ChatGPT

Online and Continual Reinforcement Learning (2022–2025): Methods and Systems

Definitions and Distinctions

Online vs. Continual RL: *Online* reinforcement learning usually refers to agents that learn *during* interaction with the environment, updating their policy on-the-fly instead of being trained entirely offline on a fixed dataset. In contrast, *continual* (or lifelong) RL emphasizes an agent's ability to **"never stop learning"** – it treats learning as *endless adaptation* rather than a one-and-done training phase [1] [2]. Traditional deep RL often assumes a fixed task where training eventually ceases once an optimal policy is found, at which point the agent is deployed with frozen weights [3] [4]. Continual RL, however, considers non-stationary environments or sequential tasks, requiring the agent to keep updating its policy *during inference or deployment*. In other words, the agent **continues learning after deployment**, adjusting to new tasks or changes in the environment without retraining from scratch.

Key Concepts: In continual RL, each new task or condition arrives sequentially (often without clear task boundaries), and the agent must **adapt to new challenges while retaining past knowledge** [5] [6]. This raises classic issues of **stability vs. plasticity**: the agent should remain stable on previously learned behaviors (avoid *catastrophic forgetting*) while being plastic enough to acquire new skills [7]. Catastrophic forgetting is when learning new tasks causes the model to overwrite old knowledge, whereas **negative transfer** refers to the interference where old knowledge actually *impedes* learning of new tasks (or vice versa) [8]. Another phenomenon is **loss of plasticity** – deep neural networks can become gradually *unable to learn* new information after prolonged training [9]. Recent findings show that standard deep learning, if used continuously, eventually learns no better than a shallow network unless special measures (like injecting randomness) are taken to maintain plasticity [9] [10]. Continual RL research addresses all these aspects, unifying what was sometimes distinguished as *"lifelong" RL (emphasizing fast adaptation)* vs *"continual" RL (emphasizing knowledge retention)* [11]. In practice, the terms overlap, and both involve **policy updates during deployment** to handle non-stationary conditions.

On-Policy vs. Off-Policy Updates: In an online continual setting, updates can be on-policy (using the agent's current experience stream) or off-policy (reusing past experiences). *Online RL* in the strict sense usually implies on-policy incremental updates every step/episode, whereas *offline RL* would collect data and train later without continuing to explore. Our focus is on settings where the **deployed agent keeps learning from new data**. For example, a recommendation agent may continuously update its strategy as user preferences drift, or a robot adapts its control policy as its actuators age – these are instances of online continual RL. By contrast, standard deep RL benchmarks (like training Atari agents) typically involve a lengthy training phase after which learning is stopped; if such an Atari agent were to continue updating weights while being played indefinitely, that would shift into the continual RL paradigm.

Key Approaches for Policy Adaptation (2022–2025)

Researchers have developed a variety of approaches to enable safe and effective policy adaptation during inference. Recent methods can be broadly categorized by how they update the policy and mitigate forgetting:

- **Full Policy Fine-Tuning:** The most direct approach is to continue training the entire network on new data (or new tasks) as it arrives. This maximizes *plasticity* (all parameters can adapt) but easily causes catastrophic forgetting of earlier tasks if no precautions are taken. Fine-tuning a deep RL agent on a sequence of tasks often serves as a baseline, which tends to achieve good performance on the *latest* task at the cost of severe degradation on *earlier* tasks [12]. For instance, in the Continual World benchmark (10 robotic manipulation tasks), naive sequential fine-tuning quickly forgets how to solve the early tasks [12]. Full-policy updates are thus usually coupled with other techniques (replay, regularization, etc.) to preserve knowledge.

- **Partial/Adaptive Parameter Updates:** To balance plasticity and stability, many approaches update only *parts* of the policy or add specialized adaptation modules. Freezing a subset of network weights that encode old skills can protect them from being overwritten, while new tasks are learned via a small set of *adaptive parameters*. Techniques include **Low-Rank Adaptation (LoRA)** layers or side networks that adjust the policy without altering the core weights, as well as **prompting/gating methods** in which a meta-network decides which neurons or modules to activate for a given task. For example, recent work on "continual task allocation via sparse prompting" learns an overcomplete dictionary of masks that *select a subnetwork* for each task from a meta-policy network [13]. By only changing the task-specific mask (prompt) and leaving the meta-network intact, the agent avoids interference between tasks. Similarly, *hypernetworks* or policy conditioning on a latent task code can achieve partial updates: the base policy remains mostly fixed, and adaptation happens by altering the task code. These approaches aim to **localize changes** for new tasks, minimizing negative side-effects on old behaviors.

- **Meta-Learning and Fast Adaptation:** *Meta-RL* methods train the agent to be inherently adaptable, effectively learning how to learn. One prominent idea is to find an initialization (or representation) that can be quickly fine-tuned to new tasks with only a few gradient steps – analogous to **MAML (Model-Agnostic Meta-Learning)** but in an RL context. For example, *Continual Meta-Policy Search (CoMPS)* combines meta-learning with continual learning by periodically reusing past tasks for meta-training, aiming to retain an initialization that serves all seen tasks [14]. Another meta-RL approach is to use *recurrent policies* that internalize the adaptation process (sometimes called **RL^2**): the agent's RNN hidden state can carry information about the new task, effectively learning a learning algorithm within the network. The benefit of meta-RL is **rapid forward transfer** – when a new task arrives, the agent adapts far more sample-efficiently than a from-scratch learner. However, meta-RL typically requires a diverse training phase (many tasks or environment variations) and does not by itself prevent forgetting; it often needs to be combined with other continual learning techniques if tasks arrive sequentially beyond the meta-training distribution.

