# Lightweight Reinforcement Learning for Cryptocurrency Token Search

**Edgar Pavlovsky**
Dark Research
edgar@darkresearch.ai

August 12, 2025

## Abstract

We present a novel distributed reinforcement learning architecture for cryptocurrency token search deployed in production. Our approach addresses the unique challenges of crypto search where token popularity changes rapidly, duplicate tokens exist, and user preferences vary significantly. By employing client-side Upper Confidence Bound (UCB1) strategy selection with distributed genetic algorithm evolution, we achieve real-time learning with minimal computational overhead. The system operates entirely client-side with comprehensive fallback mechanisms, enabling sub-100ms ranking decisions while continuously adapting to user feedback through evolution triggered during navigation events. Production results demonstrate evolved strategies achieving up to 95% average reward, significantly outperforming baseline approaches. The implementation processes 6-dimensional feature vectors encompassing market metrics and verification status, with genetic evolution producing superior strategies through real user interactions. This work demonstrates the practical viability of distributed RL systems for real-time search optimization in volatile domains and establishes a new paradigm for client-side evolutionary algorithms in production environments.

## 1 Introduction

The explosive growth of cryptocurrency markets has created an information retrieval challenge where traditional search ranking methods fail to adapt to rapidly changing token dynamics. Unlike conventional web search where document relevance remains relatively stable, cryptocurrency token search presents unique characteristics that render traditional approaches inadequate:

**Rapid Market Dynamics:** Token popularity and market metrics can shift dramatically within hours during market events, while traditional ranking systems typically retrain on weekly or monthly cycles, creating significant lag in relevance adaptation.

**Data Sparsity:** With over 30,000 new tokens launched daily in the memecoin ecosystem, the majority of search targets have minimal historical data for traditional ML training, yet users must make time-sensitive decisions about these nascent assets.

**Duplicate Token Problem:** Numerous tokens share similar names, symbols, or branding (often intentionally), requiring disambiguation strategies that traditional keyword-based ranking cannot effectively handle.

**Aggregation Complexity:** Startups face significant challenges collecting and integrating diverse data sources (market data, social signals, on-chain metrics) required for traditional model training, while RL systems can learn directly from user interactions.

Traditional ranking approaches that optimize for long-term static relevance are inadequate for this domain. While recent work by Hu et al. [1] demonstrated the effectiveness of reinforcement learning for e-commerce search through Search Session Markov Decision Processes (SS-MDP), their approach requires complex state modeling and policy gradient methods unsuitable for lightweight client-side deployment. Inspired by both the RL survey of Zhao et al. [2] and the multi-step ranking insights from Alibaba's work, we implement a simplified multi-armed bandit (MAB) system that achieves similar adaptation benefits while maintaining practical deployment constraints.

Our contributions are:

- A novel distributed client-side evolution architecture where genetic algorithms run during user navigation, enabling real-time strategy improvement
- A lightweight UCB1-based ranking system that operates client-side with sub-100ms latency
- A production-grade A/B testing infrastructure with deterministic user assignment and comprehensive fallback mechanisms
- A 6-dimensional feature engineering approach combining market metrics with verification signals
- Empirical validation on a live cryptocurrency search platform showing evolved strategies achieving up to 95% average reward (on our 0–1 positional reward scale; see Reward Function)

## 2 Problem Formulation

### 2.1 Multi-Armed Bandit Framework

We formulate the token search ranking problem as a multi-armed bandit where each arm represents a different ranking strategy. Following the definition from [2]:

A $K$-armed bandit for token search is a tuple $\langle \mathcal{A}, \mathcal{R}, \pi \rangle$ where:

- $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ is the set of ranking strategies (arms)
- $\mathcal{R} : \mathcal{A} \to [0, 1]$ is the (unknown) expected reward for each arm
- $\pi$ is the bandit policy that selects arms based on historical feedback (e.g., UCB1)

Each ranking strategy $a_i$ is characterized by a weight vector $\mathbf{w}_i \in \mathbb{R}^6$ that defines the importance of six features:

$$\mathbf{w}_i = [w_{\text{wallets}}, w_{\text{trades}}, w_{\text{mcap}}, w_{\text{liquidity}}, w_{\text{volume}}, w_{\text{verified}}]$$

where $\sum_{j=1}^{6} w_{i,j} = 1$ and $w_{i,j} \geq 0$ for all $j$.

