Chapter 1: Search Ranking as Optimization

Vlad Prytula

December 30, 2025

Contents

1 Chapter 1 — Search Ranking as Optimization: From Business Goals to RL

Vlad Prytula

1.1 1.1 The Problem: Balancing Multiple Objectives in Search

A concrete dilemma. A pet supplies retailer faces a challenge. User A searches for "cat food"—a price-sensitive buyer who abandons carts if shipping costs are high. User B issues the same query—a premium shopper loyal to specific brands, willing to pay more for quality. The current search system shows them **identical rankings** because boost weights are static, tuned once for the "average" user. User A sees expensive premium products and abandons. User B sees discount items and questions the retailer's quality. Both users are poorly served by a one-size-fits-all approach.

The business tension. Every e-commerce search system must balance competing objectives:

- **Revenue (GMV)**: Show products users will buy, at good prices
- **Profitability (CM2)**: Prioritize items with healthy margins
- **Strategic goals (STRAT)**: Expose new products, house brands, or inventory to clear
- **User experience**: Maintain relevance, diversity, and satisfaction

Traditional search systems rely on **manually tuned boost parameters**: category multipliers, price/discount bonuses, profit margins, strategic product flags. Before writing the scoring function, we fix our spaces.

Spaces (Working Definitions).

- $\mathcal{P}$: **Product catalog**, a finite set of M products. Each $p \in \mathcal{P}$ carries attributes (price, category, margin, embedding).
- $\mathcal{Q}$: **Query space**, the set of possible search queries. In practice, a finite vocabulary or embedding space $\mathcal{Q} \subset \mathbb{R}^{d_q}$.
- $\mathcal{U}$: **User space**, characterizing users by segment, purchase history, and preferences. Finite segments or embedding space $\mathcal{U} \subset \mathbb{R}^{d_u}$.
- $\mathcal{X}$: **Context space**, typically $\mathcal{X} \subseteq \mathcal{U} \times \mathcal{Q} \times \mathcal{H} \times \mathcal{T}$ where $\mathcal{H}$ is session history and $\mathcal{T}$ is time features. We assume $\mathcal{X}$ is a compact subset of $\mathbb{R}^{d_x}$ for some d_x (verified in Chapter 4).
- $\mathcal{A} = [-a_{\max}, +a_{\max}]^K$: **Action space**, a compact subset of $\mathbb{R}^K$ (boost weights bounded by $a_{\max} > 0$).
- Ω: **Outcome space**, the sample space for stochastic user behavior (clicks, purchases, abandonment). Equipped with probability measure $\mathbb{P}$ (formalized in Chapter 2).

Measure-theoretic structure (σ-algebras, probability kernels, conditional distributions) is developed in Chapter 2. For this chapter, we work with these as sets supporting the functions and expectations below.

A typical scoring function looks like:

$$s : \mathcal{P} \times \mathcal{Q} \times \mathcal{U} \to \mathbb{R}, \quad s(p, q, u) = r_{\text{ES}}(q, p) + \sum_{k=1}^{K} w_k \phi_k(p, u, q) \tag{1.1}$$

where: - $r_{\text{ES}} : \mathcal{Q} \times \mathcal{P} \to \mathbb{R}_+$ is a **base relevance score** (e.g., BM25 or neural embeddings) - $\phi_k : \mathcal{P} \times \mathcal{U} \times \mathcal{Q} \to \mathbb{R}$ are **engineered features** (margin, discount, bestseller status, category match) - $w_k \in \mathbb{R}$ are **manually tuned weights**, collected as $\mathbf{w} = (w_1, \ldots, w_K) \in \mathbb{R}^K$

Note that we've made the user dependence explicit: $s(p, q, u)$ depends on product $p \in \mathcal{P}$, query $q \in \mathcal{Q}$, **and user** $u \in \mathcal{U}$ through the feature functions ϕ_k.

Why manual tuning fails. The core problem: w_k cannot adapt to context. The "price hunter" (User A) cares about bulk pricing and discounts. The "premium shopper" (User B) values quality over price. A generic query ("cat food") tolerates exploration; a specific query ("Royal Canin Veterinary Diet Renal Support") demands precision.

Numerical evidence of the problem. Suppose we tune $w_{\text{discount}} = 2.0$ to maximize average GMV across all users. For price hunters, this works well—they click frequently on discounted items. But for premium shoppers, this destroys relevance—they see cheap products ranked above their preferred brands, leading to zero purchases and session abandonment. Conversely, if we tune $w_{\text{discount}} = 0.3$ for premium shoppers, price hunters see full-price items and also abandon.

Manual weights are **static, context-free, and suboptimal** by design. We need weights that adapt.

Our thesis: Treat $\mathbf{w} = (w_1, \ldots, w_K) \in \mathbb{R}^K$ as **actions to be learned**, adapting to user and query context via reinforcement learning.

