

Indefinitely Active Weight Updates in Large Language Models

Executive summary

An LLM whose weights continue updating indefinitely is no longer best understood as a fixed software artifact. It becomes a **stateful learning process**: model behavior at time t depends not only on code and a named checkpoint, but on the ordered data stream, optimizer state, update policy, random state, filtering rules, and every prior update. Classical stochastic approximation provides a foundation for sequential parameter updates, but once the underlying data distribution moves, ordinary convergence to one fixed optimum becomes a questionable goal; the problem shifts toward **tracking a moving target while controlling variance and forgetting**. ¹

The central technical trade-off is the familiar **stability-plasticity problem**. Aggressive updates improve adaptation to new information but increase interference with previously learned capabilities. Conservative or decaying updates improve stability but eventually make adaptation sluggish. Replay, regularization such as Elastic Weight Consolidation, constrained updates, and parameter-efficient adaptation can reduce catastrophic forgetting, but none removes the underlying trade-off. Continual-pretraining work on aligned LLMs also shows that further training can degrade safety alignment and induce undesirable outputs, meaning “learning more” is not monotonically equivalent to “getting better.” ²

Operationally, true continuous mutation of production weights is the hardest regime to make safe. It makes reproducibility, regression testing, provenance, incident reconstruction, and rollback substantially more difficult. NIST guidance for generative-AI development emphasizes model/version lineage, data provenance and integrity, continuous pipeline monitoring, regression testing after retraining or data-source changes, and human involvement for significant security decisions. Production deployment systems from major cloud providers similarly treat model promotion as a discrete, auditable event with registered versions, alarms, canaries or baking periods, and rollback—not as unconstrained mutation of a live endpoint. ³

The pragmatic architecture is therefore: **continuous learning may be reasonable; continuous uncommitted mutation of serving weights usually is not**. Keep the updater active in a shadow/candidate environment, package changes into immutable checkpoints, evaluate each candidate against retention, safety and drift gates, then promote transactionally with canary deployment and automatic rollback. Periodic batches are likely the best general-purpose compromise; true streaming updates are justified only when adaptation latency is valuable enough to pay for much stronger observability, governance, compute and failure containment.

Technical implications

A minimal online update can be written as

$$\theta_{t+1} = \theta_t - \eta_t g(\theta_t; B_t),$$

where B_t is the newly available batch or stream slice and η_t is the learning rate. In a roughly stationary environment, stochastic-approximation theory gives conditions under which repeated noisy updates can approach a stable solution. In an indefinitely operating system exposed to changing language, users, policies and source distributions, however, the effective optimum itself can move. Concept-drift research explicitly treats the target concept as changing through time, while modern non-stationary optimization evaluates algorithms by their ability to follow time-varying comparators rather than merely converge once.

1

That changes what “convergence” means. With a diminishing learning rate, parameter motion can eventually become small, which improves stability but weakens the system's ability to track later changes. With a persistent learning rate, the model can remain adaptive but generally retains an ongoing stochastic-error floor and is perpetually susceptible to new-data interference. In practice, **update frequency, learning rate, replay ratio and fraction of trainable parameters become one coupled control system**, not independent knobs.

Catastrophic forgetting is the most obvious failure mode. Neural networks can overwrite capabilities relevant to older data while fitting new data; EWC addresses this by penalizing changes to parameters judged important to previous tasks, while replay methods interleave retained examples with new observations to stabilize old behavior. Replay is effective precisely because continual learning requires both plasticity toward new information and stability with respect to old information. 4 For an LLM, the affected “old task” need not be a neat benchmark: it can be factual knowledge, reasoning behavior, formatting discipline, multilingual competence, refusal behavior, or an alignment property. Experiments on continual pretraining of aligned LLMs report forgetting and degradation of safety-related behavior, including biased or toxic generation. 5

Distribution shift is simultaneously the reason to update and a reason not to trust an update. A new cluster of examples can indicate a legitimate change in user needs, a temporary event, sampling bias, coordinated manipulation, or poisoned training data. NIST's adversarial-ML work explicitly includes poisoning of training data among major attack classes. An always-open gradient path therefore turns inputs that would otherwise influence one inference into inputs capable of influencing future behavior globally. 6

There is also an endogenous feedback problem. Once a deployed model helps create, rank or select the data that later trains it, the learning distribution is partly generated by the model itself. NIST's secure-development guidance notes that models, weights and interacting data can form closed loops that introduce manipulation risks. 7 Without source diversity, held-out reference data and explicit feedback-loop controls, such a system can amplify its own mistakes or preferences.