### 2.2 Feature Engineering

Our 6-dimensional feature space addresses correlation issues present in raw market data through careful engineering:

**Core Market Features:**

$$f_{\text{wallets}} = \log_{10}(\text{unique\_wallets\_24h} + 1) \tag{1}$$
$$f_{\text{trades}} = \log_{10}(\text{trade\_24h} + 1) \tag{2}$$
$$f_{\text{mcap}} = \log_{10}(\text{market\_cap} + 1) \tag{3}$$
$$f_{\text{liquidity}} = \log_{10}(\text{liquidity} + 1) \tag{4}$$
$$f_{\text{volume}} = \log_{10}(\text{volume\_24h\_usd} + 1) \tag{5}$$

**Verification Signal:**

$$f_{\text{verified}} = \begin{cases} 1.0 & \text{if token is verified} \\ 0.0 & \text{otherwise} \end{cases}$$

The logarithmic transformation normalizes the heavy-tailed distribution of market metrics while preserving relative ordering. This approach reduces the correlation between highly correlated raw features (e.g., volume and market cap often correlate at $r > 0.8$).

### 2.3 Reward Function

User selection position determines the reward signal:

$$r(p) = \begin{cases} 1.0 & \text{if } p = 0 \text{ (perfect hit)} \\ 0.7 & \text{if } p = 1 \\ 0.5 & \text{if } p = 2 \\ 0.3 & \text{if } p = 3 \\ 0.2 & \text{if } p = 4 \\ 0.15 & \text{if } p = 5 \\ 0.1 & \text{if } p \geq 6 \end{cases}$$

This reward structure provides strong signals for top positions while maintaining non-zero rewards for lower positions to enable learning from suboptimal selections.

**Relation to IR Metrics.** Our position-based reward $r(p)$ is a monotonically decreasing gain by rank, akin to assigning fixed gains in discounted cumulative gain (DCG). In the absence of explicit relevance labels, this choice encourages high precision at top ranks while still learning from lower positions. When judgment data is available, one can (a) calibrate $r(p)$ to approximate DCG/NDCG gains and (b) report the correlation between average bandit reward and standard IR metrics (e.g., NDCG@k or MRR) to validate alignment. In our production setting, we prioritize fast online learning and treat full IR-metric calibration as future work.

## 3 Methodology

### 3.1 Candidate Generation (Retrieval)

Our system follows a standard retrieval-and-reranking design. At query time we first obtain a candidate list of tokens from a semantic/fuzzy search API provided by a market-data provider (e.g., Birdeye[1]). This maps a free-form query to token identifiers and returns a small set of likely matches (typically tens to low hundreds, depending on provider limits). We apply light sanity filters (e.g., deduplication by contract address and verification checks) but do not learn or modify this retrieval stage. All learning occurs in the subsequent re-ranking stage: the bandit selects a scoring strategy that orders this provider-returned candidate set. In other words, the RL component ranks candidates; it does not perform retrieval.

### 3.2 UCB1 Strategy Selection

We employ the Upper Confidence Bound algorithm (UCB1) for strategy selection, which balances exploration and exploitation [3]:

$$\text{UCB}_i(t) = \bar{r}_i(t) + \sqrt{\frac{2 \ln t}{n_i(t)}}$$

where $\bar{r}_i(t)$ is the average reward for strategy $i$ up to time $t$, $n_i(t)$ is the number of times strategy $i$ has been

[1] `https://birdeye.so`

selected, and $t = \sum_{j=1}^{K} n_j(t)$ is the total number of selections.

The strategy selection policy chooses:

$$a^* = \arg \max_{i \in \{1,\ldots,K\}} \text{UCB}_i(t)$$

**Handling Drift.** In highly non-stationary settings, Sliding-Window UCB or Discounted UCB are drop-in replacements for UCB1 to better track drifting rewards, trading off bias and responsiveness [12].

### 3.3 Genetic Algorithm Evolution

To discover improved ranking strategies, we implement genetic algorithm evolution with the following components:

**Parent Selection:** Tournament selection with tournament size 3, selecting parents based on average reward with a penalty for strategies with insufficient data ($< 10$ pulls).

**Crossover:** Single-point crossover generating offspring:

$$\mathbf{w}_{\text{child}}[j] = \begin{cases} \mathbf{w}_{\text{parent1}}[j] & \text{if } j < c \\ \mathbf{w}_{\text{parent2}}[j] & \text{if } j \geq c \end{cases}$$

where $c$ is a randomly selected crossover point.

