Anton A. Nesterov · an@vski.sh · July 17, 2026

The Sequential Recommender: Fitted Q-Iteration

Machine Learning · Reinforcement Learning · Bandits

The moment an action changes what the next decision can see or do, the bandit is myopic and the problem becomes sequential reinforcement learning.

doi: 10.5281/zenodo.21418101

Assets & Materials

Gold Digger — Interactive simulation. A linear Fitted Q-Iteration agent learns to collect gold on a grid, in real time, in your browser.	https://blog.vski.ai/gold-digger/
The Sequential Recommender — Research Notebook. Trains the linear-FQI agent to 1.79-2.62x over random, in your browser.	https://blog.vski.ai/notebooks/notebooks/index.html?path=sequential_recommender.ipynb

What happens when the decision you make this round changes what you can decide next round? A contextual bandit pretends it doesn't; the agent here can't afford that pretence, because its actions move it through space. That forces a step up from the bandit family into the rest of reinforcement learning.

> *Background in Contextual Bandits and Their Ethical Use Cases and Building the Bandit Recommender.*

When bandits stop being enough

A stateless recommender — the kind built on a contextual bandit — frames the problem in four load-bearing words: **learn what to show next, online, from implicit feedback alone, while you are still being judged on every choice you make.** Keep all four constraints and you land on a contextual bandit.

Strip one more assumption and the framework breaks. The contextual bandit assumes the action you take **has no consequence for the next decision.** Showing the user a minimalist landscape this round changes your estimate $\hat{U}$ of their taste, but it does *not* change which items are available next round, or what you know about them, or where the user is in their session. The state of the world, from the agent's point of view, is a single context vector handed to it fresh each round by some upstream process. That is the definition of a contextual bandit: **stateless**.

Real systems routinely violate that assumption. The clearest examples:

Business problem	What the action changes	Why a bandit fails
Session-based recommendation — queuing the next 10 songs / videos / product cards	The user's fatigue, satiation, exposure to themes	The reward of showing item n is whether the user stays *for* item $n{+}1$; a bandit assigns that reward to $n{+}1$ instead
Last-mile delivery — dispatching a courier across a city	The courier's location, the time budget, what's been learned about traffic	The value of going West is the *future stops it makes cheap*, not the immediate drop-off
Warehouse pick-path — ordering a batch of picks	The picker's location, what's left in the batch	Going to aisle 7 now makes every other aisle-7 pick in the batch cheaper — that credit has to flow backward to the aisle-7 decision
Robotics / autonomous-vehicle coverage — moving a sensor platform	The robot's pose, what's been observed, the battery budget	The reward of a motion is the *new cells it exposes to the sensor*, not anything at the destination cell

A useful filter: **if you find yourself saying "we'll just A/B test it," you are treating each decision as independent — that is the bandit regime. If you find yourself saying "but the order matters" or "we won't see the result for three decisions" — read on.** The machinery steps up from a contextual bandit to full sequential reinforcement learning.

The Gold Digger demo is the cleanest embodiment of this regime. An agent lives on a 30×30 grid, senses gold and stones only in a small radius, and has to *move* to collect. Every leap both changes its position and changes what it can see. The state is not handed to the agent — the agent *builds it*, one noisy tick at a time. The grid-collection abstraction will run throughout — it makes every sequential-RL concept visible without burying it in domain noise.

The MDP escalation

The contextual bandit is the **non-associative**, single-state case (Sutton & Barto, §2.9). The sequential problem is a **Markov Decision Process** (Sutton & Barto Ch. 3): the agent observes a state s_t, picks an action a_t, receives a reward r_t, and lands in a next state s_{t+1} that *depends on* a_t. The objective is no longer per-round regret but the **cumulative discounted return** over the episode:

$G_t \;=\; \sum_{k=0}^{T} \gamma^k \, r_{t+k+1}, \qquad \gamma \in [0, 1)$

The discount γ is the knob that says how much the agent cares about future reward relative to immediate reward. $\gamma = 0$ collapses the problem back to a bandit (only r_{t+1} matters). $\gamma \to 1$ makes the agent care about the whole episode equally. The Gold Digger uses $\gamma = 0.95$.

