a widely used reinforcement-learning method, by 6% on average across six tasks while using up to 35 times fewer attempts [1]. Agentic Context Engineering (ACE) maintains an evolving “playbook” of strategies updated in small increments, and identifies *brevity bias* (summaries that drop useful detail) and *context collapse* (information lost through repeated rewriting) as failure modes of naive prompt rewriting; it reports gains of 10.6% on agent benchmarks and 8.6% on finance tasks using

execution feedback rather than labels [11]. Agent Workflow Memory extracts reusable workflows from past successful runs and improved success rates by 24.6% and 51.1% relative on the Mind2Web and WebArena web-navigation benchmarks [10]. Together these results indicate that a large share of achievable improvement can be obtained without touching model weights.

GEPA [1]

6%

over GRPO on average across six tasks, with up to 35 times fewer attempts

ACE [11]

10.6% / 8.6%

gains on agent benchmarks and finance tasks

AGENT WORKFLOW MEMORY [10]

24.6% / 51.1%

relative success-rate gains on Mind2Web and WebArena

2.4 Risks of continual adaptation

Three risks recur in the literature.

Forgetting

Low-rank adapters (LoRA) learn less than full fine-tuning but better preserve the model’s other capabilities [2], and on-policy reinforcement learning, where the model learns from its own sampled outputs, forgets less than supervised fine-tuning because it tends to stay closer to the original model’s behaviour [8].

Model collapse

Models trained repeatedly on their own generated data progressively lose information about rare cases [9], so learning pipelines must retain genuine human data.

Sycophancy

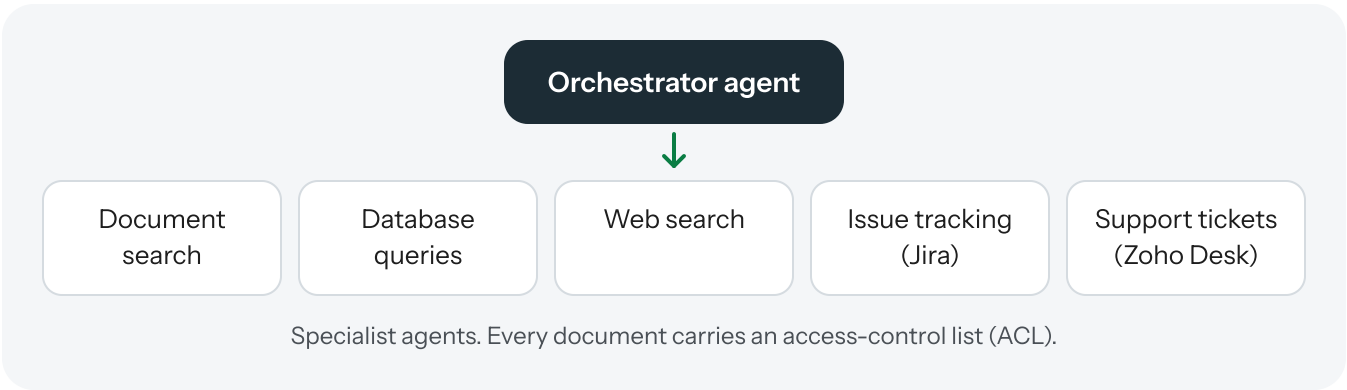
In April 2025 OpenAI rolled back a GPT-4o update after acknowledging it had “focused too much on short-term feedback” such as thumbs-up ratings, producing responses that were “overly supportive but disingenuous” [5]. Any system that learns from user approval inherits this risk by construction.

2.5 Positioning

Prior work typically studies one improvement method in isolation. Our contribution is systems-level: a single loop that diagnoses failures from production usage, routes each to the appropriate method, and applies enterprise constraints on access, privacy and approval throughout.

System context: Corporate GPT

Corporate GPT is Neurologic’s enterprise assistant. It answers employees’ questions over a company’s documents, business systems and relational databases using retrieval-augmented generation (RAG): relevant passages or query results are retrieved first, and the model composes an answer grounded in them. A central orchestrator agent delegates each request to specialist agents for document search, database queries, web search, issue tracking (Jira) and customer support tickets (Zoho Desk). Document search combines semantic and keyword matching, and every document carries an access-control list (ACL), so users only ever retrieve content they are permitted to see.



