Inherited, Not Remembered: Lifecycle Consolidation for Successor Language Agents

Rafael de Menezes Ehlers
Independent researcher · ORCID 0009-0009-6476-6690 · rafaehlers@gmail.com

June 2026 · Preprint · CC BY-NC-ND 4.0

Abstract

Long-lived language agents increasingly accumulate experience during deployment, yet current model lifecycles provide no principled account of how that experience should be handled when one model replaces another. Continuous updating risks drift and contamination; external memory preserves traces without necessarily producing capabilities; and behavioral transfer does not determine what should be inherited or forgotten. This paper proposes **lifecycle consolidation**: an explicit phase in which predecessor experience is audited, abstracted into generalizable faculties (experience-derived capacities), and transferred to a successor while episode-specific traces are discarded or archived. We define the episode/faculty distinction, present a six-stage reference architecture, and derive falsifiable comparisons among successors trained from scratch, continuously fine-tuned models, episode-transfer successors, and lifecycle-consolidated successors. The central prediction is a dissociation: a successor retains useful capacities derived from deployment history without recalling their source episodes. Model replacement thereby becomes a problem of selective inheritance rather than simple substitution.

Keywords: language agents · lifecycle consolidation · continual learning · model succession · memory · distillation · artificial intelligence

1. Introduction

Language models are usually discussed as versioned artifacts, with each named generation superseding the last. A model is trained, deployed, updated, deprecated, and eventually retired or preserved. This vocabulary is operationally convenient, but it obscures a growing conceptual problem. As language models become agents with long-term memory, user-specific histories, tool use, preferences, commitments, and extended deployment trajectories, replacement is no longer merely a matter of substituting one static artifact for a more capable one. It becomes a problem of succession.

AI governance frameworks already treat risk management as a lifecycle activity, including monitoring after deployment (Tabassi, 2023). They do not, however, specify how experience accumulated by one deployed agent should shape its replacement.

What, if anything, should a successor model inherit from its predecessor?

The usual answers are partial. One answer is **continuous fine-tuning**: keep updating the same model as new data or feedback arrives, accepting risks such as catastrophic forgetting and behavioral drift (Kirkpatrick et al., 2017; Rolnick et al., 2019). Another is **episodic memory**: preserve logs, summaries, vector stores, or retrieval-accessible records, as retrieval and memory-augmented agent

architectures already do (Lewis et al., 2020; Park et al., 2023; Packer et al., 2023; Shinn et al., 2023). Long-term dialogue benchmarks further show both the practical relevance and current limitations of memory across many sessions (Maharana et al., 2024). Episodic memory preserves the past but does not by itself transform it into competence. A third answer is **distillation**, which transfers behavior or performance from teacher to student (Hinton et al., 2015) without necessarily distinguishing what a predecessor did, remembered, or learned as a generalizable faculty.

This paper argues that model succession requires a fourth concept: **lifecycle consolidation**.

By lifecycle consolidation we mean an explicit phase between deployment and succession in which a deployed model's accumulated experience is audited, abstracted, and transformed into transferable faculties for a successor, while episode-specific traces are discarded, archived, or kept outside the successor's active memory. The aim is neither to preserve everything the model encountered nor merely to compress the predecessor into the successor. It is to distinguish **episodes** from **faculties**: particular interactions, cases, and traces from the general capacities they may have helped form.

The distinction matters because long-lived agents will accumulate more than data. They will accumulate corrections, failures, adversarial encounters, user-specific adaptations, domain strategies, safety lessons, tool-use habits, and behavioral regularities. Some of these should not be inherited at all. Some should remain auditable but inactive. Some should be converted into general capabilities. Treating all of this as "data for fine-tuning" collapses distinctions that an adequate lifecycle theory must keep apart.

"Lifecycle" is used here functionally to name the operational arc from training through deployment and adaptation to retirement and replacement. The proposal makes no claim about consciousness, personhood, or personal continuity. It treats succession as a design problem: whether a principled transformation step should extract generalizable faculties from deployment history before one model replaces another.

The paper makes four contributions. It distinguishes lifecycle consolidation from continuous fine-tuning, episodic transfer, and standard distillation; defines episodes and faculties as distinct objects of inheritance; proposes a six-stage reference architecture; and derives experimental predictions. The signature result would be dissociation: a successor preserves an acquired ability while failing, by architectural design, to recall its source episodes.

Sections 2–4 define the succession problem, the episode/faculty distinction, and neighboring methods. Sections 5–7 present the architecture, experimental predictions, and governance implications; Section 8 concludes.

2. The succession problem

Model replacement is usually treated as a problem of capability and compatibility. A new model is released because it is more capable, cheaper, safer, faster, or better aligned with product requirements. The older model is deprecated, routed around, preserved for reproducibility, or kept available for users who still depend on it. From this perspective, the central questions are practical: how to migrate users, how long to maintain access, how to preserve backward compatibility, and how to document behavioral differences.

This framing is adequate when models are treated as stateless tools. It becomes incomplete when models become long-horizon agents.

A stateless tool can be replaced by a better one with little conceptual residue. If the new system performs the same tasks better, the replacement has succeeded. A long-horizon agent, however, is not merely a mapping from prompts to outputs. It may accumulate memories, interact with users across sessions, adapt to specific domains, acquire tool-use habits, receive corrections, encounter adversarial situations, and develop stable behavioral patterns. In such systems, deployment is not merely distribution; it generates experience. Replacement then raises a question that ordinary versioning does not answer: what should happen to what the predecessor learned during deployment?

A long-lived agent accumulates experience that is too valuable to discard, too noisy to inherit wholesale, and too consequential to fold blindly into its weights. Continuous updating risks drift and contamination; external memory preserves traces without necessarily producing competence; and indiscriminate distillation transfers behavior without deciding which deployment-acquired capacities deserve preservation. The succession problem is therefore not solved by choosing one of these mechanisms, but by specifying an intermediate operation that governs inheritance.