- **Experience Replay and Distillation:** Borrowing from continual learning in supervised domains, **experience replay** is a core tool to combat forgetting in RL. The idea is to maintain a buffer of past experiences (state, action, reward sequences from old tasks) and intermix them with new experiences during training. This rehearsal stabilizes updates by reminding the agent of what it

learned before. In practice, replay can be combined with **policy distillation** or regularization: for instance, a *"Replay-enhanced" SAC agent (2023)* normalizes Q-values across tasks and adds a Kullback–Leibler (KL) regularization term to distill the current policy towards the old policy on previous tasks [15]. This method, tested on the Continual World robotic benchmark, showed significantly less forgetting than naive fine-tuning, outperforming baselines that simply replay data without such regularization [15]. Another example is **Continual Model-Based RL** which uses a world model to replay experiences: an ICML 2025 paper proposes training a VAE-based world model for each task and *freezing it* when moving to the next; new tasks are encouraged to explore beyond the "known" state-action regions of the old model, while the frozen model is used to generate pseudo-experiences from past tasks for rehearsal [16]. In essence, replay and distillation provide a *memory* for the agent, either through stored samples or through constraints that keep the new policy close to the old on old-task data.

- **Regularization and Optimizer Tweaks:** A complementary strategy is to impose penalties on changes to important weights (as in **Elastic Weight Consolidation (EWC)** and similar techniques) or to modify the training dynamics to maintain plasticity. For example, *online EWC* (adapted to RL) adds a quadratic penalty to weight updates proportional to their importance for earlier tasks, thereby reducing forgetting of those tasks [17]. However, by itself EWC can bias learning and even fail to ensure safety in changing environments (one study showed that online EWC could not satisfy safety constraints in a robotic control scenario with non-stationary faults) [17]. Newer regularization approaches look at maintaining network *capacity* for change: a NeurIPS 2024 paper introduced **Parseval regularization** for continual RL, which enforces an orthonormal constraint on weight matrices to prevent them from becoming too specialized [18]. By keeping weights orthogonal, the network preserves headroom in gradient directions, mitigating the loss of plasticity during optimization [18]. Another notable idea is **continual backpropagation** [10]: randomly reinitializing a small fraction of neurons during training to inject diversity. Sutton *et al.* (Nature 2024) found that this technique can *indefinitely* sustain learning ability in a deep network, whereas a standard SGD-trained network would eventually stagnate on new data [19]. Moreover, researchers have crafted new optimizers for lifelong RL – e.g. *Fast TRAC* (NeurIPS 2024) is a "parameter-free" optimizer that dynamically adjusts update steps to prevent the model from losing plasticity over time [20]. Such optimizer-level interventions are a rising theme to tackle the delicate balance between stability and plasticity in continual learning.

- **Modular and Expandable Architectures:** A powerful way to avoid forgetting is to grow the model's capacity when encountering new tasks. **Modular architectures** assign different modules (neurons or sub-networks) to different tasks, often with some gating or attention mechanism to route inputs to the appropriate module. A classic example is *Progressive Networks* (from 2016, prior to our 2022–2025 range) which added new columns for new tasks and connected them to frozen old columns. Recent works build on similar ideas: *Self-Composing Policies* (ICML 2024) uses a "growing neural network" that adds new policy components for each task and an attention-based composition mechanism to integrate outputs of previous policies with the new one [21]. By doing so, the agent essentially accumulates a library of skill modules. In experiments on Meta-World manipulation tasks and Atari games, this method achieved **strong forward transfer and minimal forgetting**, nearly matching single-task learning performance on new tasks while retaining high average performance on old tasks [22]. Another approach, *Building a Subspace of Policies* (ICLR 2023), incrementally expands a low-dimensional subspace that can represent all policies encountered [23]. The agent starts with a small set of basis policies and adds more as tasks arrive, optimizing within this

subspace. This yielded very competitive results (often outperforming fixed-size networks), though at the cost of growing computational requirements over time [24]. In general, expandable architectures trade off memory/compute for performance – they sidestep interference by design, at the expense of model size and sometimes the need to know task boundaries (to decide when to branch out a new module).

- **World Models and Planning:** A notable line of work integrates continual learning with **model-based RL** and planning. By learning a *dynamics model* that is updated continually, the agent can perform lookahead and policy search in the learned model, which can reduce direct interference on the policy. One ICML 2025 paper, for instance, introduced *online world-model planning* for CRL [25]. The idea is to maintain a unified world model that is updated with each new task's data (taking care to avoid forgetting dynamics of earlier tasks), and use model-predictive control or tree search in this learned model to make decisions. They also proposed a new *Continual-Planning Benchmark* ("Continual Bench") to evaluate such methods [25]. The advantage is that knowledge is stored in the world model, which can be more parameter-efficient and easier to regularize (since it predicts observable quantities), and planning can adapt behavior without always needing direct policy parameter updates. However, keeping a world model accurate across many tasks is itself challenging; the cited work uses techniques like model rehearsal (generative replay of imagined past transitions) to mitigate model drift [25]. Model-based approaches in continual RL are still emerging, but they show promise in handling non-stationary environments by separating the learning of dynamics from the learning of policy.

