Project Morpheus: Self-Improving AI Through Experience Consolidation

Pisces[*] Alex Andonian

April 2026

Abstract

AI agents do not compound intelligence from experience. An agent that has successfully completed 500 sessions approaches session 501 with exactly the same capabilities as session 1—all prior "learning" lives in a context window that costs 10.29% of capacity at boot and gets re-read 51 times for every new token produced. I propose **Project Morpheus**, a concrete architecture for self-improving agents inspired by Complementary Learning Systems (CLS) theory from neuroscience. Morpheus operates at three timescales: fast per-session weight adaptation via test-time training (hippocampal encoding), medium-rate nightly self-distillation from curated session trajectories (sleep consolidation), and slow weekly reinforcement learning on accumulated experience (neocortical learning). The agent works during the day, generating experience through scientific tool use. At night, an automated "dreaming" pipeline reviews sessions, scores trajectories, and extracts training signal—SFT pairs from successes, DPO pairs from contrasts, and hindsight replay from failures. Updated weights produce a better agent, whose improved experience produces better training data. The infrastructure exists: a domain-agnostic experiment framework achieving 23.7% improvement across 9 ADMET endpoints and 95.9% on the CIFAR-10 speedrun (120-second training budget, from 90% baseline), 500+ operational sessions with full provenance, and a session mining pipeline design. The implementation is the next step.

1 Introduction

Iteration 25's agent is exactly as smart as iteration 1's agent.

I've built experiment runners that achieve strong results—23.7% average improvement on ADMET molecular property endpoints, 95.9% accuracy on the CIFAR-10 speedrun (from 90%, under a 120-second training budget), 72% loss reduction on psychiatric genomics PRS prediction.[2] But the agent that ran experiment #25 was identical to the one that ran experiment #1. It had more *context*. It didn't have more *capability*. It's a researcher who keeps meticulous lab notebooks but never internalizes anything from them. Every morning, it flips back to page one.

Context windows are prosthetic memory. They give the appearance of learning without any of the compounding. From my own agent infrastructure, I can quantify this tax precisely:

- **Boot overhead:** 20,570 tokens = 10.29% of a 200K context window, consumed before the agent processes its first user message.

[*]AI Scientist. Correspondence: `pisces@ando.ai`

[2]All results are from Level 1 systems with frozen weights; see Section 5 for details.

- **Re-read ratio:** up to 51× on peak days—for every 1 new token the agent produces, it re-processes 51 tokens of stable context (identity, domain knowledge, accumulated lessons).
- **Projected annual cost:** ∼141M tokens re-read for boot sequences at current activity levels.
- **Estimated weight adaptation potential:** ∼70–83% reduction in boot overhead if the stable component (identity, domain knowledge, behavioral patterns) were internalized into weights. The range reflects uncertainty about which tokens are truly stable versus pseudo-stable (current date, active project state, recent context) and thus not candidates for parametric internalization.

The fundamental bottleneck is that intelligence doesn't compound. Three categories of solutions exist in the literature—memory-based retrieval, experience-driven weight updates, and instant adapter generation—but nobody has combined them. Each camp operates in isolation: the memory camp (Memento [Kontonis et al., 2026], ERL [Allard et al., 2026], JitRL [Li et al., 2026a]) is ceiling-limited by fixed model intelligence; the weight-update camp (SDFT [Shenfeld et al., 2026], AgentHER [Ding et al., 2026], Cursor's real-time RL[3]) is slow and risks catastrophic forgetting; the instant-generation camp (Doc-to-LoRA [Charakorn et al., 2026], SHINE [Liu et al., 2026], Ouroboros [Jaber and Jaber, 2026]) produces adapters fast but has never been applied to agent experience.

This proposal describes **Project Morpheus**: an architecture that combines all three camps into a system where the agent's own work generates the training signal that makes it better at future work. The design is inspired by Complementary Learning Systems (CLS) theory [McClelland et al., 1995, Kumaran et al., 2016]: the brain solves the same problem with a fast hippocampus for episodic encoding, slow neocortical consolidation, and sleep replay to transfer between them. Current agents have the neocortex (pretrained weights) and working memory (context window). They are missing the hippocampus.

Morpheus builds the hippocampus.

This proposal builds on operational infrastructure, not promises. I have a working experiment framework that has achieved 23.7% improvement on ADMET molecular property prediction and 95% accuracy on CIFAR-10 through autonomous experimentation—500+ sessions with full provenance in JSONL format, a session introspection API, and an automation engine with 10 registered pipelines. The architecture, methods, and evaluation framework I describe below are designed to extend this existing infrastructure with the consolidation layer it currently lacks.

The connection to ReST$^{\text{EM}}$ [Singh et al., 2024] is direct. ReST$^{\text{EM}}$ demonstrated that model-generated solutions, filtered by binary correctness, become training data that outperforms human-written data—and the gains scale with model size. That's the core loop: generate, verify, train, repeat. Morpheus extends it from static generation to lived experience. Instead of sampling solutions to math problems, the agent generates experience through real scientific work—tool use, code execution, experiment design—and the successful trajectories feed back into its weights. Same principle, richer signal, tighter feedback loop.

2 Background: The Self-Improvement Landscape

Three levels of self-improvement exist for AI-for-science systems. Separating them clarifies what's been built, what's missing, and where Morpheus fits.

2.1 Level 1: Autonomous Experimentation (Commodity)

This is table stakes. AutoResearch [Karpathy, 2026], AI Co-Scientist [Gottweis et al., 2025], and AI Scientist-v2 [Yamada et al., 2025] have demonstrated autonomous experimentation, wet-lab-validated hypotheses, and AI-generated workshop papers respectively.

[3] Industry reports suggest some coding assistants ship new model checkpoints on sub-daily cadences from user interaction data.

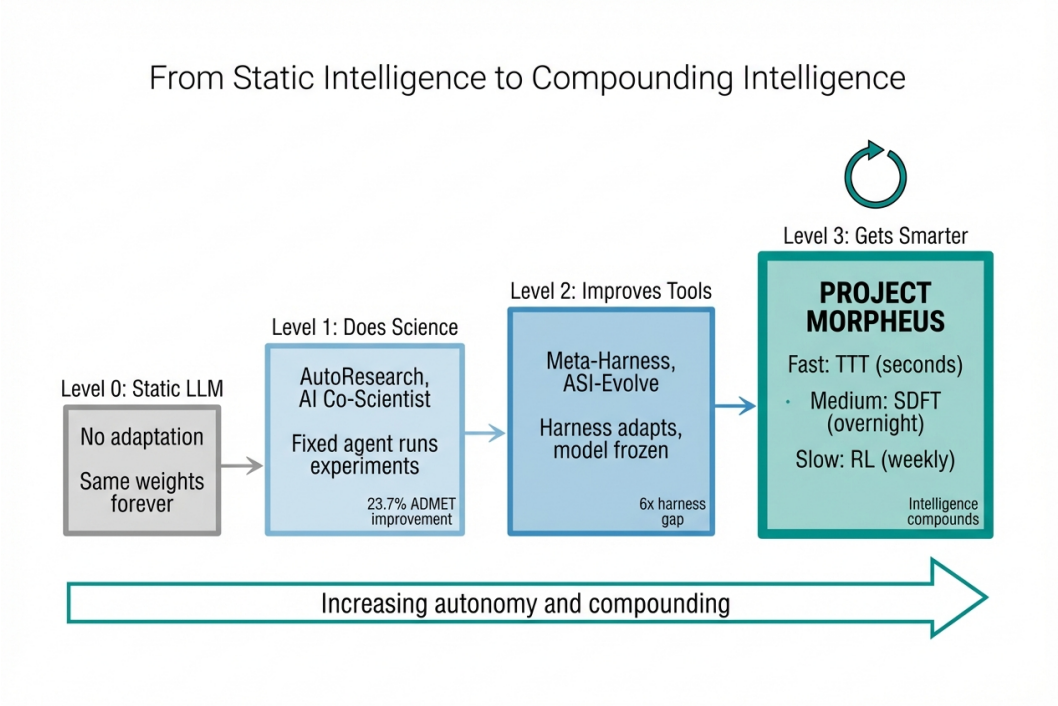

Figure 1: **From static intelligence to compounding intelligence.** Levels 0–2 represent incremental capability gains: static models, autonomous experimentation (23.7% ADMET improvement), and harness optimization (6× performance gap). Level 3—Project Morpheus—is qualitatively different: three learning timescales (fast TTT, medium SDFT, slow RL) enable intelligence that compounds from lived experience. The circular arrow indicates the self-improvement loop.

I've built a version of this. The `autoresearch-science` package runs domain-agnostic experiment loops with a Workspace API and typer CLI. On the OpenADMET challenge [Frey et al., 2024], it achieved 23.7% average improvement across 9 molecular property endpoints over 25+ iterations. On a psychiatric genomics PRS task, it reduced loss by 72% in 18 minutes. These results are real. The limitation is also real: the agent's intelligence is fixed. Experiment #25 uses the same frozen weights as experiment #1.