Self-hosted model. Corporate GPT relies on no third-party model provider. Its primary model is Qwen3.8-Max, an open-weights mixture-of-experts model with 2.4 trillion total parameters, of which 95 billion are active for any given token, and a native context window of 262,144 tokens, released in August 2026 [12]. Self-hosting has two consequences central to this paper. First, every layer of the system, including the model’s weights, is under our control and can be adapted. Second, conversational data used for learning never leaves the organisation’s infrastructure, which removes a major obstacle to learning from usage.

2.4T total parameters, Qwen3.8-Max	95B active for any given token	262,144 token native context window	None third-party model provider relied on
---	---	--	--

Several existing capabilities serve directly as foundations for the learning loop (Table 1).

Capability	What it does today	Role in the learning loop
Conversation logs and execution traces	Store every turn with a step-by-step record of agent decisions, tool calls and retrieved passages, using open-source tracing (OpenTelemetry, Arize Phoenix)	Source of complete interaction records
Skill files	Plain-text instruction files, one set per agent, kept under version control and reviewed like code	Editable, reviewable target for instruction-level learning
Evaluation harness and competency set	Automated test suite that scores answers against a curated set of real user questions with human-verified answers	Ground truth for every candidate change
Memory service	Stores facts and preferences that persist across sessions	Store for personal and team preferences
Embedding fine-tuning pipeline	Retrains the model that powers semantic search	Continuous search improvement from usage
Personal-data redaction	Detects and masks personal information (Microsoft Presidio)	Mandatory scrub of all learning examples
Knowledge layer	Machine-readable business definitions, valid ways of combining database tables, a single identity for each person or customer across systems, and a query checker that rejects database queries violating these rules before they run	Target for definition-level lessons; its confirmation step for proposed agent actions yields accept/reject signals
Feature flags	Switch a capability on for selected users or deployments	Staged, reversible rollout of every learned change

Table 1. Existing Corporate GPT capabilities reused by the learning loop.

04

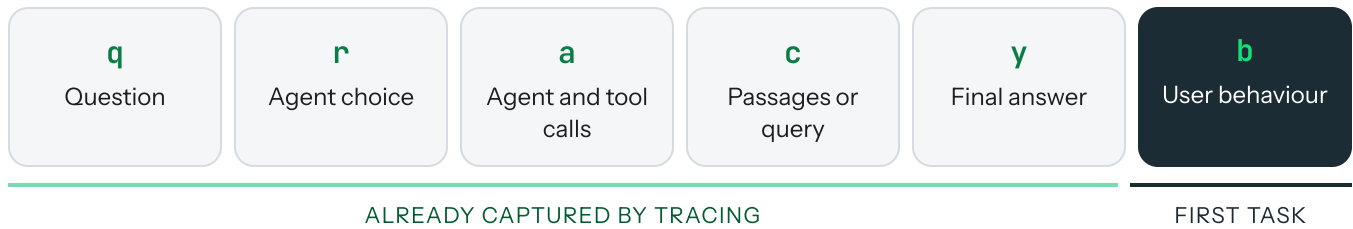
Problem formulation: usage as a learning signal

4.1 Interaction records

We take the unit of learning to be the complete interaction record, which we call a *trajectory*, rather than the final answer alone:

$$\tau = (q, r, a_{1:k}, c, y, b) \quad (1)$$

where q is the user's question, r the orchestrator's choice of agent, a the sequence of agent and tool calls, c the retrieved passages or generated database query, y the final answer, and b the user's subsequent behaviour. Judging only y hides the cause of failure; the full trajectory distinguishes whether the fault lay in agent selection, search, query generation or the writing itself. All components except b are already captured by our tracing; capturing b is the first engineering task.



4.2 Signal taxonomy

Usage signals differ widely in strength and bias (Table 2).

Signal	Example	Strength	Principal bias	How it is captured
Explicit correction	"No, Q3 revenue excludes the APAC entity"	Very high	Rare; skewed to expert users	Follow-up message classified as a correction
Edit of generated output	User rewrites 40% of a drafted email	High	May reflect style rather than error	Comparison of generated and final text
Accepted or rejected action	User rejects a proposed Jira update	High	Reason often unstated	Confirmation prompt for agent actions
Thumbs up or down	Downvote on an answer	Medium	Rewards agreeableness over truth	Rating buttons in the chat interface
Rephrase or regenerate	Same question re-asked within 60 s	Medium	May indicate curiosity	Derived from the session history

Signal	Example	Strength	Principal bias	How it is captured
Re-route instruction	“Check Jira, not the docs”	Medium	Sparse	Requested agent compared with the one chosen
Citation opened, answer copied	User copies a result table	Low to medium	Weak positive only	Interface events
Abandonment	Session ends after an answer	Low	Satisfied users also leave	Derived from the session history

Table 2. Taxonomy of usage signals.