The escalation in one table

	Contextual bandit	**Sequential recommender**
Decision frame	Stateless: each round independent	Sequential: each action changes the next state
Objective	Per-round regret $\sum_t(\mu^* - \mu_{a_t})$	Discounted return $\sum_t \gamma^t r_t$
What's learned	$\hat{U}$ — taste vector	$Q(s, a)$ — long-horizon action-value
Target	Observed reward r_t	Bootstrapped $r_t + \gamma \max_{a'} Q(s', a')$
State representation	Handed to the agent as context x_t	**Built by the agent** from noisy sensors
Update	Online ridge, one row per round	Batch Fitted Q-Iteration on a replay buffer
Sutton & Barto home	§2.9 (associative search)	Ch. 3 (finite MDPs), Ch. 6 (Q-learning)

The Q-function is the upgrade

The mechanism that lets a sequential agent plan is the **action-value function**:

$Q(s, a) \;=\; \mathbb{E}[G_t \mid s_t = s, a_t = a, \pi]$

— the expected discounted return of taking action a in state s and acting under policy π thereafter. The *and acting under π thereafter* clause is the entire difference from a bandit. Q folds the future into the present: the value of taking a length-8 East leap is not the gold it grabs but the gold it grabs **plus** the value of the state it lands in, recursively, all the way to the episode's end. A bandit has no equivalent.

The optimal Q-function satisfies the **Bellman optimality equation**:
$Q^*(s,a) \;=\; \mathbb{E}\big[r + \gamma \max_{a'} Q^*(s',a') \,\big|\, s,a\big]$
This recursive equation is what every RL algorithm is trying to solve. The next two sections are about *how* — and why the obvious approach diverges.

Why standard Q-learning fails

The obvious way to learn Q is **online temporal-difference (TD) learning**: at each transition, nudge $Q(s,a)$ toward the bootstrapped target $r + \gamma \max_{a'} Q(s',a')$. With a small enough step size and a lookup-table Q, this converges. The disease appears when you replace the lookup table with a **non-linear approximator** (an MLP, a GRU). Sutton & Barto (§11.3) name the **deadly triad** — three properties that, combined, can make Q diverge to astronomical values in a few hundred steps:

1. **Function approximation** — Q is a regressor, not a table.
2. **Bootstrapping** — the update target contains another Q-estimate.
3. **Off-policy data** — the buffer was generated by an exploratory policy but you are evaluating the optimal one.

> *Any two of the three are fine. With all three, divergence is a structural property of the algorithm — not a tuning problem.*

The first attempt at the Gold Digger agent walked straight into it. A GRU with a Q-head, trained by online TD(λ) on the live tick stream. Within a few thousand ticks the Q-values had blown up to $\sim 10^{51}$, and — the part that actually matters — **evaluation got *worse* with training**. Target networks did not help. δ-clipping did not help. Gradient clipping did not help. λ tuning did not help. They couldn't help, because the disease was the architecture, not the hyperparameters.
The research notebook reproduces this on a notebook-scale 8×8 grid with a 6→48→1 ReLU MLP, in two regimes:

- **Aggressive LR, no stability hygiene:** $|Q|$ grows 1.27 → 2.25e+1 → 1.32e+2 → **4.26e+139** → inf over 41 episodes. Textbook divergence.
- **Conservative LR + target network + δ-clip + grad-clip:** $|Q|$ stays pinned at its initial value; the policy never improves over random. Stable and inert.

Both regimes fail, for different but principled reasons. Adding more bells and whistles (experience replay, double-Q, priority weighting) stretches the curves but does not change their shape; the disease is in the algorithm class, not the hyperparameters.