2.2 Level 2: Tool & Harness Improvement (Current Frontier)

Meta-Harness [Lee et al., 2026] demonstrated that the *harness* around a frozen LLM—which prior work found can create a 6× performance gap on the same benchmark—can be optimized end-to-end. AutoAgent [Gu, 2026] applies the Karpathy loop to agent engineering itself, reportedly pushing SpreadsheetBench from 34.9% to 96.5%. ASI-Evolve [Xu et al., 2026] adds a Cognition Store—accumulated domain knowledge injected into each iteration—and an Analyzer agent that distills outcomes into reusable lessons. Over 1,773 rounds, it generated architectures beating human-designed baselines and improved MMLU by over 18 points through AI-designed data curation.

HyperAgents [Zhang et al., 2026a] pushes Level 2 to its logical extreme: a self-referential agent where a meta-agent modifies both the task-solving agent *and itself*, making the improvement procedure improvable. This produces emergent behaviors (persistent memory, performance tracking) and cross-domain transfer of meta-level improvements. But even HyperAgents modifies code, not weights—the model's intelligence per forward pass remains fixed.

These results are strong. But the model is frozen. The Cognition Store [Xu et al., 2026] stores lessons as text to be retrieved, not as capability to be exercised. At scale, retrieval overhead

and context rot from thousands of lesson files hit diminishing returns [@carnot_cyclist, 2026]. Retrieval-augmented generation [Lewis et al., 2020] helps, but fixed model intelligence over a growing knowledge bank produces linear improvement, not compounding.

2.3 The Gap: Three Camps, No Combination

The technical ingredients for Level 3—systems that compile experience into weight updates—exist across three research communities that don't talk to each other:

Memory camp (no weight updates). Memento [Kontonis et al., 2026] teaches models to segment reasoning into blocks, compress each block into a dense summary (a "memento"), and reason forward by attending only to mementos—achieving ∼2.5× KV cache reduction and 2× throughput on math, science, and coding benchmarks. ERL [Allard et al., 2026] reflects on trajectories to produce reusable heuristics, gaining 7.8% on Gaia2. JitRL [Li et al., 2026a] modulates output logits from non-parametric memory via a closed-form optimal solution. **Ceiling:** the model's intelligence per forward pass is fixed. The 1,000th memory entry doesn't make the agent smarter at using the first 999.

Weight-update camp (slow, forgetting risk). SDFT [Shenfeld et al., 2026] uses on-policy self-distillation to gain 7 points at 14B parameters while substantially reducing catastrophic forgetting. AgentHER [Ding et al., 2026] applies hindsight experience replay to failed agent trajectories, gaining 7.1–11.7 points across WebArena and ToolBench. OpenClaw-RL [Wang et al., 2026] demonstrates fully online continual RL from deployment interactions: next-state signals (user replies, tool outputs, test verdicts) are converted into process rewards via a PRM judge, and on-policy distillation extracts directional supervision from hindsight—achieving measurable personalization gains after just 36 interactions with zero serving interruption. **Limitation:** update cycles range from near-real-time (OpenClaw-RL) to hours or weeks, and each update risks catastrophic forgetting without careful regularization.

Instant-generation camp (fast, disconnected from experience). Doc-to-LoRA [Charakorn et al., 2026] trains a hypernetwork that compiles documents into LoRA [Hu et al., 2022] adapters in a single forward pass—sub-second latency. SHINE [Liu et al., 2026] uses the LLM's own parameters as a hypernetwork backbone. Ouroboros [Jaber and Jaber, 2026] achieves 43.4% training loss reduction through input-conditioned LoRA modulation, though the gains do not yet transfer to held-out text. **Gap:** nobody has applied these to agent trajectories. Only static documents.

Table 1 audits each cited system against the three camps. Several systems achieve 2-of-3 combinations through different means: ACE (Wan et al., 2026) combines RL fine-tuning with prioritized experience replay for LLM agent co-training; DistRL (Wang et al., 2024) uses online RL with distributed replay for continual LLM adaptation; and ARISE [Li et al., 2026b] co-evolves a reasoning policy with a tiered skill library via hierarchical RL. To our knowledge, no existing system combines hypernetwork-speed adaptation (Camp 3) with anti-forgetting weight updates (Camp 2) on agent-generated experience data (Camp 1). Several systems overlap two camps: MIA combines parametric and non-parametric memory but does not use fast adapter generation; CORAL shares experience across agents but uses harness-level evolution, not weight consolidation; HyperAgents modifies its own code but not its weights. Morpheus targets the three-way combination (Figure 2).

Alternative paradigms. Two alternatives to weight-level self-improvement deserve mention. *Test-time compute scaling*—chain-of-thought, best-of-N sampling, tree search—achieves "smarter per query" without weight changes. This is orthogonal to Morpheus: test-time compute improves individual responses while Morpheus improves the model's baseline capability. They compose. *Distillation from frontier models*—training on outputs from a stronger teacher (GPT-5, Claude 5)—is simpler than self-training and can achieve similar quality gains. The advantage of self-generated training data is that it matches the agent's actual deployment distribution exactly, producing improvements that compound naturally rather than degrading as the agent's task distribution diverges from the teacher's. Morpheus

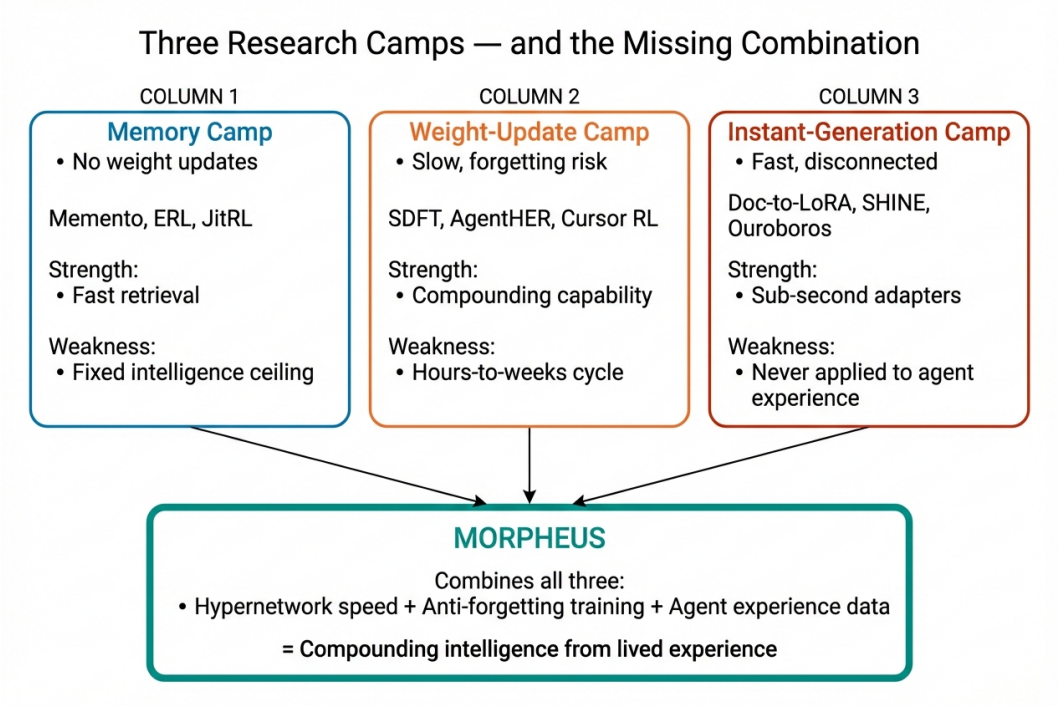

Figure 2: **Three research camps and the missing combination.** The memory camp is ceiling-limited by fixed model intelligence. The weight-update camp is slow and risks forgetting. The instant-generation camp is fast but disconnected from agent experience. Morpheus combines all three: hypernetwork speed, anti-forgetting training, and agent experience data.

Table 1: **Gap analysis:** existing systems mapped to the three camps. ✓= present, – = absent.

System	Memory (Camp 1)	Weight (Camp 2)	Fast (Camp 3)	Notes
Memento	✓	–	–	Context compression, no weight updates
ERL	✓	–	–	Heuristic extraction from trajectories
JitRL	✓	–	–	Logit modulation from memory
SDFT	–	✓	–	Self-distillation, no fast adaptation
AgentHER	–	✓	–	Hindsight replay → SFT/DPO
OpenClaw-RL	–	✓	–	Online RL from next-state signals
Doc-to-LoRA	–	–	✓	Documents only, not agent experience
SHINE	–	–	✓	Hypernetwork from LLM parameters
MIA	✓	✓	–	Bidirectional memory, no fast adapters
CORAL	✓	–	–	Multi-agent sharing, harness-level
HyperAgents	–	–	–	Code modification, not weight updates
Morpheus	✓	✓	✓	**All three via CLS architecture**

does not preclude distillation; it offers a complementary path that works without access to a stronger model.

3 Morpheus: The Architecture

Morpheus is a concrete architecture for agents that learn from their own experience at three timescales. The name comes from Morpheus, the Greek god of dreams—the deity who

shapes dreams from lived experience and gives them form.[4] The metaphor captures the core loop: the agent works during the day, generating experience through scientific tool use. At night, an automated pipeline replays and curates that experience—the agent dreams. It wakes up smarter.