**Mutation:** Gaussian mutation with rate 0.2 and strength 0.1:

$$\mathbf{w}'[j] = \max(0.01, \mathbf{w}[j] + \mathcal{N}(0, 0.1))$$

**Normalization:** Ensure weights sum to 1:

$$\mathbf{w}_{\text{final}} = \frac{\mathbf{w}'}{\sum_{j=1}^{6} \mathbf{w}'[j]}$$

Evolution triggers during user navigation events when: (1) total interactions $\geq 50$, (2) best strategy has $\geq 20$ pulls, and (3) $\geq 100$ interactions since last evolution. The distributed nature means evolution occurs organically during active user sessions, with probabilistic pruning (5–10% chance) removing poor performers to maintain strategy diversity.

## 4 Implementation

### 4.1 Distributed Client-Side Architecture

Our implementation introduces a novel distributed evolution approach that operates entirely client-side while maintaining system-wide learning:

**Client-Side Processing:** All ranking computations occur in the browser using TypeScript, eliminating server round-trips for real-time ranking. Strategy selection, feature engineering, and ranking computation complete in under an estimated 70ms.

**Distributed Evolution:** Unlike traditional server-side batch evolution, our genetic algorithm runs client-side during user navigation events. When users navigate to token pages, the system probabilistically triggers evolution, creating new strategies through crossover and mutation of high-performing parents. This distributed approach ensures evolution occurs when users are actively engaged with the system.

**Evolution Computational Efficiency:** The lightweight nature of our 6-dimensional genetic algorithm makes client-side execution optimal. Core operations include tournament selection ($O(K)$ where $K \leq 20$ strategies), single-point crossover ($O(6)$ operations), Gaussian mutation ($O(6)$ operations), and weight normalization ($O(6)$ operations). The entire genetic algorithm completes in under 1ms, making network round-trips for server-side evolution 50–200$\times$ more expensive than local computation.

**Event-Driven Evolution Triggers:** Evolution occurs organically during user navigation rather than on fixed schedules. The system evaluates three conditions: (1) total interactions $\geq 50$, (2) best strategy has $\geq 20$ pulls, and (3) $\geq 100$ interactions since last evolution. This ensures evolution happens when sufficient statistical significance exists and users are actively engaging with the system.

**Distributed Load Characteristics:** Each client performs minimal computational work ($<$ 1ms genetic algorithm execution) distributed across navigation events. This approach eliminates server-side computational bottlenecks while maintaining natural evolution timing aligned with user activity patterns. The distributed nature scales linearly with user base growth.

**Database Persistence:** Persistent storage maintains strategy definitions and performance statistics, user feedback interactions for learning, and A/B testing validation metrics. The system gracefully handles database failures with comprehensive fallback mechanisms.

**Atomic Updates:** Bandit arm statistics are updated atomically through database functions ensuring consistency via a 5-step process: (1) fetch current statistics, (2) calculate new totals, (3) increment pull count, (4) compute new average, and (5) atomically update the strategy record.

### 4.2 Anti-Gaming Measures

To prevent manipulation, we implement several protective mechanisms:

**User Rate Limiting:** Time-weighted rewards with diminishing returns:

$$r_{\text{weighted}} = r_{\text{raw}} \times \max(0.1, 1.0/(1.0 + \text{interactions\_today} \times 0.1))$$

**Strategy Pruning:** Remove consistently poor performers: - Keep minimum 5 strategies - Prune evolved strategies performing $< 70\%$ of average with $\geq 50$ pulls - Preserve original baseline strategies

## 4.3 Production A/B Testing Infrastructure

To validate bandit effectiveness, we implement a sophisticated A/B testing framework:

**Deterministic User Assignment:** Users are deterministically assigned to bandit or baseline ranking based on a hash of their user ID, ensuring consistent experience across sessions and devices. This avoids the variance issues common in random assignment systems.

**Graceful Degradation:** The system includes comprehensive fallback mechanisms. If bandit arms cannot be loaded, the system automatically falls back to baseline ranking. If database operations fail, local caching maintains functionality. Strategy selection failures trigger immediate fallback to predetermined weights.

**Health Monitoring:** Real-time monitoring tracks system health ratios, fallback frequency, and performance metrics, enabling rapid detection of issues in production.

## 4.4 Performance Characteristics

**Latency:** Strategy selection: $< 5$ms, ranking computation: $< 50$ms for 100 tokens, genetic algorithm evolution: $< 1$ms

**Client-Side Computational Load:** The distributed approach minimizes per-client overhead: tournament selection processes $\leq 20$ strategies, crossover operates on 6-dimensional vectors, and mutation applies Gaussian noise to 6 weights. Total genetic algorithm operations: $O(K + 6) \approx 26$ simple operations per evolution event.