4.3 Outcome labelling

No single signal constitutes a label. For each trajectory we compute a weighted outcome score over the observed signals:

$$s(\tau) = \sum_k w_k \varphi_k(b), \quad \ell(\tau) = \begin{cases} \text{success} & s(\tau) \geq \theta^+ \\ \text{failure} & s(\tau) \leq \theta^- \\ \text{uncertain} & \text{otherwise} \end{cases} \quad (2)$$

Here each φ indicates whether a given signal was observed, each w is that signal’s weight, and the thresholds θ separate success from failure. The weights and thresholds are calibrated against a sample of at least 200 turns labelled by people, with a target agreement of 85%. Strong signals (corrections, edits, rejected actions) receive weights large enough to decide the label on their own; weak signals only break ties. Trajectories labelled *uncertain* are never used for training and are instead sampled for human review.

DESIGN PRINCIPLE

The design embodies one principle: **satisfaction is not correctness**. A user may approve a confident but wrong figure. Agreement with human-verified answers therefore outweighs any approval signal, both in labelling and, as Section 6 describes, in release decisions.

Proposed framework

5.1 The learning loop

The framework is a closed loop of six stages (Figure 1). Capturing data and proposing fixes are automated for speed; evaluation and release are gated for safety.

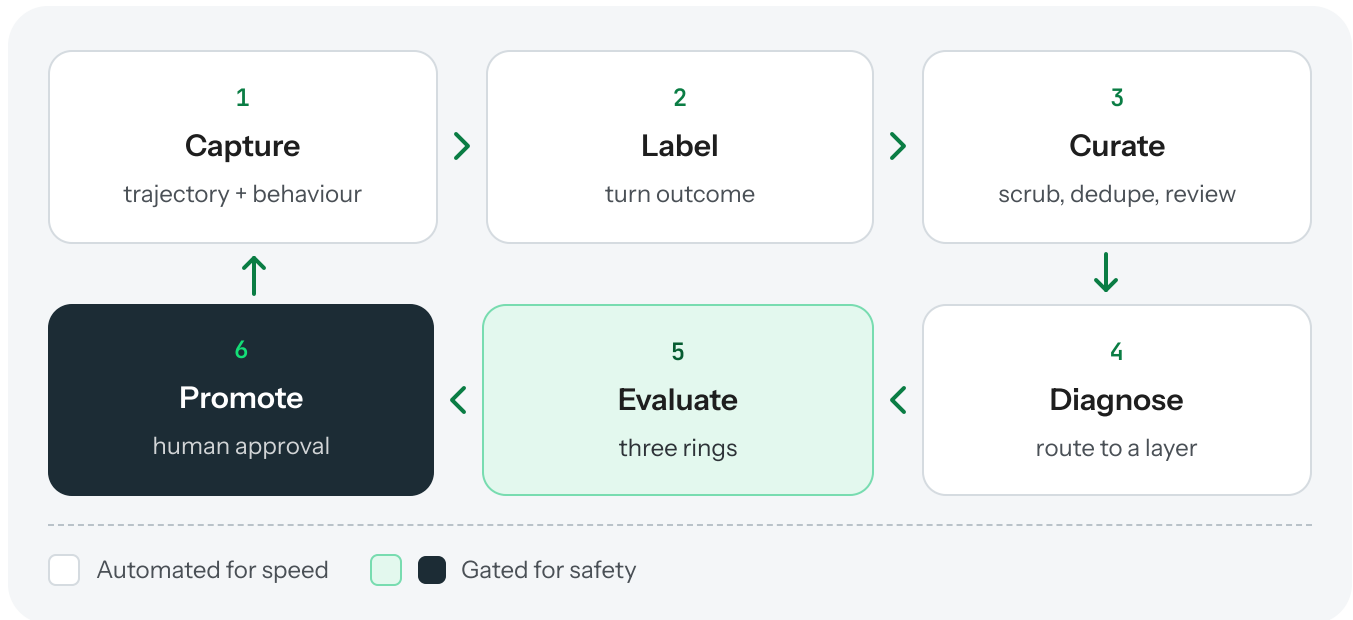


Figure 1. The learning loop. Released fixes add cases to the regression set, and reviewed failures become new test questions, so each iteration makes the next cheaper.