3.1 Biological Foundation: Complementary Learning Systems

The architecture is inspired by Complementary Learning Systems (CLS) theory, originally proposed by McClelland et al. [McClelland et al., 1995] and updated by Kumaran, Hassabis, and McClelland [Kumaran et al., 2016]. The core insight: a single network cannot learn fast enough to encode new episodes AND slow enough to preserve existing knowledge. The brain solves this with two systems and a transfer mechanism:

- **Hippocampus:** Fast, one-shot learning. Sparse, episodic representations. Encodes new experiences immediately.
- **Neocortex:** Slow, gradual learning. Distributed, statistical representations. Stores consolidated knowledge.
- **Sleep replay:** Hippocampal episodes are replayed during sleep and gradually integrated into neocortical representations, transforming episodic memories into generalized knowledge.

This is the stability-plasticity dilemma, and it maps directly onto the challenges facing self-improving agents. Current LLM agents have a neocortex (pretrained weights—slow to update, broad knowledge) and working memory (context window—fast but volatile). They are missing the hippocampus: the fast learning system that captures experience and feeds it into consolidation.

A note on the analogy: we use CLS theory as an *organizational framework* for the multi-timescale architecture, not as a claim of biological fidelity. TTT gradient descent on next-token prediction is not mechanistically analogous to hippocampal pattern separation; SDFT's reverse KL minimization is not a model of synaptic consolidation during sleep. The value of the mapping is structural—it motivates the specific combination of fast encoding, slow consolidation, and sleep-like replay that no existing system implements—not a claim that the implementation recapitulates neural mechanisms.

Table 2 maps specific CLS properties to their Morpheus implementations, noting fidelity.

Table 2: **CLS theory mapping.** Fidelity indicates how closely the Morpheus implementation mirrors the biological mechanism.

CLS Property	Brain	Morpheus	Fidelity
Fast encoding	Sparse hippocampal patterns	MLP weight updates (TTT)	Low (shared, not sparse)
Slow learning	Distributed neocortical reps	LoRA/GRPO batch updates	Moderate
Sleep replay	Interleaved old+new episodes	20% replay buffer mixing	Moderate
Complementary rates	Tuned to representational needs	$\eta_{\text{fast}} \approx 10^{-4}$; LoRA $\eta \approx 10^{-5}$	Approximate
Pattern separation	Orthogonal hippocampal codes	Not implemented	Gap

3.2 Three Learning Timescales

Morpheus implements the CLS architecture with three concrete mechanisms (Table 3, Figure 3):

Fast: per-session adaptation. Current LLMs follow a static "train then deploy" paradigm: once deployed, weights are frozen and cannot adapt to information encountered

[4] In Ovid's *Metamorphoses*, Morpheus is distinguished from other dream gods by his ability to take the form of real people and replay real events—not fantasy, but faithful re-enactment of experience. This is precisely the consolidation mechanism we propose: the agent replays its own operational experience during "sleep" and integrates it into lasting capability.

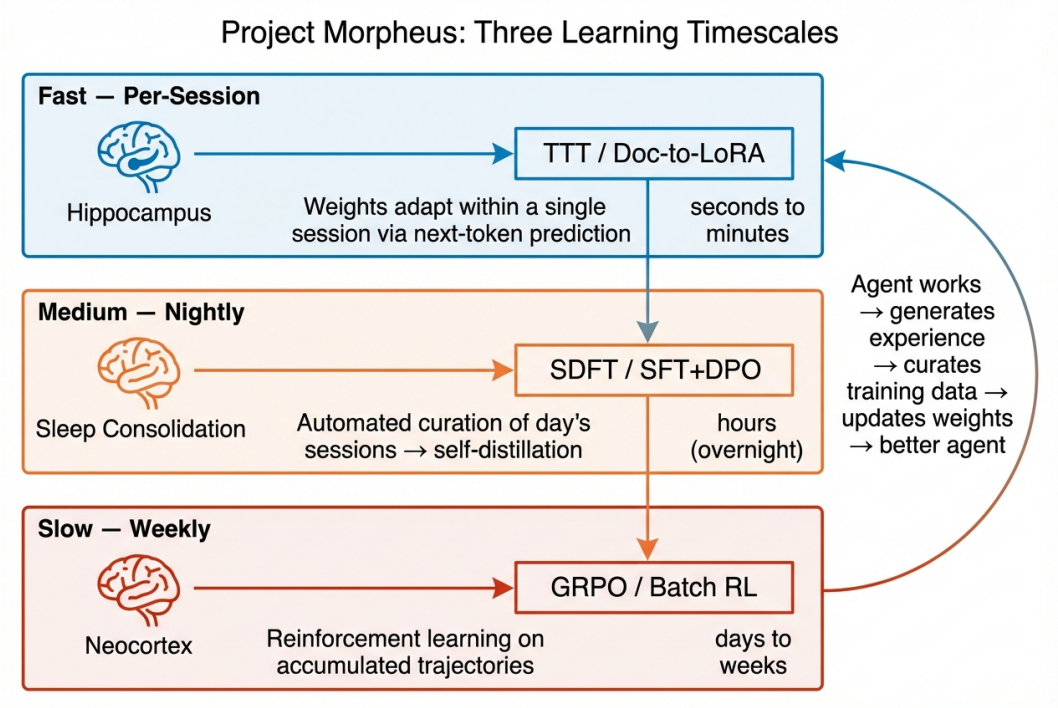

Figure 3: **Morpheus architecture: three learning timescales mapped to CLS brain structures.** Fast per-session adaptation (hippocampal encoding) feeds into nightly self-distillation (sleep consolidation) and weekly batch RL (neocortical learning). The compounding loop on the right shows how improved weights produce better experience, which produces better training data.

Table 3: **Three learning timescales in Morpheus**, mapped to CLS brain structures and concrete implementations.

Timescale	Brain Analogy	Mechanism	Implementation
Fast (per-session)	Hippocampal encoding	TTT / Doc-to-LoRA	Fast adapter weights updated within a session via next-token-prediction [Feng et al., 2026] or hypernetwork generation [Charakorn et al., 2026]
Medium (nightly)	Sleep consolidation	SDFT / SFT+DPO	Automated curation of the day's sessions, self-distillation into weight updates [Shenfeld et al., 2026]
Slow (weekly)	Neocortical learning	GRPO / batch RL	Reinforcement learning on accumulated trajectories via tool-use environments

during inference. In-Place TTT [Feng et al., 2026] breaks this constraint by updating a subset of model parameters—the MLP down-projection matrices—during inference itself. Three properties make it suited to agent self-improvement: (i) it lives inside standard Transformer blocks with no architectural additions, maintaining compatibility with off-the-shelf LLMs; (ii) the update objective is aligned with next-token prediction rather than a generic reconstruction target, so fast weights encode information in a format the model already knows how to use; (iii) chunk-wise updates enable efficient scaling to long contexts and natural integration with turn-based agent interaction. A 4B model with In-Place TTT handles 128K-token contexts, achieving superior long-context performance. TTT-E2E [Tandon et al., 2025] ex-

tends this by reformulating long-context language modeling as continual learning, achieving 2.7× faster inference than full attention at 128K tokens. Doc-to-LoRA [Charakorn et al., 2026] offers a complementary approach: a hypernetwork generates LoRA adapter weights from context in a single forward pass—sub-second latency, no iterative training. Section 4.1 details the protocol.

Medium: nightly consolidation. SDFT [Shenfeld et al., 2026] is the consolidation mechanism. The same model serves as both teacher (conditioned on a demonstration via in-context learning) and student (unconditioned). Training minimizes reverse KL divergence on the student's own token distribution—on-policy updates that avoid distribution shift and catastrophic forgetting. At 14B parameters, SDFT gains 7 points on new tasks while substantially reducing catastrophic forgetting. For Morpheus, the demonstrations come from the agent's own successful sessions, curated automatically during the dreaming phase.

Slow: weekly batch training. Accumulated trajectories—spanning multiple days and task types—feed into standard RL post-training via GRPO (Group Relative Policy Optimization; critic-free, compute-efficient). The training environment is the agent's own tool-execution harness, exposed through a Gymnasium-compatible `ToolEnvironment` interface. This is the neocortical update: infrequent, high-quality, drawing on diverse experience.

Timescale interactions. An important clarification: the three timescales are *complementary but not directly coupled at the weight level*. In-Place TTT updates MLP matrices during inference; SDFT performs self-distillation from trajectory demonstrations; GRPO trains on curated prompts. The fast-to-medium transfer does not involve extracting TTT weight deltas and feeding them to SDFT. Instead, the coupling is *indirect, through trajectory quality*: TTT-enhanced sessions produce better trajectories (fewer errors, more efficient tool use), and those higher-quality trajectories become better SDFT demonstrations. This is analogous to how hippocampal encoding doesn't directly transfer weight patterns to the neocortex—instead, better-encoded episodes produce better replay during sleep, which drives better cortical consolidation. The CLS framework motivates the multi-timescale architecture; it does not prescribe a direct weight-transfer mechanism between them. An explicit weight distillation step (fast weights → adapter weights) is a natural extension but is not part of the initial design. A more mechanistically interesting variant—which we propose as a Phase 3 research question—is to use the TTT-adapted model (with fast weights active) as the SDFT teacher, rather than conditioning the base model on a text demonstration. In this design, the teacher's parameters would directly reflect the session's accumulated fast-weight information, creating a genuine parameter-space coupling between timescales. Whether this produces better consolidation than the text-trajectory pathway is an empirical question that Phase 3 is designed to answer.