Fitted Q-Iteration

The fix is older than deep learning. **Fitted Q-Iteration** (Ernst, Geurts & Wehenkel, JMLR 2005) replaces the moving-target online update with a **frozen-target batch regression**:

$$\text{Loop } k = 0, 1, 2, \ldots \begin{cases} \text{1. Collect transitions } (s, a, r, s') \text{ via } \varepsilon\text{-greedy rollouts.} \\ \text{2. Build targets with the FROZEN } Q_{k-1}: \quad y_i = r_i + \gamma \max \\ \text{3. Refit } Q_k \text{ by regression: } \min_w \sum_i \big(Q_k(s_i, a_i; w) - y_i\big)^2. \end{cases}$$

The target y_i is computed **once** with the *old* Q, then held **fixed** while the regression solves for the *new* Q. Each iteration is now an ordinary supervised regression with a stationary target. Munos & Szepesvári (JMLR 2008) prove the convergence: total error decomposes into approximation error + estimation error + a term that decays geometrically in γ (the Bellman optimality operator is a γ-contraction). FQI is **provably stable** in a way online TD simply is not.

What FQI does *not* fix

FQI removes the *divergence* of the deadly triad by freezing the bootstrap target. It does **not** guarantee a *useful policy*. With a non-linear Q (an MLP, a tree ensemble), the regression has many local minima, and `argmax Q` can degenerate to a single action regardless of context — what the Gold Digger prototype history calls **action-space collapse**. The first FQI attempt on this agent (MLP Q, frozen target, greedy evaluation) reliably collapsed to "always pick length-8" — 379 of 400 actions in one evaluation run, 358 of them wall bumps.

The research notebook reproduces this too: the same FQI loop as the deployed agent, but with an MLP Q instead of linear ridge. The world is uniform — no length is *a priori* better. The action histogram oscillates and then degenerates within eight FQI rounds. The Q-function is *stable* (frozen-target FQI guarantees that), but the policy is *uninformative*.

The deployed agent does two things differently:

1. **Replaces the MLP with linear ridge regression.** A linear fit has a unique closed-form solution for each target set; there are no local minima, no oscillation between regimes, no degenerate argmax.
2. **Adds action masking.** Candidates the belief map says are blocked are removed from the argmax entirely, not penalised through reward shaping. The policy never has to *learn* not to ram walls.

Both fixes come from the same principle: **push the hard parts out of the learning algorithm and into either the convex model class or the constraint structure.**

Why linear — the closed-form, non-divergent fit

After everything that went wrong with the GRU and the MLP, the deployed Gold Digger Q-function is **twelve numbers and a dot product**:

$Q(s, a) \;=\; w \cdot \phi(s, a) + b$

where $\phi(s, a)$ is an 11-dimensional feature row describing one candidate leap — belief-that-destination-has-gold, path-clear flag, gold density on path, density of unknown cells ahead, raycast densities, inverse length. The fit is one Cholesky solve of a 12×12 system (~50 microseconds). Two reasons this is the *deliberately correct* choice, not a compromise.

Provable stability. A linear Q fit by ridge regression is the exact minimiser of a convex least-squares problem with a frozen target. There is no approximation error from gradient descent, no local minima, no learning-rate pathology. Tsitsiklis & Van Roy (1997) proved on-policy linear TD converges almost surely; the linear-FQI result (Munos & Szepesvári 2008, and the older Least-Squares Policy Iteration of Lagoudakis & Parr 2003) extends this to the off-policy batch case. Linear FQI is one of the few value-function settings where the deadly triad genuinely does not bite.