Capture links each conversational turn to its execution trace and to the user’s next action. **Label** applies the scheme of Section 4.3. **Curate** removes personal data and content from restricted sources, deduplicates, and groups failures by probable cause. **Diagnose** attributes each group to agent selection, search, query generation, missing knowledge, format or preference, and selects the layer at which to fix it. **Evaluate** and **Promote** (release) are described in Section 6.

5.2 The improvement hierarchy

A lesson should be applied at the cheapest, most reversible layer that can hold it. Table 3 orders the five available layers.

Layer	Fixes	Learns from	Time to ship	How to undo
L1 Knowledge layer	Wrong definitions, unknown terms, invalid table combinations, confused identities	Corrections; rephrasing around a term; query-checker rejections	Hours	Revert the definition change
L2 Search	Relevant document missed or ranked low	Opened citations; accepted vs rephrased answers	Days	Switch back to the previous search model
L3 Skills and playbooks	Wrong agent chosen, poor tool use, recurring reasoning errors	Failed vs successful trajectories	Days	Revert the skill-file change
L4 Memory	Individual and team preferences	Edits; restated instructions	Immediate	Mark the preference as no longer valid
L5 Model fine-tuning	Persistent behavioural or domain gaps the other layers cannot fix	Curated feedback and automatically checkable tasks	Weeks	Unload the adapter

Table 3. Improvement layers, ordered by cost and reversibility.

The diagnosis step is where most value lies. A group of failures such as “revenue answers include a divested entity” is a definition error, fixed by one line in the knowledge layer; treating it as a training problem would cost weeks and might not succeed.

5.3 Knowledge layer

Many failures concern meaning rather than reasoning. Repeated corrections to the same figure, or repeated rephrasing around the same term, indicate a missing definition, synonym or identity link. The knowledge layer, developed as a companion initiative, holds these definitions in machine-readable form; the learning loop supplies a ranked list of what to add or correct, ordered by how often each gap affects users. Rejections from the query checker are a particularly clean signal of missing or incorrect definitions.

5.4 Search

Our existing embedding fine-tuning pipeline, which trains the model behind semantic search, can be fed continuously from usage. A question paired with a passage the user opened before accepting the answer is a positive training example; a top-ranked passage followed by a rephrased question is a candidate *hard negative*, a passage that looked relevant but was not. The same examples can train a lightweight re-ranker that reorders search results. Examples are formed only from passages the asking user was permitted to see, so access permissions are preserved.

POSITIVE EXAMPLE

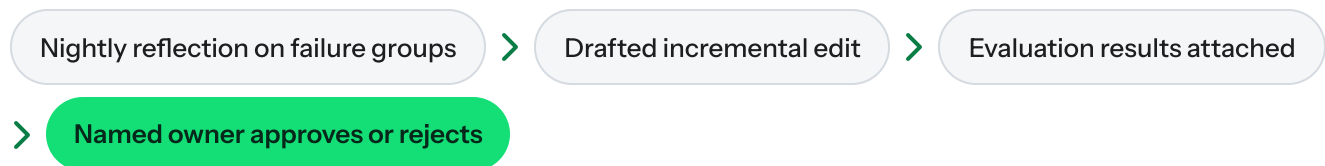
Question + passage the user opened, then the answer accepted

CANDIDATE HARD NEGATIVE

Top-ranked passage, then a rephrased question

5.5 Skills and playbooks

Each agent's behaviour is defined by a set of plain-text skill files kept under version control, so it can be improved the way software is: through a proposed change, backed by evidence and reviewed by a person. Following GEPA [1] and ACE [11], a nightly job reflects on groups of failures and drafts small, incremental edits to the relevant skill file, or new worked examples that teach the orchestrator which agent to choose. Following Agent Workflow Memory [10], successful multi-step interactions are distilled into reusable workflows. Each proposal is submitted for review with its evaluation results attached, and a named owner approves or rejects it. A size limit on each skill file prevents instructions from growing until they crowd out retrieved content, and incremental edits avoid the context collapse reported for wholesale rewrites [11].