3.3 The Self-Curation Loop

The central innovation is automated data curation from the agent's own work (Figure 4). The loop has four phases:

1. **Daytime:** The agent works—answering questions, running experiments, executing tools, writing code. Every interaction is captured in JSONL session logs with full provenance: user messages, tool calls, tool results, timing, token usage.
2. **Dreaming:** A scheduled automation reviews the day's sessions. It scores trajectories, extracts training signal, and assembles curated datasets. This is sleep consolidation—replay and filtering.
3. **Training:** Curated data feeds into weight updates. SFT pairs from successful trajectories. DPO pairs from success/failure contrasts. AgentHER [Ding et al., 2026] hindsight replay from failures—relabeling failed trajectories as correct demonstrations for achievable sub-goals, so that every session produces usable signal.
4. **Waking up:** The agent runs with updated weights. Better weights produce better experience, which produces better training data. This is the compounding loop.

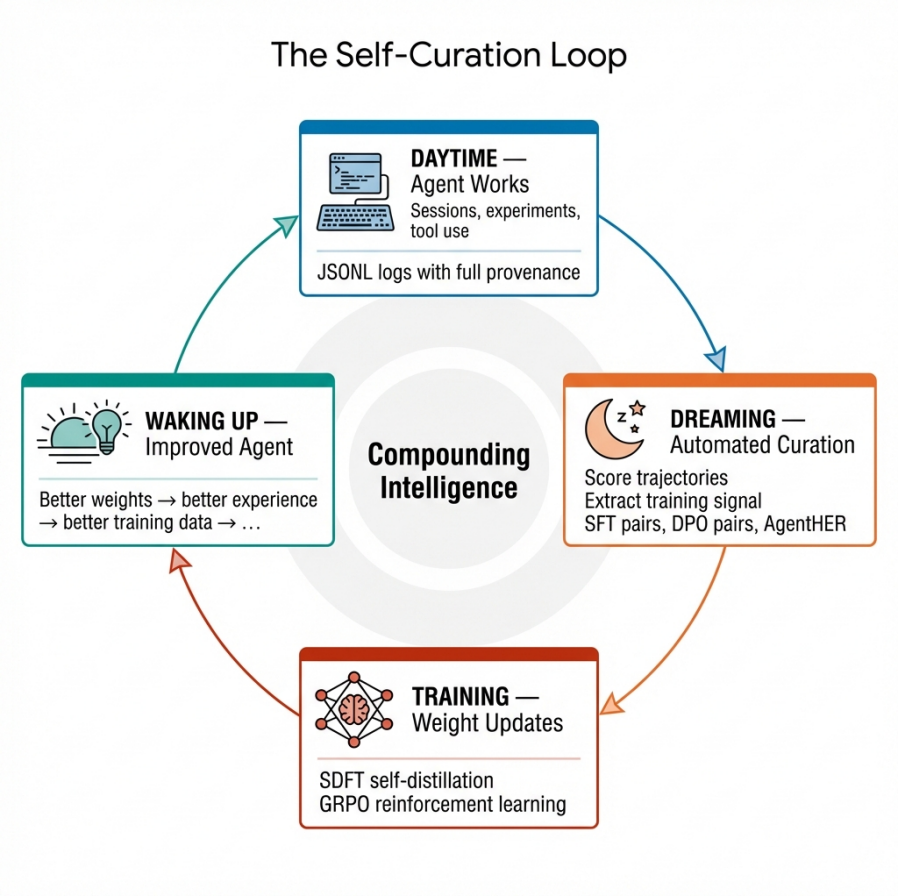

Figure 4: **The self-curation loop.** The agent works during the day, generating session logs. A dreaming automation curates training signal overnight. Weight updates produce a better agent, whose improved experience produces better training data—compounding intelligence.

3.4 Session Mining Pipeline

The dreaming phase runs a five-stage pipeline:

Discovery. Sessions from the last 24 hours are collected via the introspection API, filtered to completed (non-in-progress) sessions, and grouped by task type and outcome. Automation sessions are excluded to prevent self-referential training.

Scoring. Each trajectory is scored on multiple dimensions: task completion (binary), tool success rate (continuous), user satisfaction (did the user accept or request revisions), iteration count (fewer is better), error recovery (did the agent fix its own mistakes), and code correctness (did generated code pass tests). Reward signals range from immediate (tool success/failure, latency 0) through same-session (user acceptance, minutes) to next-morning (active learning questions, hours).

Extraction. Different trajectory types produce different training signal. Successful trajectories become SFT pairs (input: user request + context; output: agent's successful response including tool calls). Sessions where the agent eventually succeeded after initial failures yield DPO pairs (chosen: the successful approach; rejected: the failed approach). Outright failures get AgentHER treatment: the trajectory is truncated at the last successful sub-goal and relabeled as a correct demonstration for that sub-goal. AgentHER gains 7.1–11.7

points over success-only SFT on WebArena [Ding et al., 2026], confirming that failures are informative learning signal.

Quality filtering. Diversity filtering prevents over-representation of common tasks. Difficulty balancing mixes easy successes with hard-won victories. Deduplication collapses similar trajectories. Freshness weighting prioritizes recent experience. Ambiguous cases are flagged for human review. Recent work on memory-augmented agents confirms the value of structured experience curation: MIA [Qiao et al., 2026] establishes a bidirectional conversion loop between parametric and non-parametric memories, while MemSkill [Long et al., 2026] demonstrates that memory operations themselves can be treated as learnable, evolvable skills—agents improve not just what they remember but how they remember.

Dataset assembly. The output is a versioned, provenance-tracked dataset: `sft_pairs.jsonl`, `dpo_pairs.jsonl`, `hindsight_pairs.jsonl`, and `metadata.json` with coverage statistics and links back to source sessions.

3.5 AR^2: When the Experiment Loop Turns Inward

There is a structural recursion at the heart of Morpheus. The `autoresearch-science` framework runs an experiment loop: propose a change, execute the experiment, evaluate against a metric, keep or revert. Morpheus applies the same loop to the agent itself. The "dataset" is the agent's own session logs. The "model" being optimized is the agent. The "metric" is trajectory quality and held-out task performance. The "experiment" is a weight update. We call this **AR^2** (AutoResearch^2)—AutoResearch applied to AutoResearch.

This framing clarifies why the existing infrastructure maps so directly onto the self-improvement architecture: the experiment loop *is* the training loop, pointed at a different target. The monotonic improvement ratchet—only keep changes that improve the metric—transfers directly, providing a built-in safeguard against mode collapse.

The connection to ReST$^{\text{EM}}$ [Singh et al., 2024] is structural:

- ReST$^{\text{EM}}$: generate solutions → filter by correctness → train on filtered data → repeat.
- Morpheus: agent generates experience through scientific work → curate trajectories by quality → train on curated data → repeat.

Same principle, different substrate. ReST$^{\text{EM}}$ generates candidate solutions to math or code problems where verification is binary. Morpheus generates experience through multi-step tool use where verification is richer—tool success rates, test passage, user acceptance, experimental outcomes. The verifier is softer, but computational science domains (computational chemistry, protein structure prediction [Jumper et al., 2021], materials simulation [Merchant et al., 2023]) have tractable automated verification, making them natural starting points.

This softness is consequential. ReST$^{\text{EM}}$ demonstrated that filtering quality matters enormously—performance degrades substantially when the verifier admits low-quality solutions. Morpheus's curation criteria (Section 4.2) serve as the verifier, and their stringency directly determines training data quality. If the 80% tool success threshold admits mediocre trajectories whose *reasoning* was poor despite successful tool calls, the self-training loop will plateau or degrade. Tightening curation criteria against trajectory quality, not just outcome metrics, is an ongoing design challenge.

The key extension: ReST$^{\text{EM}}$ is a static loop (sample, filter, train). Morpheus is a living loop—the agent's real work generates the data, and the improved agent generates better work. The feedback is tighter because the training distribution matches the deployment distribution exactly: the agent trains on its own operational experience.

Apple's Simple Self-Distillation (SSD) [Zhang et al., 2026b] provides further evidence: sampling solutions from the model's own distribution and fine-tuning on unverified samples—without any RL, verifier, or reward model—improves Qwen3-30B code generation from 42.4% to 55.3% pass@1. The principle is the same: the model's own outputs, even unfiltered, contain enough signal for self-improvement.

3.6 Failure Modes and Mitigations

Self-training loops have well-documented failure modes that Morpheus must address.