The features already carry the non-linearity. The hard part of this problem is not the shape of the value function — it is *representing the state* well enough that the value function is simple. The agent's state is a partially-observed 30×30 grid; a neural net *could* in principle learn to read that grid directly, but that asks it to rediscover, from reward signal alone, the entire front-end of perception. We hand it a belief map maintained by a classical Bayesian perceiver instead, and the candidate features read directly off that map. By the time the linear regressor sees the feature row, the problem has been reduced to *learn that high* `goldDensityOnPath` *is good and high* `stoneDensityOnPath` *is bad.* That is linear. Adding parameters to model it would be adding parameters to fit noise.

This is the same lesson the stateless bandit regime arrives at from the other direction. LinUCB is a linear regressor wearing an exploration hat; it works because the hashing trick turns tags into a feature space where reward is linear in the weights. The sequential recommender is a linear regressor wearing a planning hat; it works because the Bayesian belief map turns a messy POMDP into a Markovian feature space where Q is linear in the weights.

> ***In both regimes the non-trivial work is in the feature engineering, not the model class.*** *Reaching for a neural network is what you do when you have raw unstructured input (pixels, tokens, audio) and no good way to hand-engineer features. The moment you do have a good feature representation, the linear model is not a compromise — it is the right tool.*

What if the grid is infinite?

The grid has been a *fixed*, bounded $N \times N$ square throughout. That is the right starting point — it makes the MDP finite, the regret bounds concrete, the demo tractable. But it is worth asking the obvious next question: **what changes if the grid is infinite — true exploration over an unbounded state space?**

The bounded grid is a finite MDP — every regret bound scales with the number of states $|S|$, and $|S|$ is small. An unbounded grid is formally either a **continual / lifelong RL** problem (the agent keeps learning forever over an unbounded stream of new states; Khetarpal et al., JAIR 2022) or a **metric-state RL** problem (the state space is a continuous or countably infinite metric space). In both regimes the regret bounds that depend on $|S|$ become vacuous.

What replaces them is a **smoothness / covering-number** argument:

Bounded finite grid	Unbounded / infinite grid
Regret scales with $\|S\|$	Regret scales with the **covering number** of the state-action space, weighted by a Lipschitz constant
PAC-MDP (Kearns & Singh)	Metric-E^3 (Kakade, Kearns, Langford 2003); adaptive discretization (Sinclair et al. 2019); ν-smooth MDPs (Maran et al. 2024)
Optimism bonus retires when an arm is well-pulled	Optimism bonus retires when a **neighbourhood** is well-covered

Three things survive the move to an unbounded grid:

1. **Linear models stay safe.** Linear value-function approximation still has the Tsitsiklis-Van-Roy / Munos-Szepesvári guarantees; the state space can be unbounded as long as the features live in a fixed finite-dimensional space. This is precisely why the Gold Digger features — belief densities, path densities, raycasts — are *normalised*: they don't grow with grid size.
2. **Fitted Q-Iteration stays stable.** The frozen-target argument is independent of $|S|$; what matters is the per-iteration regression being a convex problem, which it remains for linear Q.
3. **The belief map generalises naturally.** Maintaining beliefs over an unbounded space is just maintaining beliefs over a hash-addressed dictionary of visited cells instead of a fixed array. The perception update is unchanged.

The hardest change is the loss of a clean episode boundary. On a fixed grid the episode ends when the agent runs out of gold or steps. On an infinite grid there is no end — the agent keeps encountering new states, and *the distribution of those states shifts* as the agent's policy improves. The ridge-regression fit needs a forget-

ting factor or a sliding window, or it will be dominated by the early random-exploration data forever. That is one of the open directions for the engine.

> *The linear-FQI + belief-map architecture is surprisingly close to working in the infinite-grid regime — the convexity, the frozen target, and the Bayesian perceiver all generalise. The work that remains is operational (sliding-window replay, hash-addressed beliefs, online-boundary policy), not algorithmic. That is a remarkable property for a 12-weight agent to have.*

Business cases

The actual commercial applications are wherever **a sequence of decisions, each changing what the next one can see or do, has to be made under uncertainty and without labels.** Three families, each with verified production deployments.