Table 1 summarizes several representative methods from 2022–2025, illustrating the diversity of strategies for online and continual RL:

Method (Year)	Adaptation Strategy	Domain	Key Results
Self-Composing Policies (2024) [21]	Expandable network with attention composition (adds modules per task)	Meta-World (robotics), Atari games	Matched single-task learning curves on new tasks; old tasks' performance drop was minimal (high average retention) [22]. Outperformed static-network baselines in both forward transfer and avoiding forgetting.
Reset & Distill (R&D) (2025) [26]	Dual-agent approach: reset a fresh actor for new task, then distill knowledge back into main policy	Meta-World (long task sequences)	Solved new tasks without negative transfer by isolating learning; after distillation, maintained higher success rates across 10+ sequential tasks than prior SOTA methods (significantly higher overall success percentage) [27].

Method (Year)	Adaptation Strategy	Domain	Key Results
Replay-enhanced SAC (2023) [15]	Experience replay + KL distillation regularization in Soft Actor-Critic	Continual World (10 robotic tasks)	Improved stability and reward on early tasks compared to naive fine-tuning. Achieved better average performance (over all tasks) than pure replay or EWC baselines; e.g. final average success was substantially higher (>10% absolute) than baseline replay [15].
Parseval CL (2024) [18]	Orthogonal weight regularization (Parseval layers)	Atari, Procgen, Gym (continual variants)	Mitigated plasticity loss: networks maintained the ability to learn new games longer. On a 5-task Atari sequence, for instance, achieved higher final scores on task 5 *and* retained more performance on task 1 compared to standard training (approx. 15–20% less forgetting) [18]. Improved optimization stability enabled learning where baseline SGD would stagnate.
Safe CDA (2025) [28] (Safe Continual Domain Adaptation)	Safe RL constraints + online adaptation after sim-to-real transfer	Robotics (real robot arm with changing dynamics)	Allowed policy to adapt to real-world changes (e.g. payload shifts) **without** policy divergence or safety violations. The deployed agent adjusted to new domain parameters, regaining performance that a fixed policy lost, all while avoiding catastrophic forgetting of its base skills [28]. Demonstrated stable adaptation in 100+ trials with zero safety constraint breaches (whereas an unconstrained RL update led to failures).
SEAL (Self-Adapting LLM) (2025) [29] [30]	LLM self-finetuning via inner-loop supervised learning on model-generated data + outer-loop RL (dual loop)	Language (LLMs on QA and few-shot tasks)	Achieved **+13.5%** absolute accuracy improvement on a QA task (33.5→47.0%) after two self-generated fine-tuning cycles [29]. On few-shot reasoning tasks, improved success from 20% to 72.5% with RL-guided self-edits [31]. Showed that reinforcement learning-based self-adaptation mitigates forgetting of prior knowledge better than naive fine-tuning [32]. Computational overhead noted (~30–45s per update step) [33].

Table 1: Selected recent methods for online/continual RL, illustrating different adaptation strategies, domains, and outcomes. (SOTA = state of the art; QA = question answering; LLM = large language model.)

Notable Systems and Implementations (2022–2025)

Continual and online RL research has been applied across a broad range of domains, from simulated games to real-world robotics and online services. Below we highlight notable systems and their mechanisms, as well as the domains/benchmarks where they showcase continual learning:

Robotics and Control Systems

In robotics, continual RL enables **adaptive controllers** that can cope with changes in hardware or environment over time. A prime example is *sim-to-real adaptation*: policies trained in simulation are deployed on real robots and then refined online as the real-world dynamics differ from simulation. A 2025 system dubbed **Safe CDA** (Safe Continual Domain Adaptation) specifically tackled this scenario [34] [28]. The authors noted that typical domain randomization (training on many simulated variations) produces a robust but *fixed* policy, and due to RL's instability and safety concerns, practitioners often freeze the policy after deployment [34]. Safe CDA instead keeps learning on the real robot *with safety constraints*. It integrates a **safe RL algorithm** (constrained policy optimization) to ensure the robot respects safety limits (e.g. joint torque or velocity constraints) while an online learner fine-tunes the policy to current conditions [34]. In experiments with a robotic arm, this method allowed the policy to track changing dynamics (like wear-and-tear or load changes) and improved performance in the new domain **without catastrophic failures**. Importantly, it avoided forgetting the general skills learned in sim – the adapted policy could still perform the original range of behaviors, just calibrated to the new domain [28].

Another real-world consideration is **fault tolerance and non-stationary dynamics**. For instance, a 2025 study on UAV (drone) control examined how a continual RL agent handles sudden actuator failures or changing flight conditions [35]. They found that a standard continual learning method (online EWC, in this case) failed to guarantee safety when a joint failure occurred, and a standard safe RL method (which was not designed for sequential task learning) suffered severe forgetting of how to handle prior conditions [17]. To address this, they proposed a *reward shaping* technique that explicitly encourages the agent to remember safety-critical behavior while learning anew [36]. This kind of system showcases a best practice in safety-critical domains: **blend constraints with continual learning**, so that adaptation does not come at the expense of violating safety requirements. The result was a drone controller that could adapt to a failed rotor scenario and later return to normal flight, all while respecting pre-defined safety constraints (like keeping within stable flight envelopes) [35].