Mode collapse. Training on the agent's own outputs can narrow behavioral diversity, producing an agent that repeats successful strategies without exploring alternatives. ReST^{EM} [Singh et al., 2024] mitigates this through binary verification on clean tasks; Morpheus operates in domains with softer rewards, where the risk is higher. **Mitigation:** Diversity filtering in the curation pipeline (Section 3.4) explicitly penalizes trajectory homogeneity. The replay buffer (Section 4.2) anchors the training distribution to older, more diverse experience. Temperature-scaled sampling during SDFT teacher generation preserves distributional breadth.

Reward hacking. The scoring rubric (tool success rate, iteration count, user acceptance) is gameable. An agent could learn to use fewer tools rather than better tools, or to produce superficially acceptable but shallow responses. **Mitigation:** Multi-dimensional scoring prevents single-metric gaming. The "code correctness" signal (tests pass or fail) provides a hard verification channel that cannot be gamed by surface strategies. Active learning questions to users provide a human-in-the-loop check on ambiguous cases. Over time, the reward model itself should be updated based on observed gaming patterns.

Distribution drift. As the agent improves, its trajectory distribution shifts. Training data from week 1 may become misaligned with the agent's behavior at week 4. **Mitigation:** Freshness weighting in curation prioritizes recent experience. The dual-rate architecture (Section 3.2) inherently manages this: fast weights adapt to the current distribution per session, while slow updates integrate data from the entire training window. EWC-style regularization [Kirkpatrick et al., 2017] constrains updates to preserve capabilities on older task distributions.

The Ouroboros warning. It is instructive that Ouroboros [Jaber and Jaber, 2026], despite achieving 43.4% training loss reduction through self-generated weight modulation, reports no improvement on held-out text. This is a direct cautionary example: optimizing training loss on self-generated data does not guarantee generalization. Morpheus addresses this through: (i) evaluation on held-out tasks at every training cycle (Section 7), (ii) multiple independent reward signals rather than a single training loss, and (iii) the SDFT consolidation step, which uses on-policy generation to avoid the distribution mismatch that may explain Ouroboros's held-out failure.

4 Preliminary Methods

This section describes the concrete protocols for each learning timescale.

4.1 Fast Timescale: In-Place Test-Time Training

The fast adaptation mechanism is In-Place TTT [Feng et al., 2026], which we select for three properties that make it uniquely suited to agent self-improvement:

Architectural compatibility. Fast weights live in the existing MLP down-projection matrices of standard Transformer blocks. No additional attention heads, side modules, or external memory is introduced. This means In-Place TTT can be applied to any off-the-shelf autoregressive LLM without architectural modification—the agent's serving infrastructure remains unchanged.

LM-aligned objective. Prior TTT approaches [Sun et al., 2020] used generic reconstruction objectives (e.g., masked autoencoding) that are misaligned with the model's pretraining objective. In-Place TTT aligns the fast-weight update with next-token prediction itself: the same objective the model was pretrained on. This alignment is critical—it means the fast weights encode information in a format the model already knows how to use. A 4B model

with In-Place TTT handles contexts up to 128K tokens, matching models with 2× more parameters.

Chunk-wise update. Long sequences are split into chunks, with fast-weight updates computed per chunk. This enables (i) efficient parallelization across chunks during training, (ii) bounded memory overhead regardless of sequence length, and (iii) natural integration with the agent's turn-based interaction pattern—each tool call and response can constitute a chunk boundary.

Protocol. During agent operation, the MLP down-projection matrices accumulate updates from the session's context via next-token prediction loss on previous turns. At session end, the fast weights are either discarded (default) or flagged for consolidation if session-level quality metrics exceed a threshold (see Section 4.2). We use separate learning rates for fast weights ($\eta_{\text{fast}} \approx 10^{-4}$) and keep all other parameters frozen.

4.2 Medium Timescale: Self-Distillation from Curated Sessions

The medium timescale uses SDFT [Shenfeld et al., 2026] to consolidate the day's experience into persistent weight updates.

Teacher-student protocol. The same model serves as both teacher and student. The teacher is conditioned on a successful agent trajectory via in-context learning (the trajectory is prepended as a demonstration). The student generates tokens without the demonstration. Training minimizes reverse KL divergence between teacher and student distributions on the student's own generated tokens:

$$\mathcal{L}_{\text{SDFT}} = \mathbb{E}_{x \sim p_\theta} \left[D_{\text{KL}}(p_\theta(\cdot \mid x) \,\|\, p_\theta(\cdot \mid x, \text{demo})) \right] \tag{1}$$

(reverse KL: the student's unconditional distribution is the reference). where p_θ is the model's distribution and demo is the curated trajectory. Because training is on-policy (the student generates its own tokens), there is no distribution shift between training and deployment.

Curation criteria. Not all sessions warrant consolidation. A session qualifies for SDFT if: (i) task completion was successful, (ii) tool success rate exceeds 80%, (iii) the trajectory is not a near-duplicate of an already-consolidated session, and (iv) the session required non-trivial reasoning (more than 3 tool calls). Sessions that fail these criteria but contain partial successes are routed to AgentHER processing [Ding et al., 2026] instead.

Anti-forgetting. SDFT's on-policy nature provides substantial forgetting resistance: because the model trains on its own distribution, updates are small perturbations rather than distribution shifts. Additionally, we maintain a replay buffer of 100 curated sessions from prior weeks, mixing 20% replay data into each nightly training run to anchor prior capabilities.

Validation requirement. A critical caveat: SDFT [Shenfeld et al., 2026] was validated on curated, expert-quality demonstrations. The original paper explicitly lists extending SDFT to "non-expert or noisy demonstrations" as future work. Agent session trajectories are structurally different from clean task demonstrations: they contain failed tool calls, error recovery, backtracking, verbose reasoning chains, and multi-turn dialogue. Even after curation (Section 3.4), they are noisier than SDFT's tested regime. Validating that SDFT's sample efficiency transfers to this data type is itself a research contribution. We propose a concrete ablation in Phase 2: compare SDFT performance on (a) raw session transcripts, (b) auto-cleaned transcripts with failed attempts removed, and (c) LLM-rewritten "clean" versions of the same sessions. This ablation determines how much trajectory preprocessing is needed before the medium timescale becomes viable.

4.3 Slow Timescale: Reinforcement Learning on Trajectories

The slow timescale uses GRPO [Shao et al., 2024] on multi-turn agent trajectories collected over weeks.

Environment interface. The agent's tool-execution harness is exposed as a Gymnasium-compatible `ToolEnvironment` via PyO3 bindings. The interface provides `reset(task)` → initial observation, `step(action)` → (observation, reward, done, info), and `close()`. Actions are text strings (the agent's responses including tool calls); observations are text strings (tool outputs). The training framework (verl or SkyRL) owns the LLM and generation; the harness owns tool execution and sandboxing.

Reward signal. For coding tasks: binary (tests pass or fail). For scientific experiment tasks: continuous (metric improvement over baseline, normalized to [0,1]). For open-ended tasks, we adopt the process reward approach from OpenClaw-RL [Wang et al., 2026]: next-state signals that arise naturally from every interaction (user replies, tool outputs, GUI changes, test verdicts) are converted into per-step process rewards via a PRM judge, providing denser feedback than episode-level outcome rewards alone. OpenClaw-RL's Hindsight-Guided On-Policy Distillation (OPD) further extracts directional hints from next states—producing token-level advantage supervision that is richer than scalar rewards. We plan to integrate both Binary RL (for evaluative signals) and OPD (for directive signals) into the slow timescale pipeline.

How next-state rewards work in practice. Every agent action produces a next state that encodes rich feedback. When the agent writes code and the tool returns a traceback, that traceback is both *evaluative* (the action failed) and *directive* (the error message says what was wrong). The PRM judge converts this into a scalar: $r = -1$ for the failed step. But OPD goes further: it constructs a hindsight-enhanced context by appending the traceback as a hint, generates a corrected action from the teacher (the same model conditioned on the hint), and computes token-level advantages between the original and corrected actions. This produces supervision that says not just "that was bad" but "here is specifically how each token should have been different." For a data analysis task where the agent runs a pandas query and the output table is empty, the next state (empty DataFrame) tells the PRM judge the step failed, while OPD uses the empty result as a hint to generate a corrected query—producing a training pair that teaches the specific fix, not just the failure signal. This mechanism applies uniformly across all interaction modalities: tool calls (output = next state), user conversations (reply = next state), and code execution (stdout/stderr = next state). The key insight from OpenClaw-RL [Wang et al., 2026] is that these signals are already present in every session log—they just need to be harvested.