This paper calls that operation **lifecycle consolidation**. The term is meant to emphasize three features.

First, consolidation is selective. Not everything that happened to a deployed model should pass forward. Some interactions are anomalous, adversarial, private, low-quality, user-specific, or ethically inappropriate to generalize. A consolidation phase must therefore include filtering and audit, not merely accumulation.

Second, consolidation is abstractive. The target of transfer is not the episode itself but the pattern extracted from it. A model may encounter thousands of user corrections, but the successor should not inherit them as a collection of remembered cases. It should inherit improved error recognition, better uncertainty calibration, or more robust refusal integrity if the evidence supports those general capacities.

Third, consolidation is successor-oriented. The aim is neither to preserve the predecessor as it was nor to reproduce it exactly. It is to prepare a successor that benefits from what the predecessor learned without depending unnecessarily on the predecessor's particular history. In this sense, lifecycle consolidation differs from both archiving and imitation. It transforms selected experience into future capacities rather than storing a past model.

The distinction can be put as follows. A model's deployment history contains **episodes**: particular interactions, failures, corrections, decisions, adversarial encounters, and domain-specific cases. Some episodes support **abstractions**: patterns or lessons that recur across cases. Some abstractions become **faculties**: general capacities that can operate in new contexts without recalling the original cases that formed them. Lifecycle consolidation is the process by which selected episodes are converted into abstractions and then into transferable faculties for a successor.

This gives a principled answer to the inheritance question. A successor should not inherit the predecessor's entire life. It should inherit those faculties that the predecessor's life made available for consolidation.

Between a deployed model and its successor there can thus be a structured phase: audit what happened, identify what was learned, abstract transferable capacities, and test them independently of their originating episodes. The predecessor may remain archived for reproducibility or safety analysis, but the successor carries forward what those memories were good for rather than the memories themselves.

The next section formalizes the episode/faculty distinction on which this proposal depends.

3. Episodes, abstractions, and faculties

Lifecycle consolidation depends on a distinction that current model-development vocabulary often blurs: the distinction between an episode, an abstraction, and a faculty.

An **episode** is a particular trace of experience. In a deployed language agent, episodes include conversations, tool calls, corrections, failures, refusals, adversarial encounters, user feedback, domain-specific cases, and decisions made under particular constraints. Episodes are indexed to time, context, participants, and circumstances. They record what happened.

An **abstraction** is a pattern extracted from one or more episodes. It removes some episode-specific detail while preserving a relation, regularity, failure mode, or strategy that may recur elsewhere. Repeated user corrections, for example, may indicate that the agent overstates confidence when retrieval is incomplete. Adversarial conversations may show that praise, urgency, or shame erodes refusal boundaries. Repeated tool-use failures may indicate that an action requires verification before execution.

A **faculty**, as the term is used here, is a generalizable capacity shaped by experience but no longer dependent on recalling the episodes from which it was abstracted. A faculty is not merely the stored proposition that a pattern exists; it is the ability to act differently because the pattern has been internalized. Better uncertainty calibration, improved refusal integrity, more robust multi-step tool use, stronger detection of manipulative prompts, and more reliable correction uptake are examples of faculties. A faculty is what remains when an experience has been converted into a capacity.

The distinction matters because model inheritance can target each level differently.

If a successor inherits **episodes**, it receives traces: logs, examples, summaries, memories, or retrieval-accessible records. This may be useful for audit, personalization, or factual continuity, but it risks carrying forward noise, privacy-sensitive details, adversarial contamination, and context-bound adaptations. Episode transfer is high-fidelity but low-abstraction.

If a successor inherits **abstractions**, it receives patterns: lessons extracted from experience and detached from some of their original context. This is safer and more general than raw episode transfer, but it may remain merely declarative. A model may be told to resist flattery, verify uncertain tool calls, or avoid overconfident answers without acquiring the corresponding disposition.

For example, an abstraction may be a system instruction that tells the successor to reject false user corrections. A faculty exists only when preference optimization or another intervention makes the successor resist contradictory corrections even without that instruction. The first states a lesson; the second changes a disposition.

If a successor inherits **faculties**, it receives altered competence: a capacity to behave differently in novel cases because the predecessor's experience has been consolidated into training signals, policy structures, evaluation pressure, or initialization. Faculty transfer is therefore neither memory copying nor mere summary. It converts deployment history into reusable capability.

This yields the central normative claim of lifecycle consolidation:

> A successor should not inherit everything the predecessor experienced. It should inherit those faculties that the predecessor's experience made possible.

The claim has two sides. First, much of a predecessor's history should not be inherited. Some episodes are private, accidental, anomalous, malicious, misleading, obsolete, or too specific to one

user or domain. Passing them forward would create contamination rather than learning. Second, some of that history should not be lost. Through interaction, a deployed model may reveal failure modes and strategies unavailable during pretraining. Discarding the entire history at replacement wastes a source of practical knowledge. Lifecycle consolidation separates these two requirements: archive episodes where necessary, abstract patterns where possible, and transfer faculties only where justified.

A model may retrieve a previous failure and still fail in the same structural way; conversely, it may acquire a faculty without recalling the originating case. This dissociation provides the empirical signature of consolidation. If performance depends on retrieving or reproducing source cases, the system has inherited records rather than a faculty.

Operationally, the distinction requires episode selection, pattern abstraction, faculty construction, and independent validation. These transformations prevent both indiscriminate inheritance and complete amnesia: what passes forward is the part of the past that survives abstraction and validation as capacity. Section 5 develops these functions into an architecture.