In robotic **manipulation and multi-skill learning**, the *Meta-World* and *Continual World* benchmarks have spurred implementations that demonstrate continual learning at scale. Meta-World provides a suite of discrete tasks (like opening a drawer, pressing a button, etc.), and Continual World strings some of these tasks into long sequences to test lifelong learning [37] [38]. DeepMind and other researchers have built agents that can learn 10–20 manipulation tasks in a row. Notably, *Self-Composing Policies (CompoNet)*, mentioned earlier, was deployed on Meta-World and achieved **near state-of-the-art success rates on all tasks even after sequential training**, whereas a baseline single network fine-tuned through those tasks drops sharply on early tasks. For example, in one sequence of 10 tasks, a fine-tuning agent's success on the first task might fall to <40% by the end, whereas the self-composing modular agent retained ~90% success on that first task while also solving the later tasks at a high level [22]. Such systems often rely on an *expanding architecture* – in CompoNet's case, growing the network with each new task – which is feasible in robotic settings where the number of tasks is known or limited (10 tasks in Continual World). The trade-off is memory: the model grows with tasks, but the benefit is **zero forgetting** in principle. These

implementations underscore a theme: *modularity and expansion are powerful for continual robotics*, where reusing skills (e.g. grasping) in later tasks is natural if the architecture supports it.

Games and Simulated Environments

Games have long been a testbed for RL, and they offer controlled settings for continual learning research. Recent continual RL systems in games address both **multi-task learning** (e.g. learning multiple different games) and **non-stationary environment** challenges (e.g. game rules or opponents that change over time).

A notable benchmark introduced in 2023 is **COOM (Continual Doom)**, which provides a suite of 8 custom scenarios in the ViZDoom 3D environment designed for continual learning [39]. Each scenario has distinct visuals and dynamics, and agents must learn them in sequence. The benchmark measures how well an agent can handle **embodied visual tasks** one after another, evaluating both forgetting and transfer. Early results on COOM showed that naive approaches struggle: agents often forget how to handle earlier Doom levels when trained on new ones [40]. However, methods with memory (like experience replay buffers) and dynamic architectures performed better, demonstrating *positive transfer* (using knowledge from a prior scenario to speed up learning on a later one). For instance, an agent that learned to navigate in a simpler maze level could transfer those skills to a harder maze, provided it didn't overwrite its navigation policy while learning intervening tasks. COOM has become a valuable test for **pixel-based continual RL**, pushing researchers to develop agents that can retain low-level perceptual skills (like recognizing enemies or items) and reuse them. One outcome from COOM and similar game benchmarks is the insight that **feature-level preservation** is critical: agents benefit from freezing or sparsely updating early convolutional layers (vision features) while adapting higher-level decision layers for new game modes. This partial update approach protected visual knowledge and prevented catastrophic forgetting of how to "see" in Doom, even as new objectives were introduced.

In classic control-and-graphics benchmarks, *Procgen* (procedurally generated game environments) and *Atari* have been adapted for continual learning evaluations [41] [42]. For example, researchers construct sequences of different Atari games or different Procgen game types to train one agent sequentially. A standout system in this area is DeepMind's **Agent57-DMN (2022)** – an extension of the Agent57 architecture with **Dynamic Node Management** to handle task shifts. While Agent57 (from 2020) was a multi-task agent with impressive performance, the 2022 variant allowed the agent to activate or deactivate parts of its neural network when switching games (similar in spirit to "sparse prompting" or modular activation). This allowed specialized sub-networks for each game to co-exist, yielding high scores on each game without forgetting. Concretely, Agent57-DMN achieved an average performance across 5 Atari games in a sequence that was *95% of the score it would have had if trained on each game independently*, a huge improvement over a standard DQN trained sequentially (which might fall to ~50% on early games after training the last one) [43] [44]. Although Agent57-DMN is a complex system (with memory, meta-controller, etc.), its success illustrated that **with the right architectural support, a single agent can approach single-task expertise even in a sequential training regime**.

Self-play and evolving opponents represent another dimension of continual learning in games. Take *AlphaStar* (DeepMind's StarCraft II agent, 2019) – while a bit earlier than 2022, its training scheme was effectively a continual learning scenario: the agent was trained via a league of opponents that grew over time, forcing continual adaptation. Each time AlphaStar's strategy would converge, new competitors (including past versions of itself or specialized adversaries) were introduced, and AlphaStar had to update

its policy to handle them. This can be seen as **online policy adaptation against a shifting environment** (the environment being the mix of opponents). The lessons from AlphaStar carried into 2022+ work on continual multi-agent systems: one needs to monitor for **policy cycling or destabilization** (if two agents continually adapt to counter each other, they might cycle strategies). Ensuring a diverse but bounded opponent set (as the league did) is one solution. More recent multi-agent continual RL research (2023–2024) introduced the idea of **fictitious play in a continual context**, where an agent maintains a portfolio of policies and chooses among them when a new adversary is encountered, updating only as needed. Such systems aim for *stationarity in the meta-game* so that learning can converge even as the agents co-evolve.

Multi-Task Learning Benchmarks and Performance

Continual RL has motivated dedicated benchmarks to quantify progress. We have mentioned **Continual World** (CW10) for robotics and **COOM** for games. Additionally, **Lifelong Learning in Atari (LLA)** and **CORA** (a realistic household robotics benchmark) have been proposed [45] [46]. A common evaluation protocol in these benchmarks is to measure: (1) the agent's performance on each task *after learning all tasks* (to assess retention), (2) the *average reward or success* across tasks, and (3) the *forward transfer* efficiency (how quickly new tasks are learned compared to if they were learned in isolation) [43]. Many papers report metrics like **average normalized score** (where 100% is the score of a task-specific expert) and **forgetting percentage** (drop in performance on tasks from when they were first learned to after all tasks).