Session-based recommendation

The most direct commercial heir to the stateless bandit, and the one with the deepest production literature. Showing one item well is the bandit problem; showing a *sequence* whose joint payoff over a session is what you actually get paid for is the RL problem. Verified deployments:

- **YouTube / Google** — Chen et al. (KDD 2019), *Top-K Off-Policy Correction for a REINFORCE Recommender System* — production top-K RL recommender at YouTube scale, with a sceptical off-policy correction to keep learning stable on logged data.
- **Spotify** — McInerney et al. (2023), *Optimizing Audio Recommendations for the Long-Term* — production RL over podcast listening journeys, reported with online A/B test results.
- **Tencent** — *IntegratedRL-MTF* (2023+), an offline RL algorithm deployed in several large-scale production recommenders at Tencent since July 2023.

In all three the action changes the state of the user (fatigue, satiation, exposure) and that state change has to be folded back into the value of the action — exactly the regime where bandits are myopic and sequential RL earns its keep. The reason most teams still ship bandits here is the deadly triad — full sequential RL has historically been too unstable to ship, so the pragmatic move was to accept myopia in exchange for a system that doesn't diverge in production. Linear FQI over an engineered state representation is the cheap escape from that compromise.

Operations and logistics — sequential routing under uncertainty

An agent crossing an uncertain map, collecting reward where it finds it, learning where reward tends to be from noisy local observations. That is the literal problem statement of a remarkable amount of operations work:

- **Last-mile delivery and courier dispatch**, where the "gold" is successful drop-offs, the "stones" are traffic and access constraints that only become visible en route, and the agent has to commit to a route before it knows which stops will fail.
- **Warehouse restocking and pick-path optimisation**, where each pick both takes time and changes which other picks become cheap (the picker is already in aisle 7, so the aisle-7 picks just got cheaper for the rest of the batch).
- **Field-service and technician dispatch** — telecom repair, home-health visits, appliance service — where each job's true duration is only known once a technician arrives, and the day's route has to be re-planned online as uncertainty resolves.

In all three the bandit framing fails for the same reason it failed in the demo: the value of an action is dominated by the *future actions it makes cheap*, not by its immediate reward. Linear FQI plus a belief-map state — demand-forecast belief per zone instead of bGold per cell, route-clearance belief instead of bStone — is a remarkably close fit.

> ***Verified-production caveat.*** *Published, peer-reviewed production deployments of full sequential RL in last-mile delivery are still scarce. The literature is rich in algorithmic work (Deep RL for stochastic VRP, dynamic VRP under stochastic demand) but the deployments documented in the public literature lean more on mixed-integer programming and ML-augmented heuristics than on learned value functions. Treat this family as emerging, not mature.*

Robotics

Anytime an autonomous agent has to **move through physical space it only partially observes**, the Gold Digger architecture applies almost verbatim:

- **Coverage and exploration** — survey drones, inspection rovers in industrial plants, subsea mapping, planetary exploration. The "collect gold" reward becomes "successfully imaged a cell of the asset"; the "stones" become no-go zones; the belief map is literally what the robot believes about its surroundings.
- **Search-and-rescue and response** — the same structure with a human-life payoff: explore likely areas first, route around hazards, balance coverage against time pressure.

- **Warehouse and factory floor AMRs** — autonomous mobile robots shifting inventory. Verified at production scale by Amazon Robotics' Multi-agent RL for Robotic Sortation Centres.

The belief-map-plus-linear-FQI design ships well here for a reason that doesn't apply to most ML: **robotics is the one domain where the deadly triad is not an academic concern but a physical safety one.** A divergent Q on a recommender shows a user a bad item; a divergent Q on a robot drives it into a wall. The closed-form, provably-non-divergent fit is not a performance optimisation — it is the only responsible default for any learned policy that is allowed to move hardware.

