ChatGPT

Continual and Online Reinforcement Learning Systems

Introduction

Most machine learning models are trained once and then deployed without further learning. However, many real-world scenarios are **non-stationary**, and AI systems benefit from *continual reinforcement learning* – the ability to **learn from new experiences after deployment** while retaining prior knowledge [1] [2]. This online adaptation is crucial for agents to handle changing environments or tasks (e.g. a service robot encountering new objects, or a language model receiving feedback on answers). The challenge is doing this safely and stably: naive online training can cause *catastrophic forgetting* of earlier skills [3], instability, or unintended behaviors. Recent research (2023–2025) has focused on algorithms and architectures to enable **online policy updates** without sacrificing reliability. Below, we review examples of such systems in **large language models (LLMs)** and **traditional RL domains** (robotics, games), the approaches they use (from full model fine-tuning to meta-learning), different feedback setups, and how they address trade-offs in latency, stability, exploration, and safety during continual updates.

Online RL in Large Language Models

Early large language model alignment (e.g. ChatGPT's RLHF) used *offline* RL – gathering human preference data and fine-tuning the model in a batch process [4]. Newer systems explore **online or continual RL for LLMs**, updating the policy during live interaction or in iterative deployments. For example, Meta AI compared offline vs online alignment on instruction-following and math tasks [5] [6]. They found that **continual updates dramatically improved performance**: policies trained with *semi-online* or *fully online* RL outperformed offline RL by a large margin on both verifiable tasks (math question accuracy) and open-ended tasks (win-rates in dialogue) [7] [8]. Notably, *semi-online* training – syncing the model with new experience periodically rather than every step – achieved comparable gains to fully online training, indicating that frequent but partial updates can suffice [9] [10]. This suggests a practical compromise where the model is updated in batches during deployment (to reduce overhead) while still benefiting from continual learning.

Another example is **SCoRe (Self-Correction via RL)** by Google/DeepMind, which trains LLMs to *improve their answers over multiple turns* [11]. SCoRe uses a **multi-turn online RL** approach: the model generates an initial solution (for a math or code problem), then iteratively attempts self-corrections, using *verifiable outcomes* (e.g. whether a math answer is correct or code passes tests) as the reward signal [12]. Through this online interaction with itself, the model learns a policy for self-correction. Importantly, SCoRe uses regularization and a careful two-phase training (RL initialization + reward bonus) to avoid model collapse [13]. The results are significant – applied to math and coding benchmarks, *online self-correction RL* boosted solution success rates by **15.6% on MATH** and **9.1% on HumanEval** compared to the base model [14], **achieving state-of-the-art performance** on those tasks. This illustrates how continual learning with *verifiable rewards* (automatic correctness checks) can greatly enhance LLM reasoning abilities.

It's worth noting that fully live online RL with human users (e.g. learning from user feedback in real time) is still rare due to safety concerns. Research prototypes exist – for instance, an LLM could be deployed as a chatbot that updates its responses policy based on user upvotes/downvotes (*bandit feedback*). In practice, most LLM systems instead collect feedback and retrain offline periodically. Nonetheless, the above examples show that **continual RL for LLMs is an active area**: using *reward models vs. rule-based rewards*, and studying how to blend **human preference signals (non-verifiable)** with **ground-truth outcome signals (verifiable)**. In Meta's study, training on both types of feedback jointly yielded the best generalization, but they had to mitigate **reward hacking** (the model exploiting the learned reward model in unintended ways) by combining human and verifiable rewards [15] [16]. Overall, LLMs can be continually refined via RL, but it requires careful design (e.g. PPO/GRPO algorithms with KL penalties [17] [18], or conservative updates) to ensure the model improves without diverging from its aligned behavior.

Online RL in Robotics and Games

Multi-task robotic agents like DeepMind's RoboCat can continuously learn new tasks across different robot embodiments. The image shows RoboCat manipulating various objects; this AI was fine-tuned on each new task and generated its own practice data, enabling rapid adaptation with minimal human demos [19] [20].

In robotics and control, continual RL is pursued to let robots adapt on the fly to new goals or changing conditions. **DeepMind's RoboCat (2023)** is a prime example of a *self-improving robotic agent*. RoboCat was initialized as a **vision-language-action model** (based on the Gato transformer) trained on many demonstration trajectories across **hundreds of tasks** and **multiple robot arm types** [19] [21]. After deployment, it entered a *self-improvement cycle*: for each new task or robot, only **100–1000 human demonstrations** were needed to fine-tune a policy [22], which then **practiced the task autonomously ~10,000 times** to generate more training data [23]. That new experience was added to RoboCat's dataset, and the agent was retrained (or incrementally updated) on this expanded data [24]. This iterative online learning allowed RoboCat to continually broaden its skillset **without forgetting earlier tasks** – effectively leveraging *experience replay* by mixing old and new data. The result was an agent that can operate **across 4 different robot arm types** and solve a wide range of manipulation tasks, generalizing faster with each expansion (some tasks learned **with only 100 demos** where previous models required far more) [22]. RoboCat demonstrates how **full-model fine-tuning plus experience replay** can achieve continual learning in robotics; the key is a large, flexible model and the ability to incorporate new experience while preserving prior knowledge (here, by retraining on the cumulative dataset).