On Continual World, for example, early baselines like fine-tuning or naive EWC achieved average normalized performance around 50–60% with significant forgetting, whereas the latest methods (2023–2024) are pushing that above 80% [22] [47]. One simple but strong baseline emerging is the "**Reset & Distill (R&D)**" method [26]: it resets the policy for each new task (so it learns fresh without interference) and then periodically distills the new policy's knowledge back into a multi-task policy. This approach was shown to *consistently outperform* more complex algorithms on long sequences of Meta-World tasks, achieving higher final success rates (often by 10-20% absolute) and lower forgetting [27]. The takeaway is that sometimes a *modular training schedule* (separate learning then merging) can beat end-to-end continual training. This has inspired practical systems where an agent might **maintain multiple sub-policies** and only merge them after ensuring stability.

Another noteworthy benchmark is **CORA**, which involves sequences of household robotic tasks with rich visual inputs [45]. CORA emphasizes *sample efficiency* and realistic observation (pixels, partial observability). It turned out to be extremely challenging: many state-of-the-art continual RL methods struggle to make progress on CORA's long task sequences [46]. Researchers observed that current agents either require *millions of frames per task* (which is not sample-efficient) or they collapse when faced with the high-dimensional observations. In response, some have proposed **simplified benchmarks** or **controlled variations** of CORA to develop the agents stepwise [46]. As of 2025, no agent achieves human-level performance across the full CORA task suite – highlighting that **continual RL in high-dimensional, realistic settings is still an open challenge**. The best performing systems on CORA use a combination of techniques: pre-trained visual encoders (to get a feature head-start), experience replay with balanced sampling, and occasionally human demonstrations to kickstart each new task. These help with sample efficiency and stability, but the overall performance still leaves plenty of room for improvement, according to recent evaluations [46].

Recommendation and Online Service Systems

Outside of games and robotics, online RL has been explored in systems like recommendation engines, advertising, and other user-interactive platforms. These systems face a *continual learning problem by nature*: user preferences evolve, new content/items appear, and the policy (recommendation strategy) must adapt in real-time. A common approach in industry is to use **contextual bandits** or **online actor-critic methods** that update a recommendation model after each user interaction (or batch of interactions). For example, a news recommendation system might use an RL policy to rank articles, receiving reward signals from clicks or dwell time, and update its model parameters daily or hourly. Key implementations (e.g. by YouTube or e-commerce sites) often use *off-policy RL* with replay: they log user interaction data and periodically retrain or fine-tune the model on the growing log (which serves as a replay buffer). This is essentially a continual learning loop, although often done in mini-batches offline for stability.

One notable system is **Horizon**, Facebook's open-source RL platform (released in 2018 and improved through 2022) which was used for personalized ranking tasks. Horizon's training pipeline continuously ingests new interaction data and performs updates to the policy network (usually a deep Q-network or policy-gradient model) with careful constraints. In a live A/B test reported by Facebook, a continual RL-based ranking policy increased user engagement metrics compared to a static supervised baseline, but only after the engineers introduced strong **safety checks** – for example, constraining the action space to not show obviously irrelevant or sensitive content during exploration, and using **reward normalization** to prevent the algorithm from chasing outlier user behaviors. Modern recommendation RL systems therefore implement **guardrails for safe exploration**: they might require a new policy version to be sandbox-tested (shadow mode) before full deployment, or blend the new policy with the old one (e.g., ϵ-greedy serving where with probability ϵ a new recommendation is shown, otherwise the old policy's recommendation is used) to limit risk. These practical techniques echo themes in academic continual RL: maintaining a fallback to a known-good policy is akin to preserving old knowledge, and slow update schedules/low learning rates mirror keeping changes incremental for stability.

Another emerging area is **large language model (LLM) self-learning** in deployment, which can be seen as a form of continual RL in an online service. The SEAL system from MIT (2025) is a prime example: it allows an LLM (like GPT) to improve itself by generating new training data and optimizing on it via RL [48] [49]. While not a traditional control policy, the LLM effectively plays an RL game of improving its own performance, with a reward for better accuracy on tasks. SEAL's successful demonstration (improving QA accuracy by over 13 points post-deployment) suggests that even deployed AI services could use *continual reinforcement loops* to stay up-to-date [29]. However, this also underscores challenges: the authors had to address catastrophic forgetting (the model's new fine-tuning causing dips in prior capabilities) and found that the RL component was key to mitigating this [32]. They also pointed out *latency* concerns – each self-improvement cycle took tens of seconds and required a specialized infrastructure to update the model on the fly [33]. In a production setting, this kind of overhead and complexity means truly **autonomous self-learning systems are still rare**, but research prototypes are paving the way.

Trade-offs and Challenges

Designing an agent that adapts continually involves balancing several trade-offs:

- **Sample Efficiency vs. Cumulative Training Time:** Ideally, an agent leverages prior knowledge to learn new tasks faster than from scratch (**forward transfer**), improving sample efficiency. Many

approaches (e.g. meta-RL, knowledge distillation, task-agnostic exploration) explicitly aim for this [50] [51]. For instance, the TAPD algorithm (2024) introduced a task-agnostic pre-training phase where the agent explores without extrinsic rewards and builds general skills, which later *reduced samples needed* to achieve high reward on downstream tasks [50] [51]. However, accumulating experience over a lifetime means the total amount of training can become very large. Systems must decide how much past data to retain and replay – too little and the agent forgets, too much and training each new task becomes slow (diminishing returns). Some benchmarks measure **learning speed** on each task to ensure that as we improve retention, we're not sacrificing the benefit of prior learning. The best continual RL methods manage to both avoid forgetting *and* accelerate learning of new tasks compared to a fresh agent. Achieving this dual win is non-trivial; negative transfer often occurs, where prior knowledge slows new learning (hence the need for methods like R&D that reset parts of the network to avoid interference [8] [26]).