In Chapter 0 (Motivation: Your First RL Experiment), we built a tiny, code-first prototype of this idea: three synthetic user types, a small action grid of boost templates, and a tabular Q-learning agent that learned context-adaptive boosts. In this chapter, we strip away implementation details and **formalize and generalize** that experiment as a contextual bandit with constraints.

> **Notation**
>
> Throughout this chapter: - **Spaces**: $\mathcal{X}$ (contexts), $\mathcal{A}$ (actions), Ω (outcomes), $\mathcal{Q}$ (queries), $\mathcal{P}$ (products), $\mathcal{U}$ (users) - **Distributions**: ρ (context distribution over $\mathcal{X}$), $P(\omega \mid x, a)$ (outcome distribution) - **Probability**: $\mathbb{P}$ (probability measure), $\mathbb{E}$ (expectation) - **Real/natural numbers**: $\mathbb{R}$, $\mathbb{N}$, $\mathbb{R}_+$ (non-negative reals) - **Norms**: $\|\cdot\|_2$ (Euclidean), $\|\cdot\|_\infty$ (supremum) - **Operators**: $\mathcal{T}$ (Bellman operator, introduced in Chapter 3)
>
> We index equations as EQ-X.Y, theorems as THM-X.Y, definitions as DEF-X.Y, remarks as REM-X.Y, and assumptions as ASM-X.Y for cross-reference. Anchors like `[]{#THM-1.7.2}` enable internal linking.

> **On Mathematical Rigor**
>
> This chapter provides **working definitions** and builds intuition for the RL formulation. We specify function signatures (domains, codomains, types) but defer **measure-theoretic foundations**—σ-algebras on $\mathcal{X}$ and Ω, measurability conditions, integrability requirements—to **Chapters 2–3**. Key results (existence of optimal policies and the regret lower-bound preview in §1.7.6) state their assumptions explicitly; verification that our search setting satisfies these assumptions appears in later chapters. Readers seeking Bourbaki-level rigor should treat this chapter as motivation and roadmap; the rigorous development begins in Chapter 2.

This chapter establishes the mathematical foundation: we formulate search ranking as a **constrained optimization problem**, then show why it requires **contextual decision-making** (bandits), and finally preview the RL framework we'll develop.

1.2 1.2 From Clicks to Outcomes: The Reward Function

Let's make the business objectives precise. Consider a single search session:

1. **User** u with segment $\sigma \in \{\text{price\_hunter}, \text{pl\_lover}, \text{premium}, \text{litter\_heavy}\}$ issues **query** q
2. System scores products $\{p_1, \ldots, p_M\}$ using boost weights $\mathbf{w}$, producing ranking π
3. User examines results with **position bias** (top slots get more attention), clicks on subset $C \subseteq \{1, \ldots, M\}$, purchases subset $B \subseteq C$
4. Session generates **outcomes**: GMV, CM2 (contribution margin 2), clicks, strategic purchases

We aggregate these into a **scalar reward**:

$$R(\mathbf{w}, u, q, \omega) = \alpha \cdot \text{GMV}(\mathbf{w}, u, q, \omega) + \beta \cdot \text{CM2}(\mathbf{w}, u, q, \omega) + \gamma \cdot \text{STRAT}(\mathbf{w}, u, q, \omega) + \delta \cdot \text{CLICKS}(\mathbf{w}, u, q, \omega) \tag{1.2}$$

where $\omega \in \Omega$ represents the stochastic user behavior conditioned on the ranking $\pi_{\mathbf{w}}(u, q)$ induced by boost weights $\mathbf{w}$, and $(\alpha, \beta, \gamma, \delta) \in \mathbb{R}^4_+$ are **business weight parameters** reflecting strategic priorities. The outcome components (GMV, CM2, STRAT, CLICKS) depend on the full context $(\mathbf{w}, u, q)$ through the ranking, though we often abbreviate this dependence when clear from context.

Two strategic quantities: reward vs. constraints

In the reward [EQ-1.2], STRAT(ω) counts **strategic purchases** in the session (purchased items whose `strategic_flag` is true). In guardrails like [EQ-1.3b], we instead track **strategic exposure**—how many strategic items were shown in the ranking, regardless of whether they were bought.
We keep both on purpose: reward incentivizes realized strategic outcomes, while exposure floors enforce minimum visibility even before conversion. In code, the reward-side quantity appears as `RewardBreakdown.strat` in `zoosim/dynamics/reward.py:34-39`.

Standing assumption (Integrability). Throughout this chapter, we assume $R : \mathcal{A} \times \mathcal{U} \times \mathcal{Q} \times \Omega \to \mathbb{R}$ is measurable in ω and $\mathbb{E}[|R(\mathbf{w}, u, q, \omega)|] < \infty$ for all $(\mathbf{w}, u, q)$. This ensures expectations like $\mathbb{E}[R \mid \mathbf{w}]$ are well-defined. The formal regularity conditions appear as **Assumption 2.6.1 (OPE Probability Conditions)** in Chapter 2, §2.6; verification for our bounded-reward setting is in Chapter 2.