Other robotics works focus on **safety and stability during online updates**. A challenge in on-policy RL is that training a large model on real robots can be *unstable* and sample-inefficient. The iRe-VLA framework (2025) addressed this for a vision-language-action model controlling a robot arm [25] [26]. The approach was to alternate between **online RL** and **supervised fine-tuning**: during RL phases, they *froze* most of the pretrained model (to protect its visual-language representations) and only trained a lightweight *action head*, to avoid destabilizing the entire network [27]. In subsequent supervised phases, the whole model was fine-tuned on successful trajectories collected in the RL phase [28]. This two-stage *adapter-style update* proved far more stable than end-to-end online RL. In fact, directly applying standard RL (PPO) to the large model caused performance to **collapse below the pre-trained policy** on several tasks, even when using experience replay [29]. By contrast, the iterative iRe-VLA method steadily improved the policy **without forgetting prior tasks** [30]. On a robotic manipulation benchmark, the continual learner achieved **83% success vs 43% for the base policy** on a previously hard task (e.g. opening a left-door cabinet) [31], and even **improved success on unseen new tasks by ~24%** through adaptation [32] [33]. This showcases the

effectiveness of **partial updates (training a small part of the network)** combined with interleaved supervised learning to ensure stability.

In the domain of locomotion, researchers have explored **dynamic architectures** to avoid forgetting when a robot faces a sequence of control tasks. For instance, Gai *et al.* (2024) tackled *continual quadruped locomotion* – where a four-legged robot must learn to walk on different terrains sequentially [34] [35]. They employ a **parameter isolation** strategy: for each new terrain/task, a subset of network parameters is designated as "protected" (retained from previous tasks) while unused parameters are reset for learning the new task [36]. By *freezing* or locking the weights important for earlier behaviors and only altering other parts, the robot could **avoid catastrophic forgetting** of how to walk on earlier terrains [37]. They also add an **entropy reward bonus** to encourage continued exploration (preventing the policy from becoming too narrow) [38] [39]. Compared to standard continual learning methods, this approach was **more stable and less disruptive** to training – the robot maintained competence on prior terrains while acquiring new gaits [40]. It highlights an architectural approach (modular networks with task-specific components) for safe lifelong learning in control problems.

Continual RL has also been demonstrated in complex **games and simulated environments**. A notable example is DeepMind's **Open-Ended Learning (XLand)** platform [41] [42]. In XLand (published 2021, with ongoing improvements), agents train in a procedurally generated 3D world with **millions of possible games** and objectives. The training system dynamically adjusts the tasks presented to ensure the agent is always challenged – in effect, the agent **"never stops learning"** because the curriculum keeps evolving as the agent improves [43]. Through this continual, self-curated training, a single agent learned to succeed in a **wide spectrum of games – including some games (like new variants of hide-and-seek) that it never encountered during training** [44]. The agent developed general skills like *experimentation* and adaptation, rather than memorizing specific solutions [45]. This is enabled by a *meta-game curriculum* that continually introduces novel tasks (leveraging billions of training rounds) and by multi-agent self-play that forces constant strategy refinement. Similarly, earlier systems like AlphaStar in StarCraft II (2019) utilized **league-based continual training**, where the AI played against a growing pool of opponents (including past versions of itself), steadily improving and avoiding overfitting to any single strategy. AlphaStar's policy was updated in *near-real-time* through thousands of games, eventually reaching Grandmaster level. These game agents illustrate how **self-play and task diversification** serve as a form of continual RL, yielding highly adaptable policies. The trade-off is the enormous scale of experience required (AlphaStar experienced the equivalent of **200 years of gameplay** during training, and XLand agents played **200 billion** game rounds [46]) – highlighting a sample efficiency challenge for continual RL in rich environments.

Finally, in real-world **recommendation systems**, online RL (often in the form of contextual **bandits**) has been deployed to continually adapt to user preferences. For example, YouTube reported a **real-time bandit system** (2023) that updates its recommendation policy on the fly from user clicks and feedback [47]. Instead of retraining a deep model for every new piece of data (which would be too slow), they use a hybrid approach: a base ranking model is trained offline on historical data, but a lightweight **LinUCB bandit** layer runs online to adjust item exploration/exploitation for fresh content [48]. This closed-loop system handles **massive scale (millions of users)** by distributing the bandit computations. A key consideration here is the **exploration–exploitation trade-off**: purely exploiting the learned preferences can miss new content, but exploring too much can degrade user experience. The YouTube system specifically noted that *uncertainty-driven exploration* must be carefully managed to avoid showing irrelevant content to users [47]. Their solution improved the discovery of new videos (serendipitous recommendations) **while maintaining user satisfaction**, as confirmed by A/B tests on the live platform [49]. This example underscores that continual RL

need not always mean “learn a deep network end-to-end online” – often a **partial update (bandit or adapter)** can integrate new feedback with low latency and risk, achieving a balance of adaptability and stability in a production system.

Approaches for Online Policy Updates

Continual RL systems employ various approaches to update policies after deployment, each with pros and cons:

- **Full Policy Fine-Tuning:** The most direct approach is to *update all parameters* of the policy network with new data (e.g. via gradient descent) during online learning. This is powerful (maximizes capacity to learn new behaviors) but high-risk – large updates can destabilize the policy or overwrite previous knowledge. Many successes in RL have used full-network updates with safeguards: for instance, **Proximal Policy Optimization (PPO)** and related algorithms apply small, trust-region constrained policy updates to maintain stability [50] [51]. OpenAI’s and DeepMind’s large-scale RL agents (AlphaZero, AlphaStar, etc.) continuously fine-tuned the entire policy network, but they relied on massive experience and careful hyperparameters to avoid divergence. In LLM alignment, early RLHF fine-tuning also adjusted all weights, often with a KL penalty against the original model to prevent it from straying too far. Full-model updates are thus feasible, especially if one uses **on-policy methods (PPO/GRPO)** to limit each step’s change [18] [51] or if one periodically mixes in older data to rehearse old tasks. The Meta study showed that even with full model updates, a *semi-online schedule* (periodic syncing) can yield stable improvements [52] [10]. Still, the computational burden is high – every online update involves backpropagating through a huge model (millions or billions of parameters), which may introduce latency or require distributed training infrastructure.