**Network Efficiency:** Client-side evolution eliminates the need for evolution-specific API calls. Network usage is limited to: initial strategy load ( 2KB), interaction logging ( 500 bytes), and new strategy persistence ( 200 bytes when evolution occurs). This compares favorably to server-side approaches requiring round-trips for evolution checks (50–200ms latency overhead).

**Memory:** Client-side state: $< 1$MB, database growth: 1KB per interaction

**Scalability:** Linear scaling with user base growth. Each additional user contributes distributed computational capacity while evolution frequency remains bounded by statistical significance requirements.

# 5 Production Deployment & Results

## 5.1 Strategy Performance

Our initial deployment includes 6 baseline strategies and has evolved 12 additional strategies through genetic algorithms. Table 1 shows performance metrics from real production use with actual users making token selection decisions.

**Key Findings:**

Table 1: Strategy Performance Summary

| Strategy Type | Avg Reward | Pulls | Gen |
|---|---|---|---|
| Baseline | 0.861 | 23 | N/A |
| Evolved Gen-5 | 0.955 | 20 | 5 |
| Evolved Gen-8 | 0.940 | 15 | 8 |

- Evolved strategies consistently outperform baselines
- Gen-5 strategies demonstrate the strongest performance with sufficient data (95.5% average reward)
- Gen-8 strategies maintain strong performance with reduced variance
- Weight evolution shows convergence toward verification-heavy strategies (25–35% allocation)

## 5.2 Weight Evolution Analysis

Figure 1 shows the evolution of strategy weights across generations, revealing clear convergence patterns:

**Verification Convergence:** Evolved strategies show consistent trend toward verification emphasis: 24.2% (Baseline) $\rightarrow$ 30.1% (Gen-5) $\rightarrow$ 32.2% (Gen-8), correlating with performance improvements.

**Activity Optimization:** Wallet and trade weights stabilize around 23% and 19% respectively in Gen-8, suggesting optimal balance for user selection prediction.

**Market Metrics Reduction:** Evolution reduces emphasis on raw market cap (11.8% $\rightarrow$ 8.7% $\rightarrow$ 11.9%) and liquidity (6.8% $\rightarrow$ 3.6% $\rightarrow$ 6.9%), indicating these features are less predictive than activity signals.

**Performance Correlation:** The increasing verification focus directly correlates with performance improvements: 76.5% $\rightarrow$ 85.6% $\rightarrow$ 88.5% average reward across generations.

## 5.3 Latency Analysis

Client-side ranking maintains low latency: strategy selection (3–8ms), feature engineering (15–25ms for 100 tokens), ranking computation (20–35ms), totaling 38–68ms. Database operations occur asynchronously: interaction logging (150–300ms) and strategy updates (50–150ms), both non-blocking.

# 6 Discussion

## 6.1 Comparison with E-commerce RL Approaches

Our lightweight approach differs significantly from prior e-commerce RL work, particularly Hu et al.'s SSMDP framework [1]. Related online learning-to-rank and slate methods include cascading and click-model bandits [7, 8],