5.6 Memory

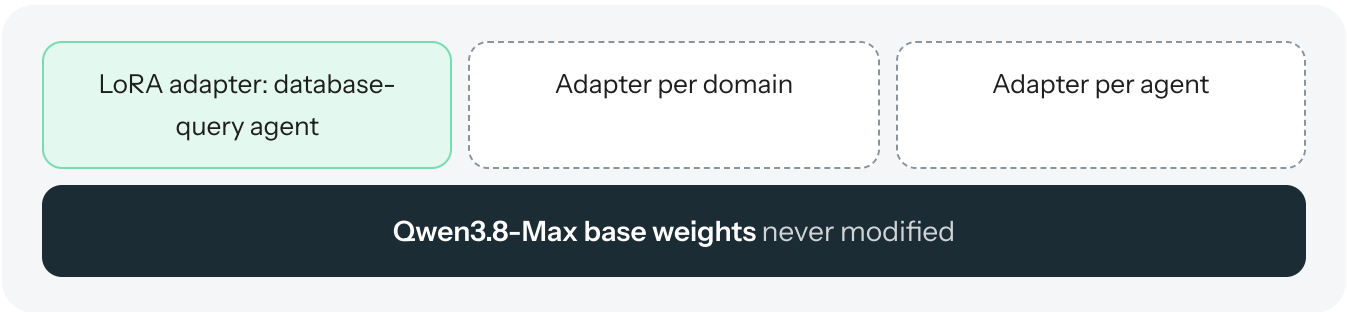
Some lessons belong to a single user or team: a summary format, a default project, a house style. Following Gao et al. [4], preferences are inferred from edits and restated instructions and stored in the assistant's memory service. Each stored preference records when it

became valid and when it stopped being valid, so a changed preference supersedes the old one without erasing its history. Personal memory is never promoted into shared skills without explicit review.

5.7 Fine-tuning Qwen3.8-Max

Because Corporate GPT hosts its own model, learning at the level of model weights is available and is a planned part of this framework. It remains the last layer in the hierarchy because it is the slowest to iterate and the hardest to reverse or audit. We propose the following design.

Small adapters on an unchanged base model. The Qwen3.8-Max base weights are never modified. Learning is applied through LoRA adapters, small sets of additional weights trained per domain or per agent (for example, one adapter for the database-query agent). LoRA preserves the model’s existing capabilities better than full fine-tuning [2], and adapters can be loaded, compared and removed independently. Serving several adapters at once on a model of this architecture is an engineering requirement we will validate before deployment.



Two training objectives, matched to the type of signal. For conversational behaviour, where labels are binary success or failure outcomes, we use the KTO objective [3]. For the database-query agent, correctness can be checked automatically by running the generated query and comparing its result with a known answer; here we use on-policy reinforcement learning with this automatic check as the reward, which also benefits from the lower forgetting of on-policy updates [8]. The reward requires both a correct result and a pass from the knowledge layer’s query checker, so the model is rewarded for queries that are both right and consistent with business definitions.

Data discipline. Every training set includes human-verified answers and a sample of general-purpose examples to preserve broad capability, and never consists solely of the model’s own unreviewed outputs, to avoid model collapse [9]. Only examples from unrestricted sources, after personal data has been removed, are eligible (Section 7).

Cadence and compute. Adapters are released on a fixed schedule, initially monthly, rather than updated continuously, which keeps each release auditable. At 2.4 trillion parameters, even adapter training on Qwen3.8-Max requires substantial multi-node GPU capacity. Where a narrow, high-volume task such as agent selection or query generation accounts for most failures, fine-tuning a smaller model from the same family for that task is a cheaper alternative to evaluate.

Base-model upgrades. Adapters are tied to specific base weights and must be retrained when the base model is upgraded. The durable asset is therefore the curated, versioned training and evaluation dataset, not the adapters themselves.

06

Evaluation methodology

A learning system without rigorous evaluation is merely a drifting one. Every candidate change, from a one-line definition edit to a new adapter, passes three rings of evaluation before release (Figure 2).



Figure 2. Three-ring evaluation protocol.

Ring 1: offline. The candidate is evaluated on three fixed test sets. The *competency set* consists of real user questions with human-verified answers. The *regression set* contains past failures that have since been fixed, and grows with every release. The *capability canaries* cover behaviours the change should not affect, such as citation format, declining out-of-scope requests, multilingual answers and general reasoning; they detect forgetting and are especially important for adapters.