- **Adapter or Partial Updates:** To mitigate the above issues, many continual learners only update a subset of the model. This could mean **freezing the backbone and training small adapter modules** or an output head (as done in iRe-VLA [27]), or using techniques like **LoRA (Low-Rank Adapters)** in LLMs for on-the-fly fine-tuning. By limiting what changes, the system reduces the risk of catastrophic forgetting and can train faster (fewer parameters to adjust). For example, in robotics, **Piggyback-style dynamic networks** load task-specific weight masks for each new task, leaving previously learned weights untouched [36] [37]. Another partial update strategy is **experience injection**: e.g. in the YouTube bandit, the main deep model isn’t retrained live – instead a lightweight parameter (say, a bias for new items or a contextual weight) is updated online, effectively adapting the policy slightly without altering the core ranking model. Partial updates tend to be **safer (less chance of destabilizing)** and allow quick learning, but they may have limited capacity – the system might plateau if the small adapters cannot express the needed policy adjustment. In practice, a common pattern is to *do partial updates online for fast adaptation, and periodically do a full update offline* to consolidate knowledge.

- **Meta-Learning and Rapid Adaptation:** Meta-learning prepares a model to **learn quickly** from new experiences, effectively handling online updates within its architecture. One approach is using a **recurrent policy (RL^2)** that *learns to adapt* its behavior from past trajectory within the hidden state [53] [54]. Here, no explicit gradient update is made online; instead the agent’s internal state acts as memory that updates as it encounters new situations (learning “on the fly” in a forward-pass manner). Such agents have shown the ability to adjust to new goals or changed dynamics within a single episode [55] – for instance, an RNN-based RL agent can handle a **sequence of different**

bandit tasks or recover from a sudden change in a robot's limb (simulated break) by virtue of its meta-trained adaptive dynamics model [56]. Another meta-learning approach is **gradient-based**: e.g. *Model-Agnostic Meta-Learning (MAML)* trains the policy's initial weights such that only a few gradient steps on new data lead to large improvements [57]. MAML and its variants make it feasible to do very fast fine-tuning in an online setting (since the model is primed for quick convergence). However, meta-learning methods can struggle with long-term continual learning – e.g. an RNN-based meta-policy may still forget older tasks if task distribution shifts too much over time [58]. Recent research like **CoMPS (Continual Meta-Policy Search)** attempts to update meta-policies gradually as tasks arrive, to reduce forgetting [59]. In summary, meta-learning provides a *framework for quick adaptation* (often essential for high-frequency update needs or tasks requiring instant personalization), but it usually complements rather than replaces explicit online training, especially for long sequences of tasks.

- **Experience Replay and Off-Policy Learning:** A core technique to enable continual updates without forgetting is to **store past experiences and mix them during training**. In traditional RL, *experience replay* (reusing past transitions in off-policy algorithms like DQN) stabilizes learning by breaking correlation in updates and preserving older knowledge. In continual RL, replay buffers or **rehearsal of past tasks** is used to avoid catastrophic forgetting – for example, after the agent learns task B, it still trains occasionally on data from task A so that A's performance doesn't collapse. Many robotic continual learning systems explicitly maintain a memory of important episodes from previous tasks. The RoboCat agent effectively did this by aggregating all data in each retraining cycle [24], so no skill was ever truly forgotten – the **consolidation retraining on a growing dataset** served as a large-scale experience replay. Similarly, in LLM continual fine-tuning, one might keep a buffer of interactions or use the original pretraining data as a regularization (this has been proposed to prevent an aligning RLHF model from drifting too far from general capabilities). Off-policy algorithms are particularly helpful in online settings because they allow learning from *older data or other agents' data*. For example, an agent could keep improving using a mix of new online experiences and a cache of diverse past experiences, which can improve sample efficiency and guard against regressions. The trade-off is that **off-policy learning must be done carefully** to avoid bias – if the environment changes or the policy has moved far from old data, blindly replaying could mislead training. Techniques like importance sampling, or simply decaying the weight of very old data, are used to ensure replay benefits outweigh the drawbacks. Recent studies in continual RL note that combining **on-policy fine-tuning with occasional off-policy updates** (or vice versa) can yield robust performance.

The table below summarizes some of the **notable systems (2021–2025)** that use online or continual RL, highlighting their domains, update approaches, and key results:

System (Year)	Domain	Update Approach	Feedback Source	Notable Results
Meta's Online LLM (2025) [7] [8]	LLM Alignment	Full policy fine-tuning (PPO/ GRPO); offline vs semi-online vs online regimes	Human preference model (non-verifiable) + programmatic rewards (verifiable math)	Online (and semi-online) RL **significantly outperformed offline** on both math accuracy and open-ended instruction benchmarks (e.g. >+10% improvement in win-rate) [7] . Semi-online updates achieved **comparable gains to fully online**, indicating efficient trade-offs [9] .
SCoRe Self-Correction (2024) [13] [14]	LLM Reasoning	Multi-turn online RL with self-generated data; full model updates with regularization	Verifiable outcomes (math solution correct? code passes tests?)	Enabled **LLM self-improvement** over multiple turns, yielding **+15.6%** solve rate on MATH and **+9.1%** on HumanEval code benchmarks versus base model [14] . Achieved state-of-art on these tasks via purely self-supervised reward signals.
RoboCat (2023) [19] [22]	Robotics (multi-arm manipulation)	Iterative fine-tuning and *experience replay* (retrain on growing dataset); full model updates (transformer)	Human demos for new tasks + self-play experience (autonomous practice)	Learned to solve **100+ tasks across 4 robot arm types**. New tasks learned from only **100–1000 demos** (high sample efficiency) [22] . Retained earlier skills by cumulative training – *first* robot agent to demonstrate broad generalization across embodiments [19] .