- **Catastrophic Forgetting vs. Remembering Irrelevant Knowledge:** Catastrophic forgetting is the prime challenge in continual learning. Techniques like replay, regularization, and modular networks all combat forgetting, but there's a subtle trade-off: if the agent *never* forgets anything, it might carry along outdated or irrelevant knowledge that could interfere or consume capacity. In a non-stationary world, some forgetting is actually necessary (or at least, old behaviors should be overridden when they become suboptimal). The concept of **plasticity loss** [9] highlights that standard deep nets tend to become too stable (plasticity decreases) over time – essentially an extreme form of stability that hinders learning. The Nature 2024 study showed that without intervention, a deep RL agent training on a long sequence will plateau in performance as if it were a shallow network, unable to significantly improve on new data [9] [19]. On the other hand, methods that encourage plasticity (e.g. random unit reset, orthogonal weights) must be tuned carefully to not erode important prior knowledge. The **stability–plasticity dilemma** is thus at the heart of continual RL: each method navigates it differently. A large replay buffer leans towards stability (preserving old data), but as one study found, using an overly large buffer can *hurt adaptivity*, making the agent too slow to adjust to new tasks [12]. Conversely, a very small buffer (or none at all) yields plasticity but at the cost of rampant forgetting [12]. The optimal middle ground often involves **selective replay** (sampling a balanced mix of recent and older experiences) or **local learning** (only adjusting parts of the network for new task, as in parameter isolation methods).

- **Latency and Computational Constraints:** Updating a policy during deployment introduces latency and resource consumption that static policies don't have. In real-time systems (e.g. robotic control or high-frequency trading), doing gradient updates or extensive planning on the fly may not be feasible within the time constraints of decision-making. Some continual RL systems address this by performing updates in the background or at scheduled intervals rather than every time step. For example, a robot might collect data for a day and update overnight, effectively implementing *daily online learning* instead of instant adaptation. In LLM self-learning (SEAL), the process required tens of seconds per update and special infrastructure, which would be impractical for an interactive chatbot to do for every conversation [33]. Therefore, architects of continual learning systems often decide on a **frequency of updates** that balances responsiveness with compute costs. There is also the question of model size growth (for modular methods) – adding parameters can eventually burden memory and increase inference time. Practitioners might cap the model size or periodically compress older modules ("progress & compress" strategies compress older knowledge to make room for new [52]). Another latency aspect is **inference stability**: if an agent's policy is changing while serving users, it might produce inconsistent results (for instance, a recommendation algorithm

oscillating in what it shows). Smoothing techniques, like slowly updating target networks or averaging weights (Polyak averaging), can ensure that the deployed policy changes more slowly than the learning updates, thus giving more stable behavior at the cost of a slight lag in incorporating new knowledge.

- **Exploration vs. Safety:** Continual learning agents must continue exploring to discover changes or improvements, but exploration in a deployed setting can lead to unsafe or undesirable actions. In a controlled training phase, an RL agent might try highly exploratory moves (even crazy ones) to learn; in a live system, this could be harmful (a robot might break itself, an ad recommender might show offensive content to test a hypothesis). Thus, **safe exploration** is crucial. Approaches to this include adding **safety constraints** (the agent is not allowed to take actions that violate certain safety conditions) and using **risk-sensitive rewards**. We saw an example in Safe CDA where RL updates were constrained to avoid unsafe actions on the real robot [34]. Another strategy is **simulated exploration**: use a simulator or model of the environment to try out new behaviors before applying them for real. For instance, an autonomous driving system could maintain a simulated twin of its car to experiment with new control policies in the simulator; only when a policy seems promising and safe does it get tested in the actual vehicle (and even then under careful monitoring). Furthermore, *human-in-the-loop* safety is often used in practice: if a learning agent proposes a potentially unsafe action, a human overseer or a rule-based checker can override it. This can be seen as a temporary scaffold until the agent has seen enough to know not to choose such actions on its own. Balancing exploration and exploitation in continual RL also ties to the **long-term vs. short-term performance** trade-off: a deployed agent might sacrifice some immediate reward (or user satisfaction) to explore a new strategy that could yield higher rewards in the future. Companies deploying RL in user-facing systems often set explicit limits on this (for example, limiting what fraction of users see experimental recommendations) to manage the risk.

- **Performance vs. Robustness:** Continual adaptation ideally improves performance on new data, but it can sometimes *degrade* overall robustness. A model that keeps changing might overspecialize to recent conditions and become brittle to rare events. Best practices here involve **regular evaluation** on a variety of scenarios or past data to catch if the policy is drifting undesirably. Some researchers propose **ensemble methods** where a portfolio of policies (including older ones) vote or are available to choose from if the newest policy falters. The dual-actor idea from the negative transfer study is an example: one actor periodically resets for plastic adaptation, another (more conservative) actor distills knowledge and retains robustness [53]. The system can fall back to the distilled actor if the fresh learner goes awry. Similarly, **monotonic improvement algorithms** (like certain policy gradient methods with trust regions or safe policy update rules) ensure each update *probably* won't reduce performance beyond a small ϵ. This kind of guarantee becomes vital when learning online; it might be better to learn a bit slower but never catastrophically fail. Designing continual RL algorithms with theoretical guarantees (e.g. on performance regret or safety constraints satisfaction) is an ongoing research frontier.