Model size changes the economics more than the basic logic. Training cost is jointly driven by parameter scale, data volume and available compute; scaling-law work establishes that these variables trade off rather than varying independently. 8 A very large model updated continuously at full precision across all parameters has high optimizer-state, communication and checkpoint costs. Restricting updates to adapters, low-rank matrices, selected layers or other parameter subsets can reduce that footprint and the rollback blast radius, although it also constrains what the system can learn.

Conceptually, the model should therefore be represented less as a file called `model-v7` than as:

$$M_t = (\theta_t, O_t, R_t, D_{0:t}, U, G, V_t),$$

where θ_t is model state, O_t optimizer state, R_t randomness/state, $D_{0:t}$ the ordered history of admitted data, U the update algorithm, G the safety/promotion gates, and V_t the corresponding version manifest. **The update machinery is part of the model.**

Operational, safety and governance implications

Reproducibility and versioning. For a static release, retaining the weights and software environment goes a long way toward identifying what ran. For an indefinitely updating system, meaningful replay additionally requires the exact data ordering or windows, preprocessing transformations, optimizer and scheduler state, random-state information, code/configuration versions, and source lineage. NIST specifically identifies versioning and lineage challenges around generative-AI models and their training artifacts. ⁹

Framework-level determinism is itself not free: PyTorch documentation notes that deterministic operations can be slower than non-deterministic alternatives. ¹⁰

This suggests a distinction between **bitwise reproducibility**, which may be expensive or impossible across changing hardware/software stacks, and **audit reproducibility**, in which an organization can reconstruct the inputs, state transition, evaluation and approval process that produced checkpoint C_t . For most production governance, the second is essential even when the first is unattainable.

Auditability and provenance. Every promotable state should have an immutable manifest containing at minimum the parent checkpoint, weight hash, optimizer/configuration identifier, training-data batch IDs, source/licensing metadata, preprocessing version, evaluation results, approval records and deployment timestamps. NIST recommends verifying provenance and integrity of training, testing, fine-tuning and alignment data before use. ¹¹ Model registries similarly exist to preserve model versions, metadata and lineage so deployments can be traced back to artifacts and experiments. ¹²

Safety and alignment. A conventional aligned model can regress only when somebody deploys a changed artifact. A continuously updating model has a standing pathway for regression. Since continual pretraining can erode safety properties, safety evaluations must therefore be treated as invariant checks on each candidate, not as one-time certification of the original model. ⁵ The most dangerous architecture is direct `production interaction → gradient → serving model`, because an attacker or merely unusual traffic can acquire long-lived influence. Training inputs should instead enter a quarantined path with source-trust, integrity, privacy, duplication and poisoning controls. NIST recommends continuous monitoring of training pipelines and model registries as part of secure AI development. ¹³

Rollback and failure containment. A useful update needs a transaction boundary: a parent state, candidate state, evaluation result and explicit promotion. Registered immutable checkpoints make the old state recoverable; canary or blue/green deployment exposes only a fraction of traffic to the candidate until metrics stabilize. AWS deployment documentation, for example, supports alarms during a post-deployment baking period and automatic rollback to the prior endpoint when an alarm fires. ¹⁴ NIST likewise recommends automated regression testing and retesting when models are retrained or new data sources are introduced. ¹⁵

A genuine streaming updater can approximate these properties only by creating frequent micro-checkpoints. But as the interval approaches individual batches—or individual interactions—the amount of state, evaluation workload and provenance metadata grows sharply. This produces an important operational constraint: **an update cadence materially faster than the organization's ability to evaluate, observe and reverse updates creates unvalidated model states by construction.**

Human-in-the-loop control. Human approval should be risk-tiered rather than placed on every gradient step. Routine candidates that remain inside tight, pre-approved envelopes can proceed automatically; changes involving new data sources, material safety deltas, sensitive domains, major distribution shifts or changed policy behavior should escalate to a human decision. NIST explicitly recommends considering a human in the path for significant security decisions. ¹¹ The human decision should approve a signed candidate checkpoint—not manually inspect raw gradients.