System (Year)	Domain	Update Approach	Feedback Source	Notable Results
iRe-VLA (2025) [31] [33]	Robotics (vision-language action)	*Partial updates* (freeze vision-language backbone, train action head in RL), alternating with full-model supervised fine-tune	Sparse environment rewards (manipulation success/failure)	Standard online RL caused training collapse; iRe-VLA's two-stage update was **stable**, improving a robot's task success from **43% to 83%** on one fine manipulation and boosting success on unseen tasks from 37% to 61% [60] [33]. Maintained performance on original tasks while learning new ones.
YouTube Bandit (2023) [47] [49]	Recommender System	Real-time **contextual bandit** (LinUCB) updates on top of a fixed ranking model (minimal parameters updated)	User clicks and watch time (implicit bandit reward)	Deployed at scale (YouTube). Improved **fresh content discovery** by adaptively exploring recommendations [49]. Employed strategies to cap exploration to avoid hurting user experience [61]. Demonstrated practical hybrid offline+online learning with millisecond-level updates.
XLand Open-Ended RL (2021) [43] [45]	Games (procedurally generated)	Continual *self-play* with dynamic task generation (no weight resets, endless training curriculum)	Automated reward per game (win/lose objectives)	Trained a single agent on **billions of tasks** in a 3D world – the agent **"never stopped learning"**, evolving general skills. Showed zero-shot success on complex games (like hide-and-seek variants) *not seen in training*, due to emergent adaptation strategies [43] [45].

Differences in Feedback and Learning Setups

Continuous RL systems also differ in the **type of feedback or reward** driving the online updates, which affects their design:

- **Human-in-the-Loop Feedback:** Some systems learn from *human feedback given online*, such as preferences, ratings, or demonstrations. In language models, this is seen in **online RLHF** prototypes

where a user's thumbs-up/down on responses could adjust the policy. In robotics, approaches like **Deep TAMER** and others have allowed humans to give reward signals in real time to shape the agent's behavior. Human feedback is invaluable for aligning AI with human values or subjective goals, but it comes with high latency and noise – humans are slow and inconsistent reward providers. Thus, algorithms must be sample-efficient (to learn from few human examples) and robust to occasional wrong signals. Safety is also a concern: the system should not aggressively exploit the human (e.g. by annoying them for feedback or by overfitting to one teacher's quirks). Many deployed systems therefore use human-in-loop sparingly, or in combination with other signals. For example, **Meta's 2025 LLM** used a reward model trained on human preferences to provide a scalar reward continuously [62], effectively mediating human feedback into a learned automated reward to allow faster online optimization.

- **Self-Supervised or Automatic Rewards:** In some cases, the environment itself provides a **verifiable, automated reward** signal, enabling fully autonomous online learning. This is ideal for tasks with clear outcomes – e.g. winning a game, solving a puzzle, reaching a goal state. For LLMs, "verifiable" tasks include math problems (where a solution can be checked), code generation (check with test cases), or factual question answering (compare to a reference). The advantage is that the agent can train continually without human intervention, as in the SCoRe self-correction example where the model knows if its answer is right by consulting a checker [12]. Similarly, robotics tasks can have built-in success metrics (did the robot grasp the object? yes/no from a sensor). These *ground-truth rewards* remove ambiguity and allow standard RL algorithms to run online. The downside is not all tasks are easily quantifiable; many require complex or subjective evaluation. Also, even with automatic rewards, online learning must handle that the **feedback may be sparse or delayed** (e.g. a long-horizon task where success is only known at the end). Techniques like reward shaping or intermediate proxy rewards are often needed to make learning efficient.

- **Bandit Feedback:** This is a special case of automatic feedback where the agent only gets a signal for the **action it took**, not a full reinforcement signal for every possible action. Online advertising and recommender systems are classic examples: the system makes a recommendation (action) and only observes whether the user clicked (reward) or not – it doesn't directly learn what would have happened for the actions not shown. This *bandit setting* means the agent must continuously **explore** different actions to discover their reward, while exploiting what it currently thinks is best. Contextual bandit algorithms (LinUCB, Thompson Sampling, etc.) are commonly used for their strong theoretical guarantees and efficiency in this scenario [47] [63]. The YouTube system above falls into this category. In bandit feedback, safety considerations revolve around ensuring exploration doesn't degrade the user experience or system performance too much. Often, a fraction of traffic is set aside for exploration, or one uses *optimistic initializations* to gradually ramp up exploration. The **latency for learning** is typically lower than full RL because each interaction is one step (no long episode to simulate) – this makes bandit algorithms attractive for real-time continual learning.

- **Demonstrations and Imitation Online:** Another form of feedback is **on-the-fly demonstrations or corrections** provided by a human or an expert policy. For instance, in robotics, if the agent encounters a novel situation, a human operator might teleoperate the robot briefly (providing a demonstration), which the agent then uses to update its policy (via imitation learning) during deployment. This can be seen as a form of continual learning (the policy improves its performance distribution with each human intervention). It is particularly useful for safety – instead of letting the robot trial-and-error in a dangerous situation, a human can guide it, and the robot learns from that

guidance. Some recent works integrate such human corrective feedback with RL updates, achieving a blend called *reinforcement learning with expert interventions*. This wasn't explicitly requested in the review, but it is a growing paradigm to ensure safety during online learning: the agent *asks for help* or a human steps in when the agent's confidence is low, thereby preventing catastrophic failures and simultaneously giving the agent new training data for that scenario [64] [65].