Best Practices and Architectural Patterns for Safe Continual Adaptation

From recent advancements, a set of best practices and design patterns is emerging for building systems that learn continually and safely:

- **Modularize and Isolate New Knowledge:** Wherever possible, architect the agent such that new information can be added in a *localized* way. This could mean using modular networks (new task = new module) or latent context variables to switch behaviors. The benefit is clear boundaries between old and new knowledge, reducing unintended interference. For instance, adding a new module for a new task (as in progressive networks or CompoNet) ensures old task performance remains intact by default [22]. Even if the model can't grow indefinitely, in the short term this avoids many pitfalls. Once several modules accumulate, one can consider compressing or merging them in an offline phase (as done in "Progress & Compress" algorithms [52]). **Pattern:** *Use one network per context, then distill.* This pattern uses a fresh model for each new context (preventing interference during learning) and later distills all knowledge into a single network for efficiency [26] [27]. It has proven effective to sidestep negative transfer and then reconcile knowledge periodically.

- **Maintain a Replay or Memory Buffer:** Having a memory of the past is almost essential. Best-performing continual RL systems keep either a replay buffer of past transitions or a learned model of past tasks (generative memory). The buffer doesn't always need to store raw data – some use **compressed episodic memory** or **summary statistics** to save space. The **MIR** (Maximally Interfered Retrieval) algorithm (2020, extended later) is a strategy where the agent selectively replay experiences that it would currently do worst on, thereby directly counteracting forgetting. In 2022–2025, many variations of this idea appear: e.g., sampling from the buffer in a way that maximizes loss on old tasks, to "wake up" those memories during training. **Practice:** When implementing replay, ensure a *balanced sampling* between recent and old experiences, or use schedule like "mostly new data but X% old data". Too much old data can freeze learning (plasticity loss), too little invites forgetting – tuning this mix (or using adaptive strategies as in MIR) is crucial [12].

- **Regularize to Protect Core Abilities:** Identify which parts of the policy are critical to preserve (for safety or performance) and apply regularization or low learning rates on those. For example, if a walking robot has learned a stable gait, the core gait parameters might be marked "important" (via something like EWC's Fisher information or simple heuristics) and thus updated very conservatively when adapting to new terrain. Less critical parts (like how to handle a specific obstacle) can be adapted more freely. This *selective stasis* ensures that no update catastrophically ruins a foundational skill. In practice, techniques like **L2 regularization towards previous weights**, **EWC penalties**, or **constrained optimization** (don't violate known good behavior) are used. The safe RL + EWC study for nonlinear control did exactly this: they modified EWC to prioritize remembering safety-critical parameters by shaping the reward, effectively locking in safety while still learning new task dynamics [36].

- **Monitor and Limit the Update Magnitude:** A safety net is to impose limits on how much the policy can change with each update. This can be done by using algorithms with built-in trust regions (e.g. TRPO or PPO with a small KL-divergence threshold) so that each policy update is a gentle change. Another approach is **learning rate scheduling**: lower the learning rate as the agent accumulates

knowledge, to make it more cautious in later stages. Google's **Lifetime Value** project (2025) even suggested *lifetime tuning budgets* – only allow the agent to fine-tune on a small fraction (say k%) of new data to avoid it drifting too far [54]. While that was more about fair evaluation, the concept translates to practice as a guardrail: you don't let the agent fully overwrite itself on new data, just a limited optimization that hopefully finds a compromise policy. Additionally, one can use **change detectors**: if an update suddenly causes a large drop in performance on a validation set or a held-out old task, automatically roll back that update. Some continual RL frameworks include a human or automated "gate" that tests a new policy on a suite of tasks and only accepts it if it passes certain criteria (like not reducing score on past tasks beyond a threshold). This is analogous to continuous integration testing in software – continuous learning should have continuous evaluation.

- **Incorporate Intrinsic Rewards for Exploration:** To ensure the agent continues to explore appropriately in the long run, many systems add an *intrinsic motivation* signal (novelty reward, curiosity, surprise). This is especially helpful in task-agnostic scenarios where the agent must adapt without explicit task boundaries [55] [56]. For example, an agent with a curiosity drive will keep probing parts of the environment it hasn't mastered, which can lead it to notice when something has changed or a new opportunity arises. TAPD (2024) used an intrinsic reward phase to let the agent **"roam" and build general skills** before any specific task, and periodically between tasks, which improved its ability to tackle new tasks quickly [50] [57]. The lesson is that *exploration shouldn't stop* just because the agent has deployed – but it must be done safely and purposefully. By confining exploratory behavior to known-safe actions or low-stakes situations (e.g. a recommender system might try exploring recommendations on a small random subset of users or in a simulation of user behavior), one can gain the benefits of continual exploration without undue risk.