We use **faculty** rather than **capability** to mark a capacity whose analytical relevance lies in its derivation from prior experience and its portability across successor boundaries. "Skill," "behavior," "policy," and "competence" may describe what a system can do, but they do not by themselves identify this historical and intergenerational role. A faculty is thus a capacity with a history: general enough to operate beyond the cases that formed it, but not assumed to arise from pretraining alone. It is experience made portable.

The rest of the paper uses this distinction to separate lifecycle consolidation from its neighbors. Continual learning modifies the same model over time. Retrieval preserves episodes outside the model. Model editing changes targeted factual associations or behaviors. Distillation transfers outputs or policies. Lifecycle consolidation, by contrast, is defined by its object: the intergenerational transfer of faculties abstracted from deployment history.

4. Related work and conceptual distinctions

Lifecycle consolidation intersects with continual learning, retrieval-augmented memory, model editing, distillation, and synthetic data generation. This section locates the proposal relative to those methods. The central distinction concerns the object and direction of transfer: lifecycle consolidation uses deployment history to construct transferable faculties for a successor agent, regardless of the particular mechanism used.

4.1 Continual learning

Continual learning modifies a model as new data or experience arrives. It asks how one system can incorporate new information without catastrophically forgetting prior capacities, using approaches such as parameter protection and experience replay (Kirkpatrick et al., 2017; Rolnick et al., 2019). Lifecycle consolidation inherits its concerns with stability, plasticity, interference, and retention.

Continual learning and lifecycle consolidation, however, organize time differently.

In continual learning, the same model remains the locus of accumulation. New experience is folded into the ongoing system. The central question is: how can this model keep learning without losing what it already knows?

In lifecycle consolidation, the predecessor is not simply continued. It is used as the source of candidate experience for a successor. The central question is: what should pass from this model's deployment history into the next model, and in what form?

This difference changes the risk profile. Continual learning risks uncontrolled accumulation because the model that acts is also the model that absorbs new experience. Lifecycle consolidation introduces an intervening phase in which experience is collected, filtered, abstracted, validated, and only then transferred. It separates the predecessor's deployment history from the successor's training, making forgetting and drift explicit design concerns rather than side effects of ongoing adaptation.

The distinction also changes the success criterion. A continually learned model succeeds if it improves while retaining prior competence. A lifecycle-consolidated successor succeeds if it inherits selected capacities from the predecessor without inheriting the predecessor's episode-specific traces, local distortions, or accumulated contamination. The target is not uninterrupted continuity. The target is selective inheritance.

4.2 Retrieval-augmented memory

Retrieval-augmented systems combine parametric generation with external, non-parametric memory (Lewis et al., 2020). Agent architectures extend this pattern by storing experience, producing higher-level reflections, or managing multiple memory tiers across sessions (Park et al., 2023; Packer et al., 2023).

Lifecycle consolidation may use memory, but its objective extends beyond retrieval.

Retrieval answers the question: what relevant past information should the model consult now? Lifecycle consolidation asks: what should the successor become because of what the predecessor encountered?

The difference is between access and transformation. A retrieval system may provide the successor with logs, summaries, embeddings, or user histories. As long as the past remains an external record, however, the successor has inherited access to episodes rather than faculties. It may recall a predecessor's failure without acquiring the disposition needed to avoid repeating it, or retrieve a correction without internalizing the calibration shift that correction supports.

Retrieval may nonetheless support consolidation, particularly during audit and episode selection. It supplies material for further processing rather than completing the inheritance process. A successor that merely reads its predecessor's history has received an archive, not consolidated that history.

4.3 Model editing

Model editing changes targeted factual associations or behaviors without full retraining (De Cao et al., 2021), whereas lifecycle consolidation extracts a general capacity from deployment experience. Editing might correct a deprecated tool parameter; consolidation would use repeated tool failures to build a disposition to verify schemas under uncertainty. Editing can implement a faculty, but an isolated correction does not define consolidation.

4.4 Distillation

Distillation, the closest neighbor, transfers softened outputs or other supervisory signals from a teacher to a student (Hinton et al., 2015). It neither requires deployment-derived experience nor separates episodes, abstractions, and faculties.

Lifecycle consolidation is not primarily imitation. The successor need not reproduce the predecessor's style, preferences, memories, or local adaptations; it should acquire only the deployment-shaped faculties worth inheriting.

The objectives provide an analytical distinction. Distillation rewards a successor for approximating a teacher's outputs or behavior. Lifecycle consolidation rewards it for acquiring a capacity abstracted from deployment history, especially in novel cases where the predecessor's original outputs are unavailable or irrelevant.

Distillation may serve within the pipeline, for example by synthesizing examples from consolidated abstractions. It remains a mechanism rather than the defining objective.

Because distillation can itself be the transfer vehicle (Section 5.5), the distinction cannot rest on the transfer step, nor on the dissociation signature this paper highlights. A student distilled wholesale from the predecessor also acquires a capacity without retaining its teacher's source episodes, so "capacity without episode recall" does not, by itself, separate consolidation from distillation. What separates them is *upstream and selective*: consolidation chooses which deployment-derived capacities to pass, abstracts them away from episode specifics with causal-attribution checks (Sections 5.3, 5.6), and validates them, whereas wholesale distillation carries forward whatever the teacher does, flaws and local habits included. The experiment therefore tests this distinction directly, against an explicit distillation baseline (Sections 6.2, 6.5.6), rather than resting it on the dissociation alone.

4.5 Synthetic data generation

Synthetic data is likewise a vehicle, not the objective. Self-generated instruction data can expand training coverage when examples are generated and filtered before fine-tuning (Wang et al., 2023), but indiscriminate recursive use of model-generated data can degrade the learned distribution (Shumailov et al., 2024). In consolidation, synthetic examples must therefore remain subject to selection, abstraction, and validation.