In summary, **verifiable and self-supervised feedback** enables fully autonomous online RL (great for games, simulations, or well-specified tasks), **human and bandit feedback** are crucial for alignment with user preferences and real-world deployment (but require careful handling of uncertainty and noise), and **hybrid setups** (like combining human rewards with verifiable signals [15] [16]) can yield the best of both. The design of a continual RL system must therefore consider what feedback is available at inference time and choose algorithms accordingly – e.g. on-policy RL for rich dense rewards, vs preference learning or bandit optimization for sparse human evaluations.

Architectures and Techniques for Safe Continual Learning

Ensuring *safe and stable* online updates is a major focus of recent systems. Several best practices and architectural choices have emerged in the last few years:

- **Trust-Region Updates:** Algorithms like PPO and its variants remain popular for online policy learning because they inherently limit the size of each policy update (via KL-divergence penalties or clipping) [18] [51]. This prevents the policy from changing too rapidly in response to a single batch of feedback – a crucial safety measure to avoid wild oscillations or mode collapse. For example, GRPO (used in LLM alignment) builds on PPO and was shown to stably train very large models online [50] [18]. Similarly, in continuous control, trust-region methods (or natural policy gradients) ensure that when the robot learns from a new experience, it doesn't unlearn how to walk altogether. **Conservative Q-learning** techniques in off-policy RL also aim to keep the value function updates within reasonable bounds, to avoid overestimation or exploitation of novel data. Overall, using algorithms with theoretical monotonic improvement guarantees or clipped updates is a de facto standard for safer continual RL.

- **Regularization and Constraints:** Beyond algorithmic constraints, adding explicit regularization can help maintain prior capabilities. **Elastic Weight Consolidation (EWC)** and other continual learning regularizers penalize changes to weights that were important for previous tasks, thereby resisting forgetting. Such techniques (originally from supervised continual learning) have been applied in RL – e.g. an agent can carry an importance matrix for its network parameters and gently push back on gradient updates that conflict with important weights. In LLM online tuning, one might regularize against the original model or a static copy (as a "reference model") – indeed, KL regularization against the pre-trained model is common in RLHF to ensure the tuned model doesn't drift into nonsensical or unsafe regions of policy space [4] [17]. Some 2024 works on LLM continual learning propose selective memory rehearsal (training on a small representative set of original data alongside new data) to retain general language ability [66] [13]. In robotics, one could enforce that certain known-safe actions remain available by not overwriting those action policies during fine-tuning. All these acts as **"anchors" to prior knowledge** while the model learns new things.

- **Modular and Multi-Model Architectures:** A trend for safety is to break the problem down – e.g. use one model for making predictions and another for deciding when to trust those predictions or when

to update. For instance, a **meta-controller** can monitor the performance of the primary policy and veto or revert changes that reduce performance on a validation set. In human-facing AI, a deployed system might maintain a fallback model (perhaps the last stable version or a rules-based system) and if the online-updated model behaves erratically, the system can switch to the safe fallback. Some architectures include multiple sub-policies (ensemble methods) and gradually shift weight to a new sub-policy as it proves better, rather than replacing outright. This *gradual handover* can prevent sudden regressions. The **dynamic architecture** approach in the quadruped example is another case – by compartmentalizing each task's knowledge into separate parameters, the system ensured new learning wouldn't interfere with old skills [37]. Likewise, *progressive networks* (adding new columns of neurons for new tasks) or *mixture-of-experts* models can learn continually by allocating new capacity instead of overwriting old weights. These come with a cost (growing model size), but research is investigating ways to prune or merge experts to manage complexity over time.

- **Safety in Exploration:** Continual learning often implies the agent will keep exploring to improve. Ensuring **safe exploration** is critical, especially in the real world. Techniques for safe RL – such as **reward penalties for entering unsafe states, constraint satisfaction via Lagrangian methods, or shielded policies** – are increasingly combined with online learning. For example, if a robot is learning to navigate a building, it can have a *safety shield* (maybe a classical controller or a supervised model) that overrides actions which would lead it down staircases or into collisions, while the RL agent explores other aspects of navigation. As another example, some systems incorporate human oversight for exploration: an agent might explicitly ask a human "can I try X?" if X is uncertain and potentially risky, rather than just doing it. In training autonomous vehicles online, one would use safety drivers or simulation-to-reality checks to intervene if the policy update starts to cause dangerous driving. **Curriculum learning** also plays a role: as seen in XLand, the task generator can ensure the agent only tackles challenges appropriate for its current skill, reducing the chances of catastrophic failure. In essence, maintaining **bounded risk during learning** is necessary for deployment; it's achieved by a combination of external constraints, cautious policy update rules, and sometimes, **stochastic smoothing** (e.g. adding entropy reward as done for the quadruped [38] so the policy doesn't get stuck in a narrow, possibly unsafe behavior).

- **Monitoring and Evaluation:** A practical but important aspect is continually evaluating the online-updated model against benchmarks or test scenarios. Many systems use a hold-out set of tasks to detect if performance on known tasks drops after an update. If it does, the update might be rolled back or adjusted. For example, in Meta's online LLM training, they monitored both the *verifiable tasks and non-verifiable preference scores* as training progressed and selected checkpoints that best balanced the two [67] [9]. Such evaluation helps catch reward hacking or skill deterioration early. In production systems (like an online service), A/B testing and gradual rollout are used – the new updated policy might be deployed to a small percentage of users, metrics are checked (click-through rate, error rate, etc.), and only then widened. This engineering practice ensures that any negative effect of an online update is limited in scope.