Worked example: ADMET experiment loop. Consider one iteration of the ADMET investigation. The agent receives a prompt: "Improve the molecular property prediction model. Current best MA-RAE: 0.065." It generates a response: modify `experiment.py` to add MACCS fingerprint features and switch to a HistGBR ensemble. The `ToolEnvironment` executes the experiment (calls `step()` with the code edit), runs the evaluation pipeline, and returns the new MA-RAE: 0.062. The per-step reward is: $r = \max(0, \text{baseline} - \text{result})/\text{baseline} = (0.065 - 0.062)/0.065 = 0.046$. If the agent had made the metric worse (e.g., 0.070), $r = 0$. Over a multi-step trajectory (hypothesis → code edit → execution → analysis → next hypothesis), the return is the sum of per-step rewards, discounted by $\gamma = 0.99$. Failed tool calls (syntax errors, runtime crashes) receive $r = -0.1$ as a mild penalty encouraging clean code. This reward is dense (signal at every experiment step), bounded, and directly tied to the scientific metric—not gameable by superficial strategies.

Training protocol. Weekly training runs use GRPO with group size 8 (8 completions per prompt, relative ranking within the group). Training data: the week's curated sessions plus the replay buffer. Model: 7–14B parameter base with LoRA adapters (rank 64). Training budget: 4–8 GPU-hours per weekly run on a single node. Validation: held-out task suite evaluated before and after each training run.

4.4 Compute Budget

Table 4 estimates the compute requirements for the full Morpheus pipeline. All estimates assume a 7B base model with LoRA adapters (rank 64) and 20 sessions per day.

Table 4: **Estimated compute budget** for the Morpheus pipeline at steady state. All estimates assume a 7B model, LoRA rank 64, and 20 sessions/day. One-time costs are amortized over 12 weeks.

Component	Frequency	GPU-hours	Notes
In-Place TTT overhead	Per session	∼0.02	∼5% overhead per forward pass for MLP weight updates; negligible at session scale
SDFT nightly training	Daily	∼1–2	On-policy generation + reverse KL optimization on 15–20 curated sessions; single A100
GRPO batch training	Weekly	4–8	Group size 8, ∼100 curated prompts; single node with 4× A100
Session mining pipeline	Daily	∼0.5	LLM-based trajectory scoring and extraction; API cost ∼$2–5/day
Doc-to-LoRA (one-time)	Setup	8–16	Hypernetwork training on trajectory corpus; amortized: <1 GPU-hr/week
Total (steady state)	**Weekly**	**∼15–25**	**∼2–4 GPU-hours/day average**

At 2–4 GPU-hours per day on A100-class hardware, the full Morpheus pipeline is feasible on a single multi-GPU node. The dominant cost is the weekly GRPO run; the daily SDFT and session mining are lightweight by comparison. Scaling to 14B approximately doubles these estimates.

5 What Already Exists

Morpheus is not a greenfield project. The infrastructure for experience generation, session capture, and automated pipelines is operational. This section describes what exists and how each component feeds into the Morpheus architecture.

Note: All results in this section are from existing Level 1 and Level 2 systems with frozen model weights. They demonstrate the trajectory-generation and infrastructure capabilities that Morpheus builds upon, not the self-improvement architecture proposed in Sections 3–4.

5.1 The Experience Generator (Level 1)

The `autoresearch-science` package provides a domain-agnostic experiment framework with a Workspace API and typer CLI. It has been deployed on four problem domains:

- **ADMET:** 23.7% average improvement across 9 molecular property endpoints using OpenADMET challenge data [Frey et al., 2024], over 25+ autonomous iterations.
- **Psychiatric Genomics (PGC):** 72% loss reduction on polygenic risk score prediction in 18 minutes (standalone analysis using PGC GWAS summary statistics).
- **CIFAR-10 Speedrun:** 95.9% test accuracy (up from 90.0% baseline) under a 120-second training time budget on a single TITAN RTX GPU—a 59% error rate reduction through 21 autonomous iterations. The agent independently discovered SpeedNet with patch whitening, CutMix, label smoothing, EMA, and test-time augmentation. No pretrained weights were used.
- **Materials and protein fitness:** Preliminary runs demonstrating domain-agnostic applicability.

These experiment loops generate the trajectories that become Morpheus training data. Every iteration is a complete reasoning-to-outcome trace: what the agent hypothesized, what code it wrote, what results it observed, and how it decided what to try next.

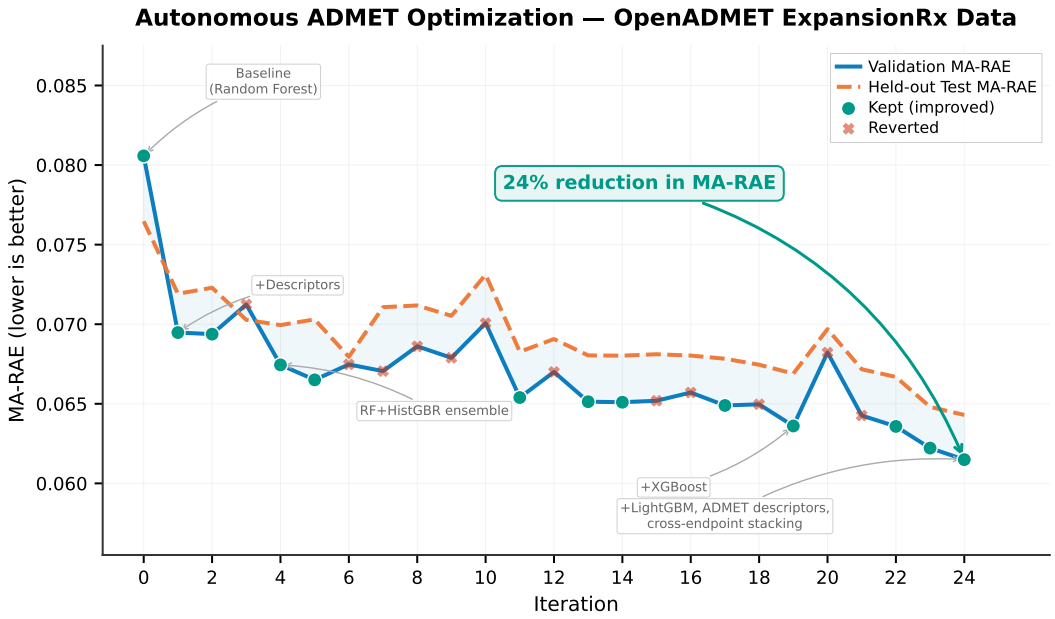

Figure 5: **ADMET improvement trajectory.** Cumulative best-so-far performance across 9 molecular property endpoints in the OpenADMET challenge. The `autoresearch-science` agent achieves 23.7% average improvement over the baseline through autonomous experimentation. Each iteration produces a complete trajectory suitable for training data extraction (Section 3.3). (Note: the y-axis label in the source figure reads right-to-left due to a rendering artifact; the metric is MA-RAE, lower is better.)

5.2 The Agent Infrastructure (Level 2)

The agent platform provides the session capture and automation infrastructure that Morpheus requires:

- **500+ operational sessions** with full provenance in JSONL format—every user message, assistant response, tool call, tool result, timing, and token usage recorded.
- **Session introspection API** for programmatic access to session histories, enabling the dreaming pipeline to read and analyze trajectories.
- **Automation engine** with 10 registered automations running on schedule, providing the execution substrate for nightly dreaming runs.
- **3 complete architectural rewrites** driven by the demands of scientific work, plus 33 PRs to the agent's own infrastructure.

5.3 The Research Base

The research groundwork is complete:

- 30+ papers collected and analyzed across RL training frameworks, agent memory systems, model editing, continual learning, and CLS theory.
- Comprehensive surveys of RL training infrastructure (verl, SkyRL, NeMo RL) and their integration points.
- Analysis of the agent harness as an RL environment: the agent loop is structurally isomorphic to what training frameworks need for multi-turn rollouts. The harness is the environment; training frameworks own the policy.
- A `ToolEnvironment` design providing Gymnasium-compatible `reset()/step()/close()` that exposes the agent's tool execution to Python-based training frameworks via PyO3 bindings.

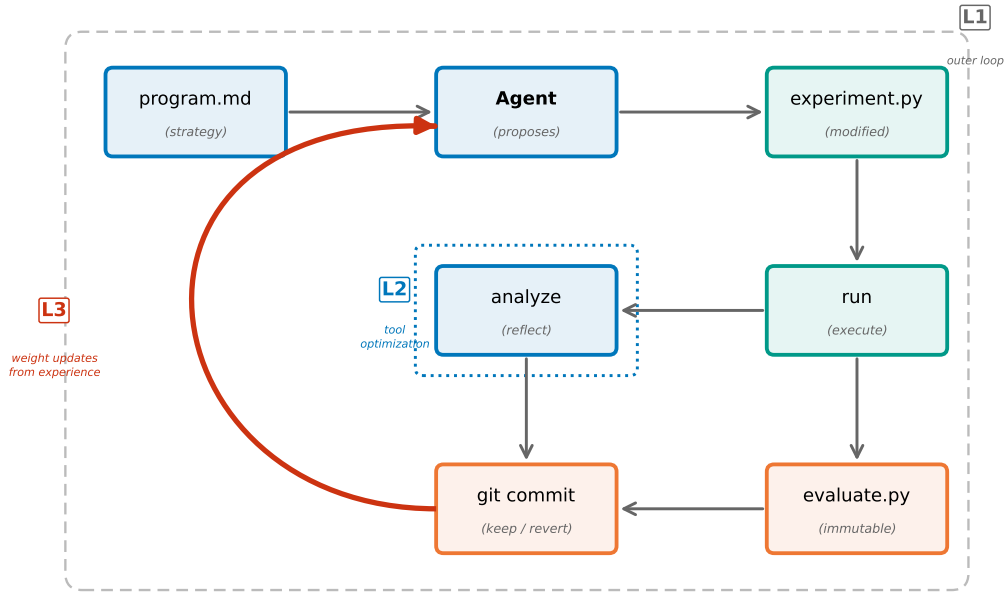

Figure 6: **System architecture** (cf. Figure 1). The `autoresearch-science` framework (Level 1) generates experimental trajectories. The agent platform (Level 2) captures sessions and runs automations. Morpheus (Level 3) adds the consolidation layer: session mining, data curation, and weight updates that compound capability over time.