Resource costs. Static models incur primarily inference and monitoring costs after release. Periodic updating adds bursty training, evaluation and checkpointing. Streaming updating makes those expenses persistent and adds replay storage, provenance infrastructure, drift detection, candidate evaluation and checkpoint retention. Because training economics scale with both model and token budget, there is no universal break-even point. ⁸ In small, frequently updated systems, governance and evaluation can even become more expensive than the actual parameter update.

Legal, regulatory and IP exposure. As of August 2026, the EU AI Act's general-purpose AI obligations are operational. Article 53 requires GPAI providers to maintain up-to-date technical documentation covering training, testing and evaluation; maintain a copyright-compliance policy; and publish a sufficiently detailed summary of training content. Providers of GPAI models with systemic risk additionally face model-evaluation/adversarial-testing, systemic-risk mitigation, serious-incident reporting and cybersecurity duties under Article 55. ¹⁶ European Commission guidance also distinguishes minor changes from significant modifications and notes that GPAI enforcement powers began on August 2, 2026. ¹⁷ An indefinite updater consequently makes “keeping documentation up to date” a live change-management function and raises a recurring question about when accumulated change becomes a legally significant modification.

U.S. copyright questions are less mechanically settled. The U.S. Copyright Office's generative-AI training report remains identified as a pre-publication Part 3 report, while official 2026 testimony emphasizes that fair-use outcomes depend on circumstances including the works used, source of the material, purpose and safeguards, and notes the emergence of licensing markets. ¹⁸ Continuous learning therefore multiplies the number of training-data admission decisions for which source, authorization, rights reservations and license status may matter. The engineering implication is straightforward even where the law remains fact-specific: **provenance metadata has to follow the sample into the update ledger.**

Update regime comparison

The qualitative ratings below are an analytical synthesis of the continual-learning literature and model-development/deployment controls above; they are not universal constants. ¹⁹

Attribute	Static post-training	Periodic batch updates	Continuous online updates
Adaptation latency	None until a new release	Minutes to months, depending on cadence	Seconds to hours
Behavioral stability	High between releases	Medium-high with good regression gates	Low-medium unless strongly constrained
Catastrophic-forgetting exposure	None from post-release learning	Bounded to update events	Persistent
Drift responsiveness	Poor	Good for moderate-rate drift	Best for fast drift
Steady resource cost	Low	Medium / bursty	High / persistent
Auditability	High	High if versions and data manifests are immutable	Difficult without dense checkpoints and event logs
Reproducibility	Relatively straightforward	Manageable	Hardest; ordering and optimizer state become critical
Safety risk from updates	Low, though model can become stale	Moderate	Highest because the update channel is continuously exposed
Poisoning attack surface	Closed after training	Opens during ingestion windows	Effectively always open
Rollback	Simple	Straightforward if each release has a parent checkpoint	Difficult unless updates are transactionized
Human oversight	Release-level	Batch/release-level	Must be exception- or risk-triggered
Operational complexity	Low	Medium	Very high
Best fit	Stable domains; tightly controlled behavior	Most general production systems	Domains where adaptation speed materially outweighs governance and instability costs

The middle regime has an important advantage: batching creates **time for evidence**. A candidate can be exposed to a fixed regression suite, red-team tests, historical replay, safety evaluations and human review before acquiring production authority. Continuous learning removes that natural boundary unless engineers deliberately put it back.

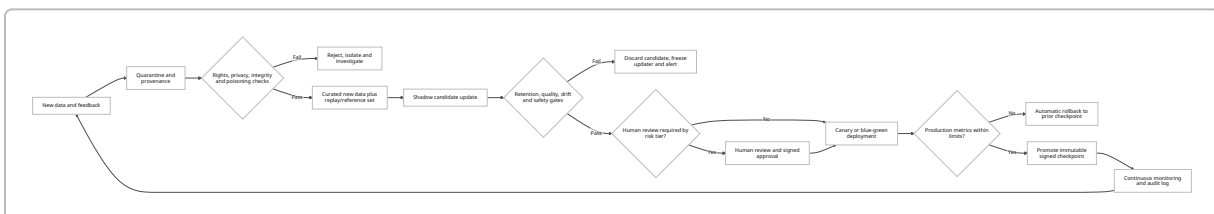
Monitoring and safeguarded update pipeline