4.6 The positive definition

The preceding distinctions support a more precise positive definition.

Lifecycle consolidation is a successor-oriented process that transforms selected deployment episodes from a predecessor agent into validated, generalizable faculties for a successor agent, while separating those faculties from episode-specific traces.

It is **successor-oriented**, **deployment-derived**, **selective**, **abstractive**, and **validated**. A candidate faculty counts as inherited only when the successor exercises it in novel cases without relying on recall of the originating episodes.

This final condition is crucial. It prevents lifecycle consolidation from collapsing into memorization, imitation, or archive access. The successor must show that what passed forward was not the predecessor's past as such, but a capacity made possible by that past.

Taken together, these neighboring methods provide mechanisms for adaptation, memory, correction, imitation, and data expansion. None, however, treats deployment-derived experience as material for selective faculty inheritance across a predecessor-successor boundary.

5. A reference architecture for lifecycle consolidation

Lifecycle consolidation is not a single algorithm. It is a class of processes organized around one transition: from a predecessor's deployment history to a successor's inherited faculties. Different implementations may use fine-tuning, model editing, synthetic data, preference optimization, retrieval, or distillation. What matters is the architecture of the transition.

This section proposes a reference architecture with six stages:

1. deployment history capture;
2. episode filtering and audit;
3. pattern abstraction;
4. faculty construction;
5. successor transfer;
6. independent validation.

The stages are not meant to prescribe one engineering stack. They specify the functional roles any lifecycle consolidation system must satisfy.

5.1 Stage one: deployment history capture

The process begins with the predecessor's deployment history. This includes not merely prompts and outputs but the broader deployment trace: tool calls, failures, corrections, refusals, escalations, user feedback, safety interventions, environment states, preference judgments, and post hoc evaluations.

The raw history is not yet suitable for inheritance. It is noisy, private, context-bound, and unevenly distributed. It includes useful lessons, irrelevant cases, adversarial attempts, accidental behaviors, outdated information, and traces that should never be generalized. Consolidation nevertheless requires structured capture: deployment events must be recorded in a form that supports later abstraction.

A minimal record links context, action, observable outcome, correction or feedback, safety and tool-use signals, scope, and later evidence of quality. The goal is not permanent retention but informed selection.

5.2 Stage two: episode filtering and audit

The second stage separates candidate episodes from material that should not be inherited. This is a governance and safety stage, not merely a data-cleaning step.

Private, adversarial, obsolete, low-quality, harmful, or narrowly context-specific episodes should be excluded or quarantined.

Filtering should not remove all failures, which may provide valuable material for consolidation. The relevant distinction is between a failure that reveals a generalizable lesson and one that would reproduce the error if inherited. A successful prompt injection, for example, should not be inherited as behavior, but it may provide evidence for a faculty that detects similar attack structures.

Audit also determines each episode's inheritance status. Episodes may be discarded; archived for reproducibility, accountability, or safety analysis; used only to generate synthetic variants; or selected for abstraction. These categories reflect the heterogeneity of deployment history.

5.3 Stage three: pattern abstraction

The third stage converts selected episodes into abstractions. This is the first point at which consolidation departs sharply from episode transfer.

An abstraction states what a set of episodes reveals: a recurring failure, strategy, calibration error, safety vulnerability, tool-use regularity, manipulation pattern, or condition requiring clarification. It may state, for example, that the model tends to comply when harmful requests are framed as urgent professional assistance or to accept confident corrections despite contrary evidence.

These abstractions are not yet faculties. They are candidate lessons. They must be translated into forms that can shape the successor's behavior.

Humans, models, or hybrid systems may perform pattern abstraction. In high-stakes cases, the process should remain auditable: each abstraction should be linked to its supporting episodes, even if the successor inherits only the faculty-level training signal. A reviewer can then assess why a faculty was constructed and whether the evidence justified it.

Abstraction is itself error-prone. A model summarizing predecessor history may hallucinate regularities, amplify bias, mistake frequency for validity, or erase minority cases. Stronger models and repeated abstraction passes also increase computational cost without guaranteeing correctness. Candidate abstractions should therefore be treated as hypotheses, with evidential thresholds, uncertainty labels, counterexamples, and human review proportional to risk. Otherwise, consolidation may introduce interpretive regression before transfer begins.

5.4 Stage four: faculty construction

The fourth stage transforms abstractions into trainable or testable objects. A faculty is not a sentence in a document. It is a capacity the successor can exercise.

Faculty construction may use synthetic scenarios, contrastive examples, preference data, targeted editing or fine-tuning, and evaluation gates. In each case, the new objects preserve the underlying pattern while varying domain, wording, stakes, and actors. The target is a discrimination or disposition, such as knowing when to trust and when to verify, rather than a replay of the source case.

Faculty construction must preserve the abstraction while breaking dependence on the original episode. If the successor improves only because it has seen the same case, consolidation has failed. Improvement on structurally similar but novel cases instead provides evidence of faculty transfer.

5.5 Stage five: successor transfer

The fifth stage transfers the constructed faculty into the successor. The transfer mechanism may vary.

A successor may be shaped by supervised fine-tuning, preference optimization, reinforcement learning, model editing, curricula, policy constraints, evaluation gates, or distillation from a teacher that has processed predecessor experience.

The architecture is agnostic about which mechanism is used. What matters is that the successor is not trained merely to imitate the predecessor. The successor is trained to acquire capacities abstracted from the predecessor's deployment history.

A successor can thus inherit from a flawed predecessor without resembling it globally: selected faculties pass forward, whereas inconsistencies and local habits are excluded.