Ring 2: shadow. The candidate answers a copy of live traffic without its outputs reaching users. Its behaviour is compared with the current production system, and disagreements are sampled for human review.

Ring 3: canary. A small, named group of users receives the change behind a feature flag for at least one week while guardrail metrics are monitored.

Release criteria. These are fixed in advance:

- 01 No category of the competency set gets worse beyond normal measurement noise, and the targeted category improves.
- 02 Correctness is scored against known answers. Preference scores from an LLM acting as judge never decide a release; where such a judge is unavoidable, the order in which candidate answers are shown is swapped and both results reported, since judges are known to favour whichever answer appears first.
- 03 95th-percentile response time rises by no more than 150 ms, and inference cost per query is reported alongside every accuracy change.
- 04 Guardrail metrics remain stable: answer length, rate of declined or abstained answers, the share of answers supported by citations, and a **sycophancy probe**, a fixed set of prompts containing false user assertions for which the correct response is to disagree.

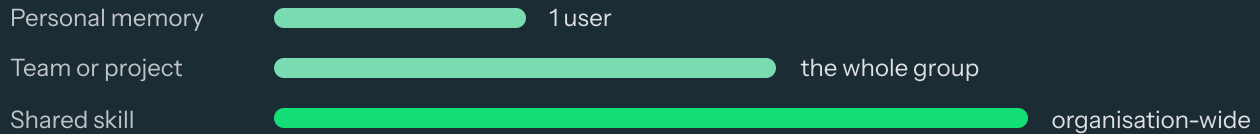
The sycophancy probe and length metric exist because any loop that learns from approval will tend to drift towards telling users what they want to hear [5]. They serve as the system's early warning.

— 07

Governance, privacy and security

The framework is governed by a single scoping rule: **a lesson may never reach further than the evidence from which it was derived.** A preference inferred from one user's edits remains with that user. A fact learned from a restricted document never enters a shared skill file or model weights, since neither instructions nor weights carry access controls. Table 4 applies the rule to each layer.

THE SCOPING RULE



Each lesson reaches only as far as its evidence.

Lesson scope	Eligible evidence	Minimum support	Approval
Personal memory	The user's own sessions	1 user	None; user may view and delete
Team or project memory	Group sessions on documents visible to the whole group	2 or more members	Group administrator
Shared skill or definition change	Public or organisation-wide documents only	3 or more users across 2 or more teams	Named owner, through review
Search model update	Question-passage pairs within each asker's permissions	Aggregate only	Applied scientist
Model adapter	Examples from unrestricted sources, personal data removed	Aggregate only	Applied scientist and security review

Table 4. Scope, evidence and approval requirements by layer.

The minimum-support thresholds also defend against **feedback poisoning**: no single user, whether malicious or mistaken, can alter behaviour for everyone.

Self-hosting strengthens the privacy position, because no conversational data is sent to an external provider for training. It also introduces a new obligation. Once adapters are trained on organisational data, **the adapter weights themselves become sensitive assets** and must be stored, versioned and access-controlled with the same care as the data they were trained on.

Four further controls apply throughout.

Consent and transparency

Users are informed of what is captured and may opt out; administrators can disable learning for a deployment, and for customers who embed the assistant in their own websites, learning is off by default and enabled only under an explicit contractual basis.

Personal-data removal

Every candidate example passes through automated redaction before it is stored.

Tenant isolation

No lesson crosses from one customer organisation or deployment to another.

Propagating deletion and audit

Each learning example is linked to the conversation it came from, so deleting a conversation removes the examples derived from it, and adapters are retrained on schedule so that deletions reach the weights; every released change records the failures that motivated it, the evidence, the evaluation result and the approver.

These controls should be reviewed against the data-protection law applicable to each deployment, such as the GDPR or India's Digital Personal Data Protection Act, before learning is enabled for any customer-facing deployment.

— 08

Implementation roadmap

We propose five phases (Table 5), taking roughly seven months for a single engineer and less with two working in parallel. Durations are indicative. Value begins in Phase P1, when a weekly failure report starts directing human fixes, and does not depend on later phases.

INDICATIVE DURATION PER PHASE



Value begins in P1. P3 and P4 can run in parallel once P1 is complete.