Monitoring has to cover **changes caused by learning**, not merely ordinary endpoint uptime. Official model-monitoring guidance already treats data quality, data drift, prediction drift, model performance and related signals as alertable production metrics. ²⁰ For an indefinitely active updater, a useful minimum set is:

Monitoring family	Suggested metrics	Typical response
Retention	Change in historical benchmark score; forgetting on protected capabilities; old-distribution loss/perplexity; backward-transfer score	Block promotion on material regression
Plasticity	Recent-window loss; adaptation gain on verified new examples; time-to-adapt	Tune cadence or update magnitude
Distribution drift	Token/topic distribution divergence; embedding-distribution distance; source-mixture change; OOD rate	Quarantine or increase review
Optimization stability	Gradient norm; clipping rate; NaN/Inf count; $\ \Delta\theta\ /\ \theta\ $; loss spikes; optimizer-state anomalies	Stop updater and restore parent state
Safety/alignment	Harmful-response rate; attack success rate; jailbreak resistance; bias/toxicity measures; refusal precision/recall; policy-suite regression	Hard fail / human escalation
Data integrity	Provenance coverage; license-status coverage; duplication rate; PII/secrets detection; anomalous contribution by source/account	Reject or isolate suspect data
Operational health	Accelerator-hours/update; queue lag; tokens/sec; evaluation latency; checkpoint/storage growth	Rate-limit or batch updates
Production canary	Quality delta vs control; error rate; latency; complaint/escalation rate; task-specific KPIs	Automatic rollback

Thresholds should be defined against a stable control checkpoint and stratified by capability rather than collapsed into one average score: a tiny mean improvement can conceal a severe regression in a safety-critical slice. NIST's recommendation to log and monitor model inputs and outputs and rerun regression tests after model or data changes supports exactly this change-oriented observability model. ¹⁵

A safer architecture makes the learner persistent but keeps production promotion discrete:



The key architectural distinction is between **learning continuously** and **committing continuously**. The first can be useful. The second discards the transaction boundary needed for auditing, regression testing and recovery.

White-paper abstract

The following uses a direct, pragmatic, technically blunt engineering register rather than attempting an exact imitation of a living person's individual style.

An LLM whose weights never stop changing is not really a model release anymore; it is a stateful service with an optimizer attached, and pretending otherwise just hides the hard parts. Sequential learning is perfectly legitimate, but under a changing data distribution there is no magic convergence fairy: you are tracking a moving target, and every knob that makes the system more plastic also gives it another way to forget old capabilities, regress its alignment, absorb poisoned data, or simply wander somewhere you did not test. Continual-learning methods such as replay and regularization reduce that damage but do not repeal the stability-plasticity trade-off, and continual pretraining of aligned LLMs has already shown that safety behavior can degrade. ² Operationally, the ugly part is worse: once weights mutate continuously, “which model produced this output?” stops having a useful answer unless every update has lineage, provenance, optimizer state, evaluation results and a recoverable checkpoint. NIST guidance consequently emphasizes model lineage, provenance verification, regression testing and continuous pipeline monitoring, while production deployment systems rely on explicit versions and rollback boundaries. ²¹ The sane design is boring on purpose: quarantine incoming data, train a candidate out of band, preserve replay and reference sets, test retention and safety, promote immutable checkpoints through a canary, monitor them, and roll back automatically when they misbehave. Periodic batch updates are therefore the practical default; true streaming updates make sense only when freshness is valuable enough to pay for the extra compute, attack surface, observability, legal bookkeeping and operational pain. Keep learning continuously if the application genuinely needs it, but do not let production weights mutate continuously without a transaction boundary.