By combining these architectural and algorithmic measures, modern continual RL systems strive to be **both adaptable and reliable**. For instance, the iRe-VLA robotics framework embodied many of these: it constrained updates (partial parameters), used supervised rehearsal between RL bursts, monitored that performance only went up, and found it beneficial to freeze high-level perception layers to avoid unwanted drifts [30] [29]. Similarly, the LLM alignment experiments introduced semi-online schedules to get the gains of online training without some instabilities (like response length divergence) [10] [68]. The takeaway is that

online learning must be engineered with guardrails – freewheeling adaptation can and will lead a complex model astray (examples abound of agents exploiting loopholes in reward functions). The best systems of the last few years use a combination of theoretical insights (provable regret bounds [69], convergence guarantees) and empirical safety nets (testing, oversight, regularization) to keep learning on track.

Performance, Adaptation Speed, and Sample Efficiency

A key question for continual RL is: does the ability to update online actually translate into better performance or faster adaptation in practice? Recent benchmarks indicate **yes**, when done correctly. We have already noted how online RL improved success rates and accuracy in various domains. To highlight a few trends:

- **Success Rates and Skill Retention:** Continual learners tend to outperform static policies in dynamic tests. In robotics, the iRe-VLA agent not only achieved higher success on its original tasks after online fine-tuning, but crucially **did not lose performance on those tasks** while gaining new skills [70]. Traditional fine-tuning might have led to trade-offs, but their two-stage approach ensured stability (the success rate on original picking tasks stayed constant while new object success rose from 35% to 80% [33]). In the LLM domain, online fine-tuned models (with RL) showed clear gains on evaluation sets constructed to measure adaptability – e.g. Meta's online-tuned 70B model was significantly better at following WildChat instructions and solving math problems than the same model fine-tuned once offline [71] [72]. These improvements are often on the order of **5-20% absolute** (depending on the metric) which is substantial in benchmarks where progress is usually incremental.

- **Sample Efficiency:** A well-designed continual learner can be *more sample-efficient* than training from scratch on a new task. For instance, RoboCat's ability to learn a new manipulation with 100 demonstrations is a testament to the power of leveraging prior knowledge – the model's rich multi-task pretraining meant each additional demo provided much more marginal utility than it would to a task-specific model [22]. Likewise, meta-RL approaches have demonstrated that an agent can adapt to *new* tasks with **fewer trials** than a non-meta-trained agent would need. Nagabandi *et al.* showed robots adapting to novel disturbances in a handful of timesteps by using meta-learning [55]. In recommendation, contextual bandits can often reach near-optimal recommendations after only a few user interactions, thanks to their analytical update rules, whereas a full deep model might need to see many samples to slightly adjust its weights. The **trade-off between sample efficiency and stability** is nuanced: off-policy replay improves sample usage but can reduce stability, on-policy sacrifices some data but is more stable – hybrid approaches try to get the best of both. The Meta LLM study interestingly found *semi-online DPO* (which reuses some generated data multiple times) was as effective as online, implying that reusing data (improving sample efficiency) did not hurt final performance [9] [10].

- **Adaptation Speed:** Continual RL systems aim to adapt *during inference* or with minimal retraining, which is crucial for real-time applications. In that vein, meta-learning stands out: a meta-trained policy can adapt essentially in one episode (no gradient step, just internal state update) to a novel situation. For example, a meta-RL agent in a non-stationary bandit can detect the reward distribution shift and adjust its arm selection strategy quickly, whereas a standard RL policy would require additional training steps to re-learn the new distribution. In practical systems like personalized recommender systems, the ability to update *user by user* (each user's preferences shift and the policy

adapts immediately for that user) is a form of meta-learning (treating each user as a task). The **latency** of updates also matters: the YouTube bandit operates in real-time, updating parameters within milliseconds after each user interaction [47]. This low latency ensures the system is always using the latest information (e.g. if a user suddenly starts liking a new genre of videos, the recommendations adjust almost immediately). In contrast, a system that retrains a deep model nightly has a latency of many hours – during which it's essentially not adapting. Thus, continual RL can dramatically reduce adaptation lag. However, one must ensure that quick updates are reliable – techniques like smoothing updates or using ensemble predictions while an update is in progress can help avoid jitter.

- **Benchmarks and Competitions:** The progress is also reflected in benchmarks specifically designed for continual learning. There are emerging **continual RL benchmarks** (like the Continual World, CRL Suite, or the Continual Habitat Lab) where agents are evaluated on sequences of tasks or on environments that change over time. State-of-the-art systems show that those using *replay, meta-learning, or adaptive modules* significantly outperform naive approaches that train from scratch on each change. For example, a 2022 benchmark on sequential manipulation tasks demonstrated that agents with **experience replay + elastic weight regularization** retained nearly 100% of prior task performance while accumulating new skills, whereas a baseline PPO fine-tuned agent dropped to below 50% on earliest tasks after learning later ones [40]. In the *Lifelong RL* competition tracks, algorithms like **CLEAR (Continuous Learning of Experience after Reward)** and improved versions of **A3C with episodic memory** have scored highest, showing minimal forgetting. Moreover, some continual learning metrics such as *Forward Transfer* (how learning one task helps with future tasks) and *Backward Transfer* (how adding new tasks impacts old tasks) are now reported. The systems we discussed achieve **positive forward transfer** (e.g. RoboCat's broad training made new tasks easier) and **low negative backward transfer** (e.g. iRe-VLA's performance on original tasks stayed high).