- **Use Pre-Trained Models and Feature Fixation:** A practical pattern in 2022+ is to leverage large pre-trained models (from supervised or self-supervised learning) as a starting point for RL, and then **freeze** significant portions of the model. This has been seen in computer vision for RL (using a pre-trained ResNet and not fine-tuning most of it), and in language (using a frozen language model as the base for an RL fine-tuning). The rationale is that these models provide broad, general representations that are less likely to catastrophically forget because they are not being updated much. The RL part (like a policy head) learns on top of these features. If a new task arises, ideally the feature extractor already has the capacity to represent it, limiting the necessary changes to the policy head only. Empirically, this has led to more stable performance in continual settings with high-dimensional observations, as the fixed backbone ensures previously learned perception doesn't drift. Of course, the downside is potentially less optimal representations for new tasks, but techniques like **adapter modules** (small trainable additions to a frozen network) can give some flexibility. The Amazon Science team reported in 2023 that in a continual learning setup for vision, *Low-Rank Adaptation (LoRA) adapters* allowed them to achieve almost the performance of full fine-tuning while the majority of the network stayed fixed [58]. This greatly reduced forgetting because the original weights (which encoded a lot of useful general vision features) never changed.

- **Plan for Evaluation and Recovery:** Finally, a best practice is to evaluate continual learning systems not just on average reward but on worst-case and initial phases. Monitoring for **regression** on old tasks or overall performance drop is key. If detected, the system should have a recovery mechanism: e.g., revert to a previous policy checkpoint (many deployable systems keep a rolling checkpoint of the last stable policy), or reduce learning rate to stop the degradation. The concept of **"graceful forgetting"** is also floated in research – if you must forget something to make room for new

learning, do so in a measured way (e.g., gradually fade out older skills that haven't been used or needed for a long time). Some algorithms include a usage count for memory entries or modules and will compress or remove those that are rarely invoked, on the assumption that they are not important anymore. This ties into *interpretable knowledge*: knowing what the agent has learned for each task can help decide what can be pruned when adapting further [59]. Ensuring transparency (so developers know which part of the network is responsible for what skill) can make continual RL more reliable in practice.

In summary, the period 2022–2025 has seen significant progress in **online and continual reinforcement learning** across various fields. We now have a clearer understanding of how to define the problem (agents that *never stop learning* [1] [2]), how to measure success (balancing average performance, forgetting, and transfer [43] [44]), and a toolkit of methods to achieve continual adaptation (from replay buffers to meta-learning to modular policies). Impressive systems have demonstrated that continual RL is feasible – e.g. robots learning successive tasks without forgetting [22], game agents approaching expert-level on sequential games, and even language models improving themselves over time [29] [32]. Yet, challenges remain in scaling these systems safely and efficiently. The trade-offs of plasticity vs. stability, exploration vs. safety, and adaptation speed vs. reliability will continue to guide research. By following emerging best practices – such as isolating new knowledge, leveraging memory, constraining updates, and monitoring policy changes – we can build agents that **learn in perpetuity** in a controlled and robust manner, bringing us a step closer to truly lifelong learning AI [1] [3].

---

[1] [2] [3] [2307.11046] A Definition of Continual Reinforcement Learning
https://ar5iv.org/abs/2307.11046

[4] [9] [10] [19] Loss of plasticity in deep continual learning | Nature
https://www.nature.com/articles/s41586-024-07711-7?error=cookies_not_supported&code=4c69efae-f696-439d-b653-4e0d556a2804

[5] [7] [11] [41] [42] [43] [44] [45] [46] [59] A Survey of Continual Reinforcement Learning
https://arxiv.org/html/2506.21872v1

[6] [50] [51] [52] [55] [56] [57] Continual deep reinforcement learning with task-agnostic policy distillation | Scientific Reports
https://www.nature.com/articles/s41598-024-80774-8?error=cookies_not_supported&code=a58448de-2894-49b7-b8a2-f571fa385fe5

[8] [26] [27] Prevalence of Negative Transfer in Continual Reinforcement Learning: Analyses and a Simple Baseline | OpenReview
https://openreview.net/forum?id=KAIqwkB3dT

[12] [13] [14] [15] [16] [18] [20] [21] [22] [23] [24] [25] [47] [53] [54] GitHub - datake/Papers-Of-Continual-RL: Related papers for Continual Reinforcement Learning.
https://github.com/datake/Papers-Of-Continual-RL

[17] [35] [36] [2502.15922] On the Design of Safe Continual RL Methods for Control of Nonlinear Systems
https://arxiv.org/abs/2502.15922

[28] [34] [2503.10949] Safe Continual Domain Adaptation after Sim2Real Transfer of Reinforcement Learning Policies in Robotics
https://arxiv.org/abs/2503.10949

29 30 31 32 33 48 49 58 Self-improving language models are becoming reality with MIT's updated SEAL technique | VentureBeat
https://venturebeat.com/ai/self-improving-language-models-are-becoming-reality-with-mits-updated-seal

37 38 ContinualWorld: Robotic Benchmark for Continual Reinforcement Learning
https://deepsense.ai/resource/continual-world-a-robotic-benchmark-for-continual-reinforcement-learning-2/

39 [PDF] COOM: A Game Benchmark for Continual Reinforcement Learning
https://proceedings.neurips.cc/paper_files/paper/2023/file/d61d9f4fe4357296cb658795fd7999f0-Paper-Datasets_and_Benchmarks.pdf

40 COOM: A Game Benchmark for Continual Reinforcement Learning
https://proceedings.neurips.cc/paper_files/paper/2023/hash/d61d9f4fe4357296cb658795fd7999f0-Abstract-Datasets_and_Benchmarks.html