Selected primary and official sources

The foundational optimization reference is Robbins and Monro's 1951 **“A Stochastic Approximation Method,”** which formalized sequential stochastic approximation. ²² Concept drift is represented by Widmer and Kubat's **“Learning in the Presence of Concept Drift and Hidden Contexts,”** which treats changing context and target concepts in online learning. ²³ Modern non-stationary optimization framing is represented by work on dynamic regret for online optimization, where the comparator itself can evolve over time. ²⁴

For catastrophic forgetting and continual learning, Kirkpatrick et al.'s **“Overcoming Catastrophic Forgetting in Neural Networks”** introduced Elastic Weight Consolidation, while Rolnick et al.'s **“Experience Replay for Continual Learning”** provides a primary treatment of replay as a way of balancing new learning with retention. ⁴ Li and Lee's work on continual pretraining of aligned LLMs directly connects ongoing pretraining with forgetting and possible safety-alignment degradation. ⁵

For operational controls, NIST SP 800-218A provides official guidance on generative-AI model lineage, provenance, continuous training-pipeline monitoring, human involvement in significant security decisions

and regression testing after retraining or data-source changes. ²⁵ NIST's adversarial-machine-learning guidance covers poisoning as a training-time threat. ²⁶ AWS and Azure documentation provide concrete examples of model registries, monitoring, canary/baking-period controls and automated rollback mechanisms used to impose discrete operational boundaries around model changes. ²⁷

For current governance, the consolidated EU AI Act and European Commission GPAI guidance specify documentation, copyright-policy, training-content-summary, evaluation, systemic-risk and modification-related obligations applicable to general-purpose AI providers. ²⁸ The U.S. Copyright Office's AI work and 2026 congressional testimony characterize generative-AI training copyright questions as context-dependent and highlight both source/use considerations and developing licensing markets. ¹⁸

¹ ²² A Stochastic Approximation Method - Project Euclid

https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.short?utm_source=chatgpt.com

² ⁴ Overcoming catastrophic forgetting in neural networks

https://www.pnas.org/doi/10.1073/pnas.1611835114?utm_source=chatgpt.com

³ ⁷ ⁹ ¹¹ ¹³ ¹⁵ ²¹ ²⁵ nvlpubs.nist.gov

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218A.pdf>

⁵ Examining Forgetting in Continual Pre-training of Aligned Large Language Models

<https://arxiv.org/html/2401.03129v1>

⁶ NIST Identifies Types of Cyberattacks That Manipulate ...

https://www.nist.gov/news-events/news/2024/01/nist-identifies-types-cyberattacks-manipulate-behavior-ai-systems?utm_source=chatgpt.com

⁸ Scaling Laws for Neural Language Models

https://arxiv.org/abs/2001.08361?utm_source=chatgpt.com

¹⁰ Reproducibility — PyTorch 2.13 documentation

https://docs.pytorch.org/docs/stable/notes/randomness.html?utm_source=chatgpt.com

¹² ²⁷ Model Registration Deployment with Model Registry

https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html?utm_source=chatgpt.com

¹⁴ Auto-Rollback Configuration and Monitoring - Amazon SageMaker AI

<https://docs.aws.amazon.com/sagemaker/latest/dg/deployment-guardrails-configuration.html>

¹⁶ ²⁸ Consolidated TEXT: 32024R1689 — EN — 27.07.2026

<https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A02024R1689-20260727>

¹⁷ Guidelines for providers of general-purpose AI models | Shaping Europe's digital future

<https://digital-strategy.ec.europa.eu/en/policies/guidelines-gpai-providers>

¹⁸ Copyright and Artificial Intelligence | U.S. Copyright Office

https://www.copyright.gov/ai/?utm_source=chatgpt.com

¹⁹ Experience Replay for Continual Learning

<https://proceedings.neurips.cc/paper/2019/hash/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Abstract.html>

20 **Model monitoring in production - Azure Machine Learning**

https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-monitoring?view=azureml-api-2&utm_source=chatgpt.com

23 **Learning in the presence of concept drift and hidden contexts | Machine Learning | Springer Nature**
Link

<https://link.springer.com/article/10.1007/BF00116900>

24 **Adaptivity and Non-stationarity: Problem-dependent Dynamic Regret for Online Convex Optimization**

<https://jmlr.org/papers/v25/21-0748.html>

26 **AI 100-2 E2025, Adversarial Machine Learning: A Taxonomy ...**

https://csrc.nist.gov/pubs/ai/100/2/e2025/final?utm_source=chatgpt.com