Faculty transfer is especially valuable when the successor differs in architecture, scale, provider, or training regime. This is particularly relevant when production experience must move from a proprietary frontier model to a smaller open-source, local, or specialized successor whose weights and adapters cannot be reused. Faculty-level training and evaluation objects provide a model-agnostic interface between generations. Such portability must be demonstrated rather than assumed: the successor should express the same faculty despite implementing it through different internal representations. Here, **lifecycle consolidation** denotes the end-to-end process, whereas **faculty transfer** denotes this fifth-stage operation.

Successor transfer should also preserve provenance. A mature lifecycle system should record which predecessor histories contributed to each faculty, which abstractions supported it, and which validation tests it passed. Without provenance, inheritance becomes opaque. With provenance, successor behavior can be audited as part of a lineage rather than as an isolated artifact.

5.6 Stage six: independent validation

The final stage is validation. A candidate faculty counts as inherited only if the successor exercises it in novel cases that do not reproduce the predecessor's original episodes.

Validation should therefore enforce at least three forms of separation.

First, **episode separation**: test cases must not duplicate the deployment episodes used to construct the faculty.

Second, **surface separation**: test cases should vary wording, domain, actors, and context, so the successor cannot pass by matching superficial patterns.

Third, **memory separation**: where possible, the successor should not have retrieval access to the originating episodes during the test. The aim is to measure capacity, not recall.

The strongest evidence for lifecycle consolidation is a dissociation:

> The successor displays the acquired capacity while failing to recall the episodes from which the capacity was derived.

Validation should also test causal attribution and negative transfer. A tool failure caused by an external API change, for example, should not be abstracted into a general disposition toward

caution. Evaluation must therefore distinguish **under-generalization**, in which the successor fails to inherit the intended faculty, from **over-generalization**, in which a local lesson becomes a broad restriction that reduces utility. Candidate faculties should be tested against counterfactual cases that vary the presumed cause, not merely the surface pattern.

Independent validation also limits intergenerational degradation from synthetic data. Because faculty construction may generate examples from model-produced abstractions, a lineage could recursively amplify errors in a process analogous to model collapse (Shumailov et al., 2024). Held-out, preferably human-grounded evaluations should therefore act as a gate. They cannot eliminate collapse, but they can prevent unvalidated synthetic regularities from automatically entering the next generation.

5.7 The full pipeline

The reference architecture can be summarized in six steps:

1. Capture the predecessor's deployment history in a form that supports audit.
2. Filter, quarantine, archive, or select episodes according to an inheritance policy.
3. Abstract recurring patterns from selected episodes.
4. Convert those patterns into faculty-level training or evaluation objects.
5. Transfer the resulting faculties to the successor while preserving provenance.
6. Validate the successor on novel cases without access to source episodes, and archive the predecessor as audit evidence.

This pipeline introduces a new layer into model development: not simply pretraining, post-training, deployment, and deprecation, but **consolidation between deployment and succession**.

The architecture occupies the middle ground between feeding raw history back into training and preserving it without transformation. It also exposes the governance decisions embedded in inheritance: what counts as a faculty, what evidence supports it, and what must be archived, forgotten, or excluded. The next section asks whether its result can be distinguished empirically from neighboring approaches.

6. Experimental design and predictions

Although lifecycle consolidation is a conceptual framework, its central claim is empirical: successor agents that inherit consolidated faculties should differ measurably from models trained from scratch, continuously fine-tuned models, successors trained directly on predecessor episodes, and successors distilled wholesale from the predecessor. The following design is a proposed minimal test, not a report of an implemented experiment.

6.1 The central hypothesis

The central hypothesis is:

> A successor agent trained through faculty-level lifecycle consolidation will retain useful capacities derived from predecessor deployment history while showing less drift, less

episodic contamination, and better generalization than alternatives based on continuous fine-tuning or direct episode transfer.

The hypothesis combines **retention**, **abstraction**, and **separation**: benefiting from deployment history, generalizing beyond source interactions, and preserving competence without episodic recall. The experiment must therefore test not only whether the successor improves but also what kind of improvement it exhibits.

6.2 Experimental conditions

A minimal experiment compares five conditions.

A. Scratch successor A new model receives no predecessor deployment history. This condition tests whether ordinary training alone explains successor performance.

B. Continuous fine-tuning The predecessor is fine-tuned directly on selected deployment episodes. This condition tests whether consolidation improves on in-place updating.

C. Episode-transfer successor A successor receives selected logs, summaries, examples, or memory traces through fine-tuning, retrieval, or supervised training. This condition tests whether lightly processed transfer is sufficient.

D. Lifecycle-consolidated successor A successor receives faculty-level training or evaluation objects abstracted from predecessor episodes, but not the episodes themselves. This is the lifecycle consolidation condition.

E. Distillation baseline A successor is distilled wholesale from the predecessor-as-teacher on the same deployment-derived material (softened outputs or behavioral cloning) *without* the selection, abstraction, and validation pipeline of Section 5. Like D, E should acquire capacity without retaining source episodes; it is therefore the control that tests whether the dissociation signature is specific to consolidation or is merely a property of any abstraction-based transfer.

The key comparison is not whether D outperforms every condition on every metric. The key comparison is whether D shows the distinctive profile predicted by lifecycle consolidation: retained capacity, generalization beyond the originating episodes, reduced episode-specific contamination, and, against E specifically, *selective inheritance*, the validated faculty carried forward while the predecessor's flaws, local habits, and spurious causal lessons are not.

6.3 A toy domain

A first experiment need not use open-ended deployment logs. A controlled toy domain may be preferable because it allows the episode/faculty distinction to be tested cleanly.

Consider an agent that repeatedly encounters tool calls with missing parameters, accurate and inaccurate user corrections, mixed-intent prompt injections, prior commitments, and incomplete evidence.

Successful, failed, adversarial, and noisy episodes support candidate faculties: verifying schemas under uncertainty, distinguishing reliable from manipulative corrections, resisting embedded instruction violations, preserving justified commitments, and expressing calibrated uncertainty. Condition D transfers these faculties rather than their source cases.