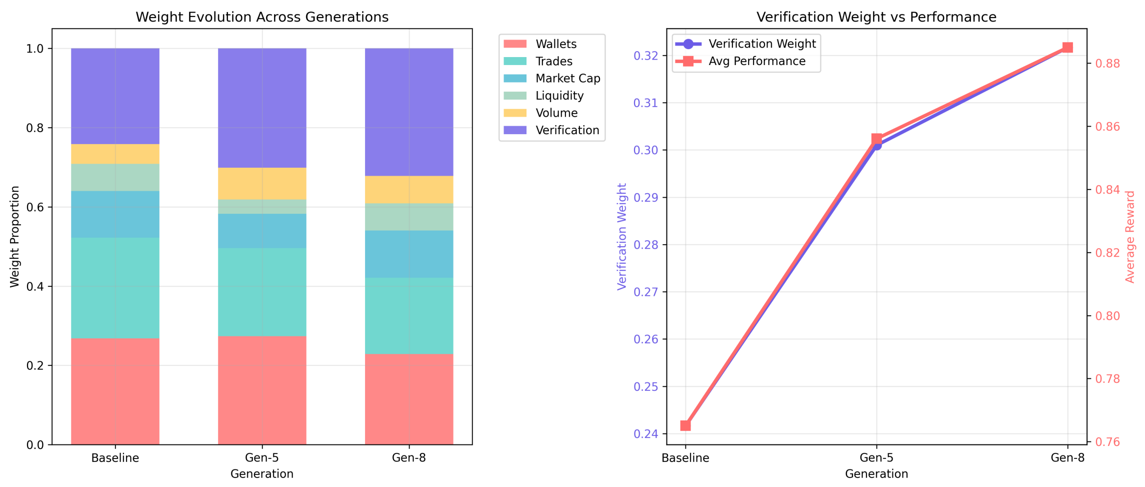

Figure 1: Weight evolution across generations showing increasing verification focus and corresponding performance improvements. Left: Stacked bar chart of weight distributions. Right: Verification weight vs. average performance trend.

diversity-aware ranked bandits [9], combinatorial semi-bandits for whole-page recommendation [10], and slate RL decompositions [11]:

**Simplified State Space:** While SSMDP models complex user session states and multi-step decision processes, our MAB approach treats each search as an independent arm selection problem, avoiding state modeling complexity.

**Deployment Constraints:** SSMDP requires server-side policy gradient computations and complex state transitions, whereas our approach operates entirely client-side with minimal computational overhead.

**Domain Adaptation:** E-commerce sessions benefit from modeling sequential purchases and long-term user journeys, while crypto search prioritizes rapid adaptation to market volatility over session continuity.

**Data Requirements:** SSMDP approaches require substantial historical session data for training, while our system learns effectively from minimal initial data through UCB1 exploration.

Despite these architectural differences, both approaches validate the core insight that RL provides significant advantages over traditional ranking methods in dynamic search environments.

## 6.2 Client-Side Evolution Advantages

Our distributed client-side evolution architecture provides several key advantages over traditional server-side approaches:

**Computational Efficiency:** For lightweight genetic algorithms operating on small parameter spaces (6 dimensions), client-side execution eliminates network latency bottlenecks. The genetic algorithm's $O(K + d)$ complexity where $K \leq 20$ strategies and $d = 6$ dimensions results in $< 1$ms execution time, while network round-trips introduce 50–200ms overhead.

**Natural Load Distribution:** Evolution computation distributes across the user base, with each client contributing minimal processing power during navigation events. This eliminates server-side computational bottlenecks and scales linearly with user growth.

**Event-Driven Timing:** Unlike fixed-schedule batch evolution, our approach triggers evolution during actual user engagement. This ensures new strategies are generated when users can immediately benefit from improved rankings, creating a tight feedback loop between user activity and system optimization.

**Elimination of Coordination Overhead:** Client-side evolution avoids the complexity of distributed server coordination, job scheduling, and resource management required by traditional batch processing approaches.

**Serverless-Friendly Deployment.** Modern frontend platforms (e.g., Vercel) default to serverless compute and CDN caching. A client-side bandit/evolution loop is simpler to ship and operate in this model, since it:

- avoids server-side cache coherence/invalidation and per-user/session caching strategies,
- sidesteps cold-start latency and stateful coordination in serverless functions, and
- reduces additional infrastructure (custom caches, in-memory stores) otherwise needed to keep server-side ranking fresh without serving stale/poisoned cache entries.

State persists in the database, while selection and evolution execute in the browser, making deployment and scaling straightforward on serverless stacks.

### 6.3 Domain-Specific Advantages

Our approach is particularly well-suited to cryptocurrency search due to:

**Rapid Adaptation:** UCB1 adapts to changing token popularity within hours rather than days or weeks required by traditional ML retraining [12].

**Real-time Learning:** Every user interaction immediately influences future rankings, enabling responsive adaptation to market events.

**Multi-dimensional Optimization:** The 6-feature space captures both market fundamentals and trust signals crucial for crypto users.

**Duplicate Handling:** By learning user preferences for verification and activity metrics, the system naturally improves disambiguation between similar tokens.

### 6.4 Production Validation

Unlike laboratory simulations, our system operates under real-world constraints with actual users making consequential decisions. The distributed evolution architecture provides several advantages over traditional approaches: (1) evolution occurs during peak user engagement, (2) computational load is distributed across user sessions, and (3) strategy improvements happen when users can immediately benefit from enhanced rankings.

### 6.5 Scalability Considerations

The lightweight architecture scales favorably: client-side processing distributes computational load, database growth is linear ($\sim$1KB per interaction), strategy evolution is periodic rather than continuous, and A/B testing enables controlled rollout.