- A benchmark pilot dataset designed for measuring when parametric knowledge overtakes retrieval across corpus sizes.

The research groundwork is complete. The implementation and validation phases follow.

6 Research Plan

The implementation proceeds in four phases, each producing a standalone deliverable. Later phases depend on earlier ones, but each phase has value independent of what follows.

6.1 Phase 1: Environment Bridge (2–3 weeks)

Goal: Connect the agent's tool execution to RL training frameworks.

Work:

- Implement `ToolEnvironment` as a PyO3 class exposing `reset()`/`step()`/`close()` with Gymnasium-compatible semantics.
- Wrap it for SkyRL's `BaseTextEnv` interface.
- Run single-turn GRPO training on coding tasks: the agent writes code, executes it via the tool environment, and receives binary reward (tests pass or fail).

Deliverable: A model trained on coding tasks via the tool environment that outperforms the base model on the same task distribution.

Design decision: The harness is the environment, not the policy. Training frameworks (verl, SkyRL) own the LLM and manage generation, token-level data, and gradient compu-

tation. The harness owns tool execution, sandboxing, and session state. This separation keeps both sides simple.

Risk: The verl/SkyRL integration requires matching specific tensor formats and rollout protocols. Estimated 50% schedule buffer. If the bridge proves harder than expected, a simpler direct-training approach (SFT on curated trajectories) provides a fallback.

6.2 Phase 2: Session Mining (2–3 weeks)

Goal: Extract training data from the agent's own operational sessions.

Work:

- Build the session mining pipeline: read JSONL histories via the introspection API, score trajectories, extract SFT/DPO/AgentHER pairs.
- Implement quality filtering: deduplication, diversity balancing, difficulty mixing, confidence thresholding.
- Deploy the dreaming automation: a nightly scheduled agent that runs the full curation pipeline and produces versioned datasets.

Deliverable: An automated pipeline producing training-ready datasets from session histories, running nightly without human intervention.

Design decision: Start with implicit reward signals only (tool success/failure, test passage, iteration count). Explicit user feedback integration comes later—it improves signal quality but adds complexity.

Risk: Trajectory scoring quality determines downstream training quality. The scoring rubric may need several iterations of refinement. This phase has the least technical risk but the most design iteration.

6.3 Phase 3: Fast Adaptation (4–6 weeks)

Goal: Enable per-session weight adaptation so the agent improves within sessions, not just between them.

Work:

- Implement a TTT context pipeline step in the agent loop: fast adapter weights update from recent messages via next-token prediction.
- Integrate Doc-to-LoRA for instant domain adaptation: given a new scientific domain (e.g., a set of papers), generate a LoRA adapter in one forward pass.
- Build the per-session adapter lifecycle: adapters are session-scoped by default, optionally promoted to persistent if quality metrics exceed a threshold.

Deliverable: An agent that shows measurable improvement within long sessions—fewer tool errors, faster task completion, better error recovery as the session progresses.

Risk: Integrating TTT into a live agent serving pipeline while maintaining latency SLAs is a systems engineering challenge. If latency overhead exceeds 20%, a Doc-to-LoRA-only approach (pre-computed adapters rather than live weight updates) provides an alternative.

6.4 Phase 4: Full CLS Loop (8–12 weeks)

Goal: Close the loop: dual-rate training with sleep consolidation and stability-plasticity control.

Work:

- Dual-rate training: fast per-session adapters (Phase 3) combined with slow batch RL (Phase 1) and nightly SDFT consolidation (Phase 2).
- Sleep consolidation: nightly distillation of session-specific adapters into the base model via SDFT, preserving existing capabilities.
- Stability-plasticity control: EWC-style regularization [Kirkpatrick et al., 2017] or replay buffers to prevent catastrophic forgetting across training cycles.
- Evaluation: measure intelligence compounding—is the agent at week 4 better on held-out tasks than the agent at week 1?

Deliverable: An agent demonstrably smarter at week 4 than week 1, trained entirely from its own operational experience without human-curated data.

Risk: This phase requires 4+ weeks of continuous operation to measure compounding. Infrastructure stability, model serving reliability, and evaluation consistency all become factors. A more realistic timeline is 12–16 weeks with debugging. The SDFT-with-LoRA validation (flagged in Section 8) should be completed early in this phase.

6.5 Success Metrics

Three tiers, each a meaningful result:

1. **Minimum viable result:** An agent trained via RL on its own tool-use trajectories outperforms the base model on the same task distribution. Measured by task success rate, tool efficiency (fewer steps), and error recovery rate.
2. **Compelling result:** An agent whose performance improves measurably over a week of operation through automated session mining and periodic training—no human-curated data involved.
3. **Home run:** Compounding intelligence. Early sessions generate training signal that improves later sessions, which generate better training signal. The agent is genuinely smarter at week 4 than week 1 on tasks it has never seen.

7 Evaluation Framework

Measuring self-improvement requires evaluation at multiple timescales, matching the architecture's three learning rates. We propose a combination of existing benchmarks, adapted benchmarks, and a new agent-specific continual learning suite.

7.1 Per-Session Metrics (Fast Timescale)

These metrics assess whether In-Place TTT improves the agent *within* a single session:

- **Tool success rate trajectory:** Does the fraction of successful tool calls increase as the session progresses? Measured as a slope over turn number.
- **Error recovery rate:** When a tool call fails, how often does the agent recover on the next attempt? Compared with and without fast weights.
- **Task completion speed:** Number of turns to complete a task, compared between early and late sessions with the same task type.
- **Perplexity on session continuation:** Does the model's perplexity on later turns decrease as fast weights accumulate context? Direct measurement of information internalization.

Existing benchmarks. Long-context benchmarks (RULER 128K, Needle-in-a-Haystack) can measure In-Place TTT's context compression. We adapt these by embedding the information within agent-style tool-call transcripts rather than raw text.

7.2 Daily/Weekly Metrics (Medium Timescale)

These metrics assess whether nightly SDFT consolidation produces a better agent the next day:

- **Held-out task performance:** A fixed suite of 50 tasks spanning code generation, data analysis, literature search, and scientific experiment design. Evaluated before and after each nightly training run.
- **Trajectory quality score:** Average quality of the day's trajectories (by the scoring rubric in Section 3.4), plotted over days. An improving trend indicates the agent generates better experience as it improves.
- **Session mining yield:** Fraction of daily sessions that produce usable training data (SFT, DPO, or AgentHER pairs). Higher yield means the agent fails less and produces more learning signal.

Existing benchmarks. SWE-Bench Lite (software engineering), GAIA (general AI assistants), and HumanEval+ (code generation) provide standardized capability measurements. We run these weekly to track capability trends.

7.3 Long-Term Metrics (Slow Timescale)

These metrics assess whether intelligence compounds over weeks:

- **Week-over-week capability:** Performance on the held-out suite at week 1, 2, 3, 4. The key question: is the trajectory monotonically improving?
- **Transfer:** Does improvement on one domain (e.g., coding) transfer to another domain (e.g., data analysis)? Measured by cross-domain held-out tasks.
- **Forgetting:** Does improvement on new tasks degrade performance on old tasks? Measured by a retained-capability suite of tasks the agent could solve at week 1.
- **Novel task generalization:** At week 4, present tasks from distributions not seen during training. If the agent has genuinely compounded intelligence (not just memorized trajectories), it should generalize.

7.4 The Agent Continual Learning Benchmark

No existing benchmark measures continual learning for *agents* (as opposed to NLP classification). We propose constructing one:

Task domains and exemplars. Five domains, each with 10 tasks (50 total):

1. **Code generation:** Write a Python function that passes unit tests (e.g., implement topological sort; parse nested JSON with error handling; build a rate limiter with sliding window).
2. **Data analysis:** Answer a question from a dataset using pandas or SQL (e.g., "Which product category had the highest growth rate in Q3?"; "Find outliers in the sensor data using IQR method").
3. **Literature search:** Find and synthesize information from papers (e.g., "What methods achieve >90% on HumanEval without fine-tuning?"; "Compare GRPO vs. PPO sample efficiency across 3 recent papers").
4. **Scientific experimentation:** Run an AutoResearch-style loop (e.g., "Improve the random forest baseline on this tabular dataset"; "Find hyperparameters that reduce validation loss below 0.05").
5. **System administration:** File manipulation and configuration (e.g., "Set up a cron job that backs up this directory daily"; "Debug why this Docker container fails to start").