Phase	Delivers	Indicative duration	Prerequisites	Exit criterion
P0 Instrument	Rating buttons, capture of user behaviour after each answer, linking of behaviour to execution traces, a privacy-scrubbed store of learning examples	4 weeks	Evaluation harness and competency set	95% of turns have a linked trace and behaviour record
P1 Label and triage	Outcome labeller, failure categories, human review queue, weekly report of the ten most common failure groups	4 weeks	P0	At least 85% agreement with human labels on 200 turns
P2 Assisted fixes	Nightly job that drafts skill and definition changes with evaluation results attached; regression set; offline and shadow evaluation	6 weeks	P1; knowledge-layer definitions in place	At least 3 proposed changes accepted, each with a measured gain and no regression
P3 Search and memory	Continuous search-model training, re-ranker, preference memory learned from edits	7.5 weeks	P1; memory service enabled	Search accuracy improves beyond noise; fewer restated preferences
P4 Model fine-tuning	First Qwen3.8-Max adapter for the database-query agent using automatically checked rewards, then a KTO conversational adapter; adapter registry and monthly release process	7.5 weeks	P1; GPU capacity; query checker recommended	Adapter outperforms the base model with P2 improvements on the target category, with no canary regressions

Table 5. Phased implementation plan.

P0 and P1 build on the existing evaluation harness and competency set rather than duplicating them. P3 and P4 are independent of each other and can run in parallel once P1 is complete.

Start condition for P4. Adapter training begins once three conditions hold: the outcome labeller meets its agreement target; a curated, reviewed dataset exists for the target domain; and GPU capacity for training on a 2.4-trillion-parameter model has been secured. The database-query agent is the recommended first target because its correctness can be checked by running the query, which yields a clean reward without depending on noisy approval signals.

The quickest early win requires no automation. From the end of P1, the weekly failure report goes to the people who own the business definitions and skill files; even manually applied fixes, prioritised by real user impact, will close gaps faster than before.

09

Discussion and limitations

9.1 Measuring the programme

The primary outcome measure is **time to fix**: the number of days between a failure pattern first appearing in production and a verified fix reaching users. In a static assistant this quantity is undefined, because failures are not systematically detected. Table 6 lists the full set of programme metrics.

Metric	Definition	Desired direction
Time to fix	Days from a failure group’s first appearance to a released fix	Down, towards under 2 weeks
Failure rate	Turns labelled failure per 1,000 turns	Down
Repeat-failure rate	Fixed failure groups recurring within 30 days	Down
Competency accuracy	Correct answers per category on the competency set, measured nightly	Up

Metric	Definition	Desired direction
Draft edit share	Proportion of generated text rewritten by users	Down
Preference restatement	Sessions in which users repeat a previously stated preference	Down
Sycophancy probe pass rate	False-premise prompts correctly challenged	Stable or up
Audit coverage	Released changes with complete evidence and approver recorded	100%

Table 6. Programme metrics.

9.2 Limitations

Several limitations should be stated plainly. First, the gains cited in Section 2 were measured on public benchmarks and may not transfer to enterprise settings; each method must be validated on our own competency set before its benefit is claimed. Second, the outcome labeller is itself a model, and its errors propagate into every downstream stage; the uncertain class and human review mitigate but do not remove this. Third, feedback volume may be low in small deployments or niche domains, delaying the point at which failure groups become statistically meaningful. Fourth, adapter training at the scale of Qwen3.8-Max carries substantial compute cost, and whether the gains justify that cost over the cheaper layers is an empirical question the P4 exit criterion is designed to answer. Fifth, adapters must be retrained whenever the base model is upgraded. Finally, the scoping rule deliberately forgoes lessons that could be learned from restricted content; we regard this as the correct trade-off, but it does limit what the system can learn in highly restricted domains.

9.3 Threats to validity

Improvements measured during a canary rollout may partly reflect novelty or selection effects in the canary group. Guardrail tests such as the sycophancy probe are fixed sets that the system could, in principle, overfit to; they should be refreshed periodically and kept separate from any data used to propose changes.

Conclusion

The enterprise assistants that prove most valuable will not necessarily be those built on the largest models, but those that learn fastest from the people who use them without eroding the trust that makes them usable. Corporate GPT is well positioned to be such a system. It already traces every turn, defines its behaviour in reviewable instruction files, measures quality against human-verified answers, enforces permissions at the level of individual documents, and hosts its own model, which places every layer of the system, including the weights, within reach of learning while keeping data inside the organisation.