Conclusion

The bandit family — linear, GLM, neural, adversarial — comes with a standing recommendation to start simple and escalate only when the regret of the simpler tool is the bottleneck. The same advice applies one level up. The hierarchy of sequential decision problems is roughly:

Regime	Tool	When it applies
Stateless, immediate reward	Contextual bandit (LinUCB, Thompson)	Most production "recommenders." See the bandit recommender.
Sequential, but stationary and short-horizon	Tabular Q-learning / SARSA	Small state space you can enumerate.
Sequential, long-horizon, structured features	**Linear FQI**	Belief maps, session state, engineered features. The regime covered here.
Sequential, long-horizon, raw input	Deep RL (DQN, SAC, PPO)	Pixels-to-actions. The deadly triad has to be managed, not avoided.
Sequential, adversarial	Differentiable game-theoretic RL	Fraud, auctions, security. Out of scope.

The sequential recommender sits in the third row. It is the simplest tool that honestly handles "my action changes the next state" — simpler than deep RL by orders of magnitude in both compute and operational risk, and the only one of the four whose training step is provably convex. The question to ask, before reaching past it, is whether your state really is pixels. If it isn't — if you can write down a feature representation of the state that a domain expert would agree captures the relevant information — then linear FQI is not a compromise. It is the right answer, and a neural net would be over-engineering.

The demo, in the end, is a recommender. It just happens to recommend *leaps* instead of *items*, and to learn its objective one noisy tick at a time. The fisher is still on the lake. The fish have just learned to swim.

References

1. **Sutton, R. S. & Barto, A. G.** *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018 — §2.9 (associative search / contextual bandits), Ch. 3 (finite MDPs), §11.3 (the deadly triad). Full PDF.
2. **Ernst, D., Geurts, P. & Wehenkel, L.** *Tree-Based Batch Mode Reinforcement Learning*. (JMLR 2005) — the original Fitted Q-Iteration paper, and the algorithm the Gold Digger agent runs.
3. **Munos, R. & Szepesvári, C.** *Finite-Time Bounds for Fitted Value Iteration*. (JMLR 2008) — the convergence proof that explains why FQI is stable: the Bellman optimality operator is a γ-contraction.
4. **Tsitsiklis, J. & Van Roy, B.** *An Analysis of Temporal-Difference Learning with Function Approximation*. (IEEE TAC 1997) — the formal result that on-policy linear TD converges almost surely. The theoretical anchor for "linear FQI is safe."
5. **Lagoudakis, M. G. & Parr, R.** *Least-Squares Policy Iteration*. (JMLR 2003) — the linear/convex cousin of FQI; the closed-form solve that replaces the moving-target update.
6. **Liu, Y. & Swaminathan, A.** *Provably Good Batch Off-Policy Reinforcement Learning Without the Deadly Triad*. (NeurIPS 2020) — the modern framing of why batch (FQI-family) methods avoid the triad.
7. **Chen, M. et al.** *Top-K Off-Policy Correction for a REINFORCE Recommender System*. (KDD 2019) — production YouTube RL recommender.
8. **McInerney, J. et al.** *Optimizing Audio Recommendations for the Long-Term*. (2023) — production Spotify RL for podcast listening journeys.
9. **Khetarpal, K. et al.** *Towards Continual Reinforcement Learning: A Review and Perspectives*. (JAIR 2022) — the formal framing of the infinite-horizon / lifelong regime.
10. **Maran, D. et al.** *No-Regret Reinforcement Learning in Smooth MDPs*. (PMLR 2024) — the covering-number / Lipschitz regret bounds that replace $|S|$-scaling in the unbounded regime.
11. **Related work on the same engine.** *Contextual Bandits and Their Ethical Use Cases* (the bandit math) and *Building the Bandit Recommender* (the stateless recommender this one escalates from).
12. **Executable and live companions.** Research notebook (runs every claim in the browser) and the Gold Digger demo (the live agent).