In summary, when designed well, online RL systems demonstrate **robust gains in success rates and adaptability** over static policies, often with impressively few additional samples. This comes at the cost of extra complexity in the training loop and careful tuning of hyperparameters (learning rates, update frequency, etc.). There are still cases where things can go wrong – for instance, if the environment changes too rapidly or the reward is misleading, an online learner might chase noise. But across many benchmarks in the last 2–3 years, the best results in non-stationary scenarios are consistently achieved by those algorithms that *keep learning* instead of sticking to a fixed policy.

Trade-Offs and Challenges in Continual RL

Despite the successes, deploying continual reinforcement learning in the real world involves navigating key trade-offs:

- **Latency vs. Performance:** Integrating learning into an inference-time system can introduce latency. Training updates are computationally intensive; doing this concurrently with serving users or controlling a robot in real time is non-trivial. Systems like the YouTube bandit address this by using very lightweight models for online updates [48], while heavy lifting is done offline. Similarly, in robotics, one might run online RL on an onboard computer with limited resources – if each update step takes too long, the robot may not respond in time to changes. There's a balance between how frequently/quickly you update and the benefit you get. Meta's semi-online LLM training suggests you don't always need to update every single interaction; syncing periodically (say every N steps)

achieves nearly the same performance [9]. This can dramatically cut down computation and latency. Another tactic is asynchronous training: e.g. in AlphaStar, the policy network was continuously trained on a server farm while a copy was used to play games – periodically the new weights were swapped in. This decouples latency from learning somewhat, but introduces its own complexity (you must ensure the new policy is safe to deploy when swapped). In summary, designers must decide how *fast* the loop should close. For many applications, a slightly slower learning loop (minutes or hours) that yields stable improvements may be preferable to a millisecond loop that might be noisy.

- **Stability vs. Plasticity:** This is the classic stability-plasticity dilemma. Highly plastic systems (learn very fast) risk instabilities and forgetting; highly stable systems (learn very cautiously) might not adapt sufficiently. Continual RL must strike a balance. Techniques like those discussed (trust-region, regularization) skew towards stability – preventing large changes – at the cost of some plasticity. On the other hand, algorithms that were very plastic (like naive Q-learning on a non-stationary task) can chase moving targets and oscillate wildly. The **ideal** is an adaptive algorithm that can detect when the environment has changed significantly and **increase plasticity** at that moment (making larger updates), but otherwise remain stable. Some meta-learning research tries to achieve this, e.g. by having a learned update rule that adjusts its own learning rate based on novelty detection. Practically, many systems use a decay schedule or trigger conditions – e.g. if reward drops below a threshold, allow a bigger update (plastic phase) to recover, then go back to conservative updates. The **loss of plasticity** over time is also an observed issue – a policy might become stuck in its ways (a form of convergence) and struggle to explore new strategies. Approaches like the quadruped's **reinitialization of unused parameters** [36] is explicitly to *inject plasticity* back into the network for each new task. This is an active research area: how to maintain an agent that is a seasoned expert in many things and yet remains a curious learner.

- **Exploration vs. Safety:** Continual learning implies continuous exploration – but exploring in a live system can lead to errors or unsafe actions. In reinforcement learning, exploration is usually achieved via randomness (epsilon-greedy, entropy bonuses, etc.) or deliberate strategies (curiosity-driven goals). In a contained training environment, an agent can afford to try silly things; in a deployed system, those silly things could be catastrophic (or at least user-alienating). The trade-off here often requires **outside input or constraints**: as noted, one might limit exploration to certain bounds or have human oversight. The YouTube example shows even an algorithmic trade-off: they needed to *balance exploration with user satisfaction*, leading to methods that gradually test new content rather than flood the user with random picks [47]. Another angle is **multi-objective optimization**: treating safety as another reward (or constraint). The agent then doesn't just maximize task reward, but is penalized for unsafe behavior. However, setting those penalties correctly is hard; if too harsh, the agent might stick to overly conservative actions and not learn quickly. If too lenient, it might still occasionally cross the line. In physical systems, simulation is often used as a compromise: let the agent explore freely in sim (where it can't do damage), and only deploy when it has a good policy. Continual sim-to-real transfer can then happen (the agent keeps improving in simulation with new scenarios, and those updates get transferred to the real robot periodically). This reduces direct real-world exploration needs, though it introduces the sim-vs-real gap issues. All these reflect the fundamental challenge: an agent that keeps learning is inherently less predictable than a fixed agent, so we must shape its learning behavior to avoid dangerous exploration.

- **Local vs. Global Optima (Exploration vs. Exploitation over long term):** A related issue is that an online RL agent might get stuck in a **local optimum policy** that is "good enough" and, because it is continually reinforcing that behavior, it never explores other, potentially better strategies. This can be seen in self-play systems too – e.g. an agent might find a strategy that beats its current set of opponents and then stop improving, because it doesn't encounter opponents that force it out of that strategy. In a non-stationary world, there's also the phenomenon of **cycling**: agent chases moving target, overshoots, then relearns something akin to its old behavior when circumstances change back – effectively oscillating without settling. To counter this, techniques like **population-based training** (maintaining multiple policy instances, some exploring more, some exploiting) are used. AlphaStar's league training is an example where a population of agents with different strategies ensured continued diversity and improvement. For a single agent, occasionally injecting *directed exploration* (like resetting it to an earlier policy and exploring again, or using intrinsic motivation bonuses) might be necessary to escape plateaus. This ties back to the stability-plasticity dilemma: too stable and it'll stick with suboptimal behavior, too plastic and it might destabilize trying to explore.