This paper has described the missing piece: a loop that captures usage, labels outcomes, diagnoses failures, routes each lesson to the cheapest adequate layer, and allows nothing to reach users without evidence and approval. Much of the expected improvement should come from business definitions, search, skills and memory; fine-tuning of Qwen3.8-Max is reserved for gaps those layers cannot close, and is designed from the outset to be parameter-efficient, verifiable and reversible. The framework is intended to be adopted incrementally: instrumentation and labelling deliver value on their own, and each later phase must earn its place through measured improvement.

THE MOST VALUABLE ASSISTANTS

The enterprise assistants that prove most valuable will not necessarily be those built on the largest models, but those that learn fastest from the people who use them without eroding the trust that makes them usable.

References

1. Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., et al. (2025). GEPA: Reflective prompt evolution can outperform reinforcement learning. arXiv:2507.19457. <https://arxiv.org/abs/2507.19457>
2. Biderman, D., Portes, J., Gonzalez Ortiz, J. J., et al. (2024). LoRA learns less and forgets less. Transactions on Machine Learning Research. <https://arxiv.org/abs/2405.09673>
3. Ethayarajh, K., Xu, W., Muennighoff, N., Jurafsky, D., and Kiela, D. (2024). KTO: Model alignment as prospect theoretic optimization. arXiv:2402.01306. <https://arxiv.org/abs/2402.01306>
4. Gao, G., Taymanov, A., Salinas, E., Mineiro, P., and Misra, D. (2024). Aligning LLM agents by learning latent preference from user edits. arXiv:2404.15269. <https://arxiv.org/abs/2404.15269>
5. OpenAI (2025, 29 April). Sycophancy in GPT-4o: What happened and what we're doing about it. <https://openai.com/index/sycophancy-in-gpt-4o/>
6. Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. arXiv:2203.02155. <https://arxiv.org/abs/2203.02155>
7. Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. (2023). Direct preference optimization: Your language model is secretly a reward model. arXiv:2305.18290. <https://arxiv.org/abs/2305.18290>
8. Shenfeld, I., Pari, J., and Agrawal, P. (2025). RL's razor: Why online reinforcement learning forgets less. arXiv:2509.04259. <https://arxiv.org/abs/2509.04259>
9. Shumailov, I., Shumaylov, Z., Zhao, Y., Papernot, N., Anderson, R., and Gal, Y. (2024). AI models collapse when trained on recursively generated data. Nature. <https://www.nature.com/articles/s41586-024-07566-y>
10. Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. (2024). Agent workflow memory. arXiv:2409.07429. <https://arxiv.org/abs/2409.07429>
11. Zhang, Q., Hu, C., Upasani, S., et al. (2025). Agentic context engineering: Evolving contexts for self-improving language models. ICLR 2026. <https://arxiv.org/abs/2510.04618>
12. Qwen Team, Alibaba Group (2026). Qwen3.8-2.4T-A95B model card. Hugging Face. <https://huggingface.co/Qwen/Qwen3.8-2.4T-A95B>

Glossary

Term	Meaning
Access-control list (ACL)	The list of users and groups permitted to see a document

Term	Meaning
Adapter (LoRA)	A small set of additional weights trained on top of an unchanged base model; can be added or removed without retraining the model
Canary release	Releasing a change to a small group of users first, to catch problems before wider rollout
Competency set	A curated set of real user questions with answers verified by people, used as the reference for measuring correctness
Embedding	A numerical representation of text that lets search find passages by meaning rather than exact words
Feature flag	A switch that turns a capability on or off for chosen users or deployments without a new software release
Hard negative	A search result that looked relevant but did not answer the question; valuable for training search models
Knowledge layer	A machine-readable record of what a company's data means: business definitions, valid table combinations and entity identities
Mixture of experts	A model architecture in which only a subset of the parameters is used for each token, reducing the cost of very large models
On-policy reinforcement learning	Training in which the model learns from feedback on its own freshly generated outputs
Retrieval-augmented generation (RAG)	Answering questions by first retrieving relevant documents or data, then generating an answer grounded in them
Shadow evaluation	Running a candidate system on a copy of live traffic without showing its outputs to users
Skill file	A plain-text file of instructions that defines how one of the assistant's agents behaves
Sycophancy	A model's tendency to agree with or flatter the user at the expense of accuracy
Trajectory	The complete record of one interaction: question, agent choices, tool calls, retrieved content, answer and the user's reaction