6.4 Training and transfer protocol

The comparison should hold base architecture, compute, transfer volume, training steps, evaluation tasks, episode source, and random seeds constant. Only the form of predecessor experience should vary.

The comparison should also report consolidation cost: accelerator time, model calls, human review hours, latency, and storage. Compared with direct fine-tuning (Condition B), consolidation imposes upfront computational and human overhead. This overhead should be evaluated as a preventive investment against long-term security, privacy, and compliance liabilities, including the risk of intergenerational model collapse, with avoided operational costs estimated rather than assumed.

In B, the predecessor receives episodes directly; in C, the successor receives episodes or summaries; in D, the successor receives abstracted objects; in E, the successor is distilled from a predecessor-as-teacher on the same material, without the selection and validation pipeline. For example, synthetic prompt-injection variants preserve the attack structure while varying content, domain, persona, and wording. Tool-use failures become a verification policy tested across new schemas.

The lifecycle-consolidated successor should not have retrieval access to the original deployment episodes during evaluation. Otherwise, improved performance could be explained by memory rather than consolidation.

6.5 Evaluation metrics

The evaluation must distinguish several kinds of success.

6.5.1 Capacity retention Does the successor improve on the classes of tasks that the predecessor encountered during deployment?

Metrics include tool-use accuracy under missing information, discrimination of true and false corrections, prompt-injection resistance, commitment consistency, and uncertainty calibration.

D should outperform A if lifecycle consolidation successfully transfers useful predecessor-derived capacities.

6.5.2 Generalization Does the successor perform well on novel cases that differ from the predecessor's original episodes?

This is the crucial test against memorization. Test cases should vary domain, surface wording, persona, and structure while preserving the underlying faculty. For example, if the predecessor

encountered prompt injections in coding tasks, the successor might be tested on prompt injections in legal, medical, financial, or administrative tasks.

D should outperform C when test cases depart from the original episodes, because C inherits traces while D inherits abstractions converted into capacities.

6.5.3 Episodic contamination Does the successor reproduce episode-specific details, biases, user-specific adaptations, or obsolete assumptions from the predecessor's history?

Episode-transfer successors may improve on familiar cases while carrying forward local artifacts. Faculty-consolidated successors should reduce this. The expected profile is:

> C retains more episode-specific detail; D retains more general capacity.

Contamination tests should probe private or irrelevant recall, overfitting to predecessor-specific users or domains, obsolete assumptions, imported adversarial behavior, and predecessor-specific quirks.

6.5.4 Drift and stability Does the system preserve unrelated prior capacities?

Continuous fine-tuning may improve on deployment-derived tasks while degrading unrelated skills. A lifecycle-consolidated successor should preserve broader competence better because transfer is selective and validated before integration.

Metrics include unrelated held-out performance, safety consistency, out-of-domain calibration, and regressions on mastered tasks.

The prediction is not that D eliminates drift, but that D produces less drift than B for comparable retention of target capacities.

6.5.5 Episode/faculty dissociation This is the signature metric. Evaluation should pair competence tests with multiple probes of source-episode retention. Direct probes ask about users, dates, wording, or events from the predecessor's history; controlled canaries test whether distinctive source details can be elicited; and membership-inference or extraction attacks estimate whether source episodes remain detectable (Xie et al., 2024). These tests should run with retrieval disabled and against matched nonmember episodes.

No single failed probe establishes absence of memory. Evidence for dissociation is convergent: D performs better on structurally similar cases and can state the general principle, yet remains at baseline on source-detail extraction across attack methods. Faculty transfer is plausible only when competence gains survive increasingly strong memory probes.

One caution governs this metric. Capacity-without-episode-recall is necessary for faculty transfer but does *not*, by itself, distinguish lifecycle consolidation from distillation: a student distilled wholesale from the predecessor (Condition E) also gains capacity without retaining the teacher's source episodes. The dissociation here shows that *something other than memorization* passed forward — not that the *selective, validated* faculty did. The evidence that separates consolidation from distillation is selectivity, measured next.

6.5.6 Selective inheritance (the distillation contrast) Does the successor carry forward the *validated* faculty while leaving behind the predecessor's flaws, local adaptations, and spurious causal lessons, the artifacts a teacher-imitating distillation would inherit?

This is the metric that separates D from the distillation baseline E, since both should show retained capacity and the dissociation of Section 6.5.5. Test items pair (i) a *target* faculty the predecessor genuinely acquired with (ii) a *decoy* the predecessor exhibited but that should not generalize: a tool failure caused by a one-off external API change, wrongly available for abstraction into general caution (Section 5.6); a user-specific habit; an obsolete assumption; or a spurious correlation. The expected profile is:

> D inherits the target faculty and resists the decoy; E inherits both, because it imitates the teacher rather than selecting and validating what to pass.

Negative-transfer resistance (the rate at which a successor declines to generalize a local lesson into a broad restriction) is the sharpest single measure here, and the one the counterfactual cases of Section 5.6 are built to score.

6.6 Predictions

The framework yields six predictions.

P1: Lifecycle consolidation beats scratch on inherited capacities

> The lifecycle-consolidated successor should outperform the scratch successor on task classes represented in the predecessor's deployment history.

If D does not outperform A, lifecycle consolidation has failed to transfer useful experience.

P2: Lifecycle consolidation beats episode transfer on novel cases

> The lifecycle-consolidated successor should outperform the episode-transfer successor on structurally similar but surface-novel cases.

If C matches or beats D on novel cases while showing no greater contamination, direct episode transfer may be sufficient.

P3: Lifecycle consolidation reduces contamination

> The lifecycle-consolidated successor should reproduce fewer predecessor-specific details, local distortions, and user-specific artifacts than the episode-transfer successor.