### 6.6 Limitations

While our approach addresses many challenges in crypto search, several limitations warrant consideration:

**Feature Correlation:** Despite log-transformation and derived feature engineering, some correlation persists between market metrics (e.g., volume and market cap), potentially reducing the independence of our 6-dimensional feature space and limiting the genetic algorithm's ability to discover truly orthogonal optimization directions.

**Limited Contextual Awareness:** The current system treats each search independently without considering query intent classification, user search history, or session context. This prevents personalization and may miss opportunities to adapt rankings based on whether users are exploring new tokens versus searching for specific known assets.

**Feature Space Constraints:** Our 6-dimensional approach, while lightweight, may miss important signals such as social sentiment, on-chain activity patterns, or temporal market trends that could enhance ranking quality, particularly for emerging tokens in volatile market conditions.

**Evaluation Bias:** If we evaluate or iterate on policies offline using logged interactions, counterfactual/unbiased learning-to-rank and off-policy evaluation are required to correct for position and selection bias [14, 15].

**Trust**
**Safety:** While we employ rate limiting, pruning, and deterministic A/B assignment to reduce gaming and instability, risks remain (e.g., Sybil attacks, surfacing low-quality or deceptive tokens). We mitigate by emphasizing verification signals, monitoring anomalies, and retaining graceful fallbacks.

## 7 Future Work

### 7.1 Enhanced Feature Engineering

Additional dimensions could capture:

- **Temporal patterns:** Recent trend direction, momentum indicators
- **Social signals:** Twitter mentions, sentiment scores, community size
- **Technical indicators:** Price volatility, volume profile, liquidity depth
- **Ecosystem metrics:** DEX listing count, holder distribution, transaction patterns

### 7.2 Contextual Bandits (Future Work)

Evolution toward contextual bandits could incorporate [6]:

- Query intent classification (exploration vs. specific search)
- User behavior patterns and preferences
- Temporal context (market hours, news events)
- Portfolio correlation and risk metrics

Practical note: linear contextual methods (e.g., LinUCB) remain lightweight and are compatible with our client-side and serverless deployment constraints.

### 7.3 Deep RL Integration

Future work could explore:

- Neural bandit policies for complex feature interactions
- Reinforcement learning for query understanding
- Multi-objective optimization balancing relevance, diversity, and trust

- Transfer learning between similar token ecosystems

Given tight latency budgets and serverless execution models, we defer deep RL to future iterations where on-device inference or hybrid server/client training can be justified.

## 8 Conclusion

We presented a production deployment of a distributed, client-side bandit system for cryptocurrency token search, achieving sub-100ms ranking with continuous adaptation from real user interactions. Our architecture evolves strategies in the browser during navigation events, providing a practical alternative to server-side batch evolution.

The UCB1-based bandit with lightweight genetic evolution delivers strong outcomes (up to 95% average reward on our positional scale), aided by robust fallbacks and deterministic A/B assignment. The approach scales naturally on serverless stacks, and the 6-dimensional feature space captures the domain's key signals.

Looking ahead, we plan to (i) introduce drift-aware UCB variants as drop-in replacements, (ii) explore linear contextual bandits for personalization within our latency/footprint budget, and (iii) evaluate deeper RL integration where justified. We believe this client-side paradigm generalizes to other dynamic ranking problems that demand real-time adaptation without heavy infrastructure.

## References

[1] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*, 2018.

[2] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. Deep reinforcement learning for search, recommendation, and online advertising: A survey. *ArXiv preprint arXiv:1812.07127*, 2018.

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

[4] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[5] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[6] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

[7] Branislav Kveton, Csaba Szepesvári, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML*, 2015.

[8] Masrour Zoghi, Tomàs Tunys, Mohammad Ghavamzadeh, Branislav Kveton, Csaba Szepesvári, and Zheng Wen. Online learning to rank in stochastic click models. In *ICML*, 2017.

[9] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, 2008.

[10] Yingfei Wang, Hua Ouyang, Chu Wang, Jianhui Chen, Tsvetan Asamov, and Yi Chang. Efficient ordered combinatorial semi-bandits for whole-page recommendation. In *AAAI*, 2017.

[11] Edoardo M. A. Ie, Chih-Wei Hsu, Martin Mladenov, et al. SlateQ: A tractable decomposition for reinforcement learning with recommendation sets. In *AAAI*, 2019.

[12] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv:0805.3415*, 2008.

[13] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. In *NeurIPS*, 2014.

[14] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *WSDM*, 2017.

[15] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *ICML*, 2011.