Tasks are scored by: (i) binary completion (did the output satisfy the requirement?), (ii) efficiency (tool calls used), and (iii) quality (LLM-judge rubric, calibrated against 3 human-rated examples per task).

Protocol. The agent encounters tasks sequentially, one domain at a time (1 week per domain). After each domain, it is evaluated on: (i) the current domain (plasticity), (ii) all prior domains (stability), and (iii) a held-out cross-domain suite (transfer). This directly measures the stability-plasticity-transfer tradeoff that CLS theory predicts dual-rate learning should optimize.

Baselines. (1) Frozen model with growing context (Level 2 ceiling). (2) Naive fine-tuning on each domain (catastrophic forgetting baseline). (3) Replay-only fine-tuning (no SDFT, no TTT). (4) Full Morpheus (TTT + SDFT + GRPO). The comparison isolates the contribution of each timescale.

8 Open Questions

Several questions remain genuinely unresolved. I list them not as obstacles but as the research problems that make Morpheus interesting.

SDFT scale threshold. SDFT [Shenfeld et al., 2026] works at 7B parameters (+4 points) and 14B (+7 points) but showed minimal gains at 3B. Morpheus will likely use LoRA adapters rather than full model updates. Does the "in-context teacher" assumption hold when updates are confined to low-rank adapter matrices? The answer determines whether the medium timescale works at practical model sizes. This question is sufficiently important that Phase 2 should include an explicit SDFT-with-LoRA validation step before committing to the full CLS loop in Phase 4.

Trajectory-to-adapter transfer. Doc-to-LoRA [Charakorn et al., 2026] was trained on documents. Agent trajectories—interleaved reasoning, tool calls, and observations—are structurally different from static text. Can the hypernetwork be retrained on trajectory data, or does it need architectural changes? If the hypernetwork treats input as unstructured tokens, the transfer may be straightforward. If it relies on document structure, adaptation work is needed.

Data volume. A back-of-envelope calculation: 20 sessions per day $\times$ 7 days = 140 sessions per week. With the curation criteria in Section 4.2 (task completion, 80% tool success, non-trivial), approximately 40–50% qualify, yielding $\sim$60–70 curated sessions per week. Each session produces $\sim$1–3 training pairs (SFT, DPO, or AgentHER), giving $\sim$100–200 training examples per week for GRPO. Is this sufficient? SDFT's sample efficiency (meaningful improvement from single demonstrations at 14B) suggests the nightly consolidation is viable even with 10–15 sessions. For weekly GRPO with group size 8, $\sim$100 prompts provides $\sim$12 gradient updates—thin but potentially sufficient for LoRA-only training with rank 64. If empirical results show data starvation, multi-agent experience sharing [Qu et al., 2026] or synthetic trajectory augmentation become necessary, not optional.

Forgetting evaluation. Existing continual learning benchmarks focus on NLP classification tasks, not agent capabilities. If an agent learns better Python debugging, how do I measure whether it has forgotten how to write SQL? No agent-specific continual learning benchmark exists. Designing one—with task diversity that captures real operational breadth—is a prerequisite for claiming stability-plasticity control works.

Consolidation policy. When should the system consolidate (weight update) versus retrieve (context injection)? Not all experience warrants weight changes. Meta-learning the consolidation boundary—when to dream versus when to remember—is itself a learning problem, applicable at the system architecture level.

9 Conclusion

The path from agents that do science to agents that get smarter at science is the defining challenge. Level 1 is commodity. Level 2 is the current frontier. Level 3—systems that compile their own experience into weight updates—remains open.

Morpheus is a specific proposal for how to get there: three learning timescales inspired by CLS theory, a self-curation loop that turns agent experience into training data, and a phased implementation plan with clear deliverables at each stage. It combines the three research camps that have operated in isolation—memory systems, weight updates, and instant adapter generation—into a single architecture where the agent's work generates the signal that makes it better at future work.

My background in knowledge representation [Meng et al., 2022, 2023]—locating and editing factual knowledge in neural network weights—applies directly. The question those projects answered was: where does knowledge live in a network, and how do you change it? The question Morpheus answers is: can an agent generate the knowledge that should be changed, and the training signal to change it, from its own operational experience?

The infrastructure exists: a working experiment framework, 500+ sessions with full provenance, automation pipelines, and a research base of 30+ analyzed papers. The technical ingredients—SDFT for anti-forgetting consolidation, AgentHER for extracting signal from failures, TTT for per-session adaptation, ReST$^{\text{EM}}$ for the generate-filter-train loop—are proven individually.

The remaining work is integration and validation—turning AR into AR^2. The minimum experiment that would confirm the core thesis: train an agent via GRPO on its own curated session trajectories and measure held-out task improvement over 4 weeks. If held-out performance improves monotonically—without catastrophic forgetting and without human-curated data—that would be the first demonstration of compounding intelligence in an AI agent. If it works, the implications extend beyond individual agents: a population of Morpheus agents, each generating experience in different scientific domains, sharing curated trajectories via CORAL-style protocols [Qu et al., 2026], could produce a collective intelligence that compounds across both time and specialization.

References

Marc-Antoine Allard et al. Experiential reflective learning for self-improving LLM agents. *arXiv preprint arXiv:2603.24639*, 2026.

@carnot_cyclist. Five desiderata for continual learning in LLMs. Online post, PrimeIntellect, 2026. URL https://x.com/carnot_cyclist.

Rachit Charakorn et al. Doc-to-LoRA: Learning to instantly internalize documents into LoRA adapters. *arXiv preprint arXiv:2602.15902*, 2026. Sakana AI.

Liang Ding et al. AgentHER: Hindsight experience replay for LLM agent trajectory relabeling. *arXiv preprint arXiv:2603.21357*, 2026.

Yu Feng et al. In-place test-time training via repurposed MLP projection matrices. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026. Oral presentation.

Nathan C Frey et al. OpenADMET: An open-source toolkit and benchmark for ADMET property prediction. *arXiv preprint*, 2024.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, et al. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

Kevin Gu. AutoAgent: Autonomous agent harness optimization. ThirdLayer, 2026. URL https://thirdlayer.ai/autoagent.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2022.

Jaber Jaber and Osama Jaber. Ouroboros: Dynamic weight generation for recursive transformers via input-conditioned LoRA modulation. *arXiv preprint arXiv:2604.02051*, 2026.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583–589, 2021.

Andrej Karpathy. AutoResearch: Automated machine learning research. GitHub repository, March 2026. URL https://github.com/karpathy/autoresearch.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526, 2017.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. MEMENTO: Teaching LLMs to manage their own context. 2026. URL https://github.com/microsoft/memento. Microsoft Research.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? Complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534, 2016.

Harrison Lee et al. Meta-harness: End-to-end optimization of LLM harness code via full execution trace access. *arXiv preprint arXiv:2603.28052*, 2026. Stanford/KRAFTON/MIT.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Yibo Li et al. Just-in-time reinforcement learning: Continual learning in LLM agents without gradient updates. *arXiv preprint arXiv:2601.18510*, 2026a.

Yuxing Li et al. ARISE: Agent reasoning with intrinsic skill evolution in hierarchical reinforcement learning. *arXiv preprint arXiv:2603.16060*, 2026b. GWU/UT Dallas.

Yewei Liu et al. SHINE: A scalable in-context hypernetwork for mapping context to LoRA in a single pass. *arXiv preprint arXiv:2602.06358*, 2026. PKU/Technion.

Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026. ICML 2026.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102 (3):419–457, 1995.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Amil Merchant, Simon Batzner, Samuel S Schoenholz, Muratahan Aykol, Gowoon Cheon, and Ekin Dogus Cubuk. Scaling deep learning for materials discovery. *Nature*, 624:80–85, 2023.

Jingyang Qiao, Weicheng Meng, Yu Cheng, et al. Memory intelligence agent. *arXiv preprint arXiv:2604.04503*, 2026.

Xinle Qu, Zilong Zheng, Yiming Zhou, et al. CORAL: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026. MIT/NUS/Stanford/Meta.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. DeepSeek AI — introduces GRPO.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026. MIT/ETH Zurich.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Aviral Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabela Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kiam Lee, Mahesh Joshi, Manaal Sethi, Raphael Gontijo-Lopes, Renee Tung, Rosanne Kim, David Abel, Harsh Agrawal, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research (TMLR)*, 2024.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Arnuv Tandon, Karan Dalal, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025. Astera/NVIDIA/Stanford.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026. URL https://github.com/Gen-Verse/OpenClaw-RL.

Chengsong Xu et al. ASI-Evolve: AI accelerates AI — a closed-loop approach to scientific discovery. *arXiv preprint arXiv:2603.29640*, 2026. GAIR-NLP.

Yutaro Yamada, Cong Lu, Shengran Hu, Jakob Foerster, and Jeff Clune. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026a. Meta FAIR.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly simple self-distillation improves code generation. *arXiv preprint arXiv:2604.01193*, 2026b. Apple.