- **Evaluation and Reproducibility:** With continual learning, the model is a moving target – this complicates evaluation. If your system is updating daily, which version do you evaluate on the benchmark? How do you ensure a result is reproducible if the training is endless or depends on a specific sequence of experiences? Researchers have started to emphasize **evaluation protocols** for continual learning (like measuring performance after each task, or average performance over time). In deployed systems, one might log the performance metrics continuously (like reward or user engagement) and use statistical tests to see if the trend is upward or if a new update caused a significant drop. Detecting issues might require rollbacks, as mentioned. From a scientific perspective, continual RL sometimes blurs the line between training and testing – especially in open-ended setups like XLand where learning *is* the product. One evaluates such systems by emergent capabilities rather than a fixed test set. This is exciting (it's a path toward more general AI), but also challenging because it's harder to compare algorithms when each might generate its own curriculum. Thus, creating *standardized continual learning benchmarks* (like sequential task lists, or known environment shifts at certain times) is an ongoing effort to fairly compare methods.

- **Ethical and Control Considerations:** When an AI system can change itself after deployment, it raises questions of control and alignment. A fixed model can be audited and its behavior at least characterized for known inputs. A continually learning model might, after some time, become quite different from its initial version (especially if the reward signal has any gaps that allow for distribution drift). This is particularly sensitive in LLMs or content recommenders – you wouldn't want a chatbot to gradually learn harmful behavior because a subset of users gave it bad feedback, for instance. OpenAI and others have thus far refrained from true online learning in their deployed chatbots, opting instead for periodic retraining with heavily filtered data. The **trade-off here is agility vs oversight**: a fully online model adapts quickly but could go out of bounds just as quickly; a slower, manual retraining process ensures humans can vet changes but at the cost of responsiveness. One interesting middle ground is **incremental learning under human review**: e.g. an LLM could log new conversations, a smaller model or script could propose updates based on them, but a human (or committee) reviews those updates before they are applied to the live model. This introduces latency but retains some agility and adds a safety net.

In conclusion, continual reinforcement learning systems provide a path toward AI that is **adaptive, resilient, and long-lived**, learning from each interaction. They have proven effective in the last few years across diverse fields – from **LLMs learning to better follow instructions on the fly** [71] **, to robots rapidly fine-tuning to new tasks** [22] **, to recommendation engines staying up-to-date with user tastes** [48] . The best implementations blend *multiple approaches* – e.g. meta-learning for quick adaptation, partial updates for stability, and human oversight for safety – to navigate the intricate trade-offs involved. Continual RL is still a developing area, and challenges like **catastrophic forgetting, safe exploration, and scalable evaluation** remain active research topics. Nonetheless, the progress in the last 2–3 years suggests that *online learning can be deployed usefully and safely* when guided by the right algorithms and engineering practices. As AI systems become more embedded in everyday life, the ability to **learn and improve continually** will be invaluable – allowing systems to personalize, to handle evolving scenarios, and to correct their mistakes in real time. The work reviewed here lays the foundation for such lifelong learning agents, and we can expect future systems to push these boundaries even further, achieving greater autonomy and robustness through continual reinforcement learning [73] [74] .

---

[1] [2] [34] [35] [36] [37] [38] [39] [40] Continual Reinforcement Learning for Quadruped Robot Locomotion
https://www.mdpi.com/1099-4300/26/1/93

[3] Advancements and Challenges in Continual Reinforcement Learning: A Comprehensive Review
https://arxiv.org/html/2506.21899

[4] [5] [6] [7] [8] [9] [10] [15] [16] [17] [18] [50] [51] [52] [62] [67] [68] [71] [72] Bridging Offline and Online Reinforcement Learning for LLMs
https://arxiv.org/html/2506.21495

[11] [12] [13] [14] [66] Paper page - Training Language Models to Self-Correct via Reinforcement Learning
https://huggingface.co/papers/2409.12917

[19] [20] [21] [22] [23] [24] RoboCat: A self-improving robotic agent - Google DeepMind
https://deepmind.google/discover/blog/robocat-a-self-improving-robotic-agent/

[25] [26] [27] [28] [29] [30] [31] [32] [33] [60] [70] Improving Vision-Language-Action Model with Online Reinforcement Learning
https://arxiv.org/html/2501.16664v1

[41] [42] [43] [44] [45] Generally capable agents emerge from open-ended play - Google DeepMind
https://deepmind.google/discover/blog/generally-capable-agents-emerge-from-open-ended-play/

[46] DeepMind: Generally capable agents emerge from open-ended play
https://www.lesswrong.com/posts/mTGrrX8SZJ2tQDuqz/deepmind-generally-capable-agents-emerge-from-open-ended

[47] [48] [49] [61] [63] [2307.15893] Online Matching: A Real-time Bandit System for Large-scale Recommendations
https://arxiv.org/abs/2307.15893

[53] [54] [55] [56] [57] [58] [59] Meta-Reinforcement Learning with Discrete World Models for Adaptive Load Balancing
https://arxiv.org/html/2503.08872v1

[64] Online Iterative Reinforcement Learning from Human Feedback with ...
https://neurips.cc/virtual/2024/poster/95002

65 Decision Making for Human-in-the-loop Robotic Agents via ... - arXiv
https://arxiv.org/abs/2303.06710

69 73 74 ICML Poster Continual Reinforcement Learning by Planning with Online World Models
https://icml.cc/virtual/2025/poster/44151