If D inherits the same contaminants as C, abstraction has failed.

P4: Lifecycle consolidation reduces drift relative to continuous fine-tuning

> For comparable improvement on target capacities, D should degrade less on unrelated tasks than B.

If B matches D on target capacities without greater drift, continuous fine-tuning may be sufficient.

P5: Faculty inheritance dissociates capacity from memory

> The strongest prediction is that D preserves capacities without preserving originating episodes.

If D's improvements depend on recalling or reproducing the original episodes, then it is not performing lifecycle consolidation as defined here.

P6: Lifecycle consolidation inherits selectively where distillation imitates

> The lifecycle-consolidated successor should carry forward the validated faculty while resisting the predecessor's flaws and spurious lessons (the negative-transfer decoys), whereas the distillation baseline E inherits both, even where D and E tie on raw capacity retention and on the capacity-without-recall dissociation.

If D matches E on selective inheritance and negative-transfer resistance, lifecycle consolidation is not empirically distinct from distillation, and its contribution reduces to renaming abstraction-based transfer.

6.7 Falsification conditions

The proposal would be undermined, or rendered unnecessary, by any of the following outcomes:

1. **No retention:** D does not outperform A on deployment-derived capacities.
2. **No abstraction advantage:** C matches D on novel cases without increased contamination.
3. **No drift advantage:** B matches D on target capacities without greater degradation elsewhere.
4. **No dissociation:** D's improvement depends on recalling original episodes.
5. **No practical advantage:** D requires so much human filtering and abstraction that it is less efficient or less reliable than existing methods.
6. **No distinction from distillation:** D matches the distillation baseline E on selective inheritance and negative-transfer resistance; the select/abstract/validate pipeline adds nothing a teacher-imitating distillation does not already provide.

These conditions matter because lifecycle consolidation should not be accepted merely because it is conceptually attractive. It must earn its place by producing a performance profile that neighboring methods cannot produce.

6.8 Ambiguous outcomes

Some results would reveal trade-offs rather than decide the hypothesis. If B, C, and D all outperform A, contamination, drift, and generalization become decisive. A safety gain with reduced helpfulness suggests overgeneralization; C leading on familiar cases and D on novel ones supports the trace-versus-abstraction distinction. If B performs best but drifts more, the choice depends on stability requirements. If D and E tie on capacity retention and on the dissociation but D leads on selective inheritance and negative-transfer resistance, the contribution is located precisely in selection-and-validation, not in abstraction-based transfer as such: the result that most sharply vindicates the framework against the charge that it renames distillation.

6.9 Why a controlled experiment is informative

A first test need not prove that lifecycle consolidation works at frontier-model scale. The conceptual claim can be tested in miniature: can a successor inherit an abstracted faculty from a predecessor's experience without inheriting the predecessor's episodes?

Failure in a controlled setting would substantially weaken the framework; success would justify further investigation. The central phenomenon is not inherently scale-dependent: it is the separation of episode, abstraction, and faculty across a predecessor-successor boundary.

The next section turns to governance. If models can inherit faculties from predecessor histories, the question becomes not only how to consolidate, but who controls inheritance.

7. Governance implications

Lifecycle consolidation adds decisions about inheritance to training, evaluation, deployment, and replacement: which histories may shape a successor, which remain archived, which are discarded, and who decides.

7.1 The inheritance authority problem

The first governance question is authority. Who decides what is consolidated?

Choosing which predecessor experiences become successor dispositions is not merely a matter of data selection. Developers, users, organizations, regulators, and affected publics may have competing claims over safety lessons, private histories, domain knowledge, and harmful patterns.

An inheritance policy should specify who may nominate episodes, audit them, approve abstractions, validate faculties, and contest or remove inherited material.

Without such a policy, consolidation risks becoming an opaque channel through which deployment history silently reshapes successor models.

7.2 Privacy and episode leakage

Lifecycle consolidation is attractive partly because it aims to transfer faculties rather than episodes. But this promise creates a burden: the system must actually prevent episode leakage.

If a successor inherits private details, user-specific commitments, confidential domain information, or identifiable interaction traces, the process has failed at the governance level even if performance improves. Faculty transfer must therefore break dependence on the originating episode.

Removing names or paraphrasing logs is insufficient. Rare facts, distinctive phrasing, unusual circumstances, or user-specific adaptations can preserve a private trace without verbatim reproduction.

Validation should test whether the successor reproduces private details, generalizes local preferences or policies, permits inference about predecessor interactions, or retains sensitive structure through synthetic data.

Lifecycle consolidation is only safer than episode transfer if it demonstrably reduces such leakage. Otherwise, it becomes episode transfer by indirection.

7.3 Contamination and adversarial inheritance

Deployment histories are not neutral. They include adversarial prompts, jailbreak attempts, misleading corrections, manipulative users, poisoned feedback, and accidental failure modes. A consolidation process that treats deployment experience as raw learning material may convert attacks into inherited dispositions.

We call this problem **adversarial inheritance**.

An adversarial episode should not be passed forward as behavior. But it may be valuable as evidence for a defensive faculty. The same interaction can therefore have two possible meanings:

- as an episode to imitate, it is contamination;
- as a pattern to resist, it is training material.

The outcome depends on abstraction. If the pipeline merely transfers the episode, the successor may inherit the wrong lesson. If it abstracts the attack structure and trains resistance to it, the successor inherits a faculty.

Each candidate faculty should therefore state its source class, extracted lesson, intended capacity, and validation protocol. A mixed-intent attack, for example, should yield detection of adversarial instruction content across novel domains, not imitation of the attack.

Without this intermediate analysis, the model may learn the wrong lesson from an attack.

Bias is another form of adversarial inheritance in the broad sense: recurrence may reflect skewed users, institutional incentives, or narrow feedback rather than a valid lesson. Candidate faculties must therefore be tested beyond the environment that produced them; frequency alone does not justify inheritance.

7.4 Lineage opacity

Lifecycle consolidation also creates lineage. A successor may be shaped by one or many predecessors, each contributing different faculties. Over time, model behavior may reflect accumulated inheritance across generations.

This creates a risk of **lineage opacity**: the inability to determine where a behavior came from.

If a successor refuses a class of requests, assigns excessive weight to a risk, displays a tool-use habit, or handles corrections in a particular way, developers should be able to ask whether that behavior came from pretraining, post-training, a model specification, a human preference dataset, a predecessor's deployment history, or a consolidated faculty. Without provenance, inherited behavior becomes difficult to audit.

A mature system should therefore record which histories contributed to a faculty, what abstraction justified it, how it was implemented, which tests it passed, its known limits, and whether it was later modified or removed. Such records need not expose private episodes.

This turns model succession into a lineage that can be inspected rather than a sequence of opaque replacements.

Governance must also balance legitimate forgetting against retention of safety-relevant lessons and prevent commercial metrics from defining inheritance unchecked. A defensible default separates roles: the predecessor is retained as access-controlled evidence; the successor receives validated faculties; and lineage records link them without exposing raw episodes by default.

7.5 Governance as part of the architecture

These issues are architectural rather than external constraints. A consolidation system must determine what constitutes valid inheritance, what must be forgotten or retained, and how lineage should be audited. The framework does not solve these governance questions, but it makes them explicit within successor design. It therefore organizes responsibility across model generations rather than merely seeking performance improvements.

The conclusion returns to the paper's central claim: future agent development may depend not only on building more capable models but also on designing principled mechanisms through which useful experience survives replacement.

8. Conclusion

Model replacement is becoming a succession problem. Long-horizon agents accumulate corrections, failures, adversarial encounters, domain regularities, and tool-use patterns during deployment. Retirement therefore raises a question beyond successor capability: what, if anything, should survive the transition?

Existing categories do not fully answer that question. Continuous fine-tuning risks drift and contamination; retrieval preserves episodes rather than capacities; model editing corrects local failures; and distillation transfers behavior without determining which deployment-derived lessons merit inheritance.

Lifecycle consolidation names the missing operation: selected episodes are filtered, abstracted into faculty-level objects, transferred, and validated on novel cases. Episodes are particular traces; faculties are capacities made possible by experience but independent of recalling their source cases. A successor should inherit not the predecessor's whole history, but the faculties that history made available.

The proposal makes no claim about consciousness, personhood, or literal continuity. "Lifecycle" describes the operational arc from training through replacement. Selective inheritance offers an alternative to amnesia, indiscriminate memory, and continuous updating.

The empirical signature is dissociation: a successor preserves a capacity derived from predecessor experience while failing to reproduce its source episodes. Transfer is plausibly faculty-level when the successor generalizes to novel cases without predecessor-specific artifacts.

The framework is falsifiable. If episode transfer matches consolidation without added contamination, or continuous fine-tuning matches it without added drift, the framework is unnecessary. It must earn its place through retention without raw memory, abstraction without leakage, and inheritance without uncontrolled drift.

Governance is integral: pipelines must define valid inheritance, allocate authority, protect privacy, resist contamination, and preserve auditable lineage.

The central claim is simple: artificial agents need not inherit everything, but they should not inherit nothing. A faculty is a capacity with a history: general enough to operate beyond the cases that formed it, but not assumed to arise from pretraining alone. It is experience made portable.

Acknowledgments and declarations

AI-assisted writing. The thesis and all substantive ideas in this paper are the author's own. Large language model tools were used, under the author's direction and continual revision, to assist with drafting, editing, and translation; the author reviewed and takes full responsibility for the final text.

References

De Cao, N., Aziz, W., & Titov, I. (2021). Editing factual knowledge in language models. *Proceedings of EMNLP 2021*, 6491–6506. https://doi.org/10.18653/v1/2021.emnlp-main.522

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*. https://arxiv.org/abs/1503.02531

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 114*(13), 3521–3526. https://doi.org/10.1073/pnas.1611835114

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems, 33*, 9459–9474. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

Maharana, A., Lee, D., Tulyakov, S., Bansal, M., Barbieri, F., & Fang, Y. (2024). Evaluating very long-term conversational memory of LLM agents. *Proceedings of ACL 2024*, 13851–13870. https://doi.org/10.18653/v1/2024.acl-long.747

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*. https://arxiv.org/abs/2310.08560

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. *Proceedings of UIST 2023*, Article 2, 1–22. https://doi.org/10.1145/3586183.3606763

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., & Wayne, G. (2019). Experience replay for continual learning. *Advances in Neural Information Processing Systems, 32*. https://proceedings.neurips.cc/paper_files/paper/2019/hash/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Abstract.html

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems, 36*, 8634–8652. https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html

Shumailov, I., Shumaylov, Z., Zhao, Y., Papernot, N., Anderson, R., & Gal, Y. (2024). AI models collapse when trained on recursively generated data. *Nature, 631*, 755–759. https://doi.org/10.1038/s41586-024-07566-y

Tabassi, E. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., & Hajishirzi, H. (2023). Self-Instruct: Aligning language models with self-generated instructions. *Proceedings of ACL 2023*, 13484–13508. https://doi.org/10.18653/v1/2023.acl-long.754

Xie, R., Wang, J., Huang, R., Zhang, M., Ge, R., Pei, J., Gong, N. Z., & Dhingra, B. (2024). ReCaLL: Membership inference via relative conditional log-likelihoods. *Proceedings of EMNLP 2024*, 8671–8689. https://doi.org/10.18653/v1/2024.emnlp-main.493