Remark (connection to Chapter 0). The Chapter 0 toy used a simplified instance of this reward with $(\alpha, \beta, \gamma, \delta) \approx (0.6, 0.3, 0, 0.1)$ and no explicit STRAT term. All analysis in this chapter applies to that setting.

Key insight: R depends on $\mathbf{w}$ **indirectly** through the ranking π induced by scores from (1.1). A product ranked higher gets more exposure, more clicks, and influences downstream purchases. This is **not a simple function**—it's stochastic, nonlinear, and noisy.

1.2.1 Constraints: Not All Rewards Are Acceptable

High GMV alone is insufficient. A retailer must enforce **guardrails**:

$$\mathbb{E}[\text{CM2} \mid \mathbf{w}] \geq \tau_{\text{margin}} \tag{1.3a}$$

$$\mathbb{E}[\text{Exposure}_{\text{strategic}} \mid \mathbf{w}] \geq \tau_{\text{strat}} \tag{1.3b}$$

$$\mathbb{E}[\Delta\text{rank@}k \mid \mathbf{w}] \leq \tau_{\text{stability}} \tag{1.3c}$$

where the notation $\mathbb{E}[\cdot \mid \mathbf{w}]$ denotes expectation over stochastic user behavior ω and context distribution $\rho(x)$ when action (boost weights) $\mathbf{w}$ is applied, i.e., $\mathbb{E}[\text{CM2} \mid \mathbf{w}] := \mathbb{E}_{x\sim\rho,\omega\sim P(\cdot|x,\mathbf{w})}[\text{CM2}(\mathbf{w}, x, \omega)]$.

Definition (Δrank@k). Let $\pi_{\mathbf{w}}(q) = (p_1, \ldots, p_M)$ be the ranking induced by boost weights $\mathbf{w}$ for query q, and let $\pi_{\text{base}}(q)$ be a reference ranking (e.g., the production baseline). Let $\text{TopK}_{\mathbf{w}}(q)$ and $\text{TopK}_{\text{base}}(q)$ denote the *sets* of top-k items under these rankings. Define:

$$\Delta\text{rank@}k(\mathbf{w}, q) := 1 - \frac{|\text{TopK}_{\mathbf{w}}(q) \cap \text{TopK}_{\text{base}}(q)|}{k}$$

the fraction of top-k items that changed (set churn). Values range in $[0, 1]$; $\Delta\text{rank@}k = 0$ means identical top-k *set* (reordering within the top-k does not count), and $\Delta\text{rank@}k = 1$ means the two top-k sets are disjoint.

This is the set-based stability metric used in Chapter 10 **??** and implemented in `zoosim/monitoring/metric`. A position-wise mismatch rate is a different metric; if we use it, we will name it explicitly and not call it "Delta-Rank@k".

- **CM2 floor** (1.3a): Prevent sacrificing profitability for revenue
- **Exposure floor** (1.3b): Ensure strategic products (new launches, house brands) get visibility
- **Rank stability** (1.3c): Limit reordering volatility (users expect consistency); $\tau_{\text{stability}} \approx 0.2$ is typical

Taken together, the scalar objective (1.2) and constraints (1.3a–c) define a **constrained stochastic optimization** problem over the boost weights $\mathbf{w}$. Formally, we would like to choose $\mathbf{w}$ to solve

$$\max_{\mathbf{w}\in\mathbb{R}^K} \mathbb{E}_{x\sim\rho,\omega\sim P(\cdot|x,\mathbf{w})}\big[R(\mathbf{w}, x, \omega)\big] \quad \text{subject to (1.3a–c)}.$$

From the perspective of a single query, there is no internal state evolution: each query arrives with a context x, we apply fixed boost weights $\mathbf{w}$, observe a random outcome, and then move on to the next independent query. This "context + one action + one noisy payoff" structure is exactly the **contextual bandit** template.

In §1.3 we move from a single global choice of $\mathbf{w}$ to an explicit **policy** π that maps each context x to boost weights $\pi(x)$. In **Chapter 11**, when we introduce multi-step user/session dynamics with states and transitions, the resulting model becomes a **constrained Markov decision process (CMDP)**. Contextual bandits are the $\gamma = 0$ special case of an MDP.

Now we understand the complete optimization problem: maximize the scalar reward (1.2) subject to constraints (1.2.1). This establishes what we're optimizing. Next, we'll dive deep into one critical component—the engagement term—before implementing the reward function.

Code $\leftrightarrow$ Config (constraints)

Constraint-related knobs (`MOD-zoosim.config`) live in configuration so experiments remain reproducible and auditable. These implement (1.2.1) constraint definitions:

- Rank stability penalty weight: `lambda_rank` in `zoosim/core/config.py:230`
- Profitability floor (CM2): `cm2_floor` in `zoosim/core/config.py:232` (defined in config; enforcement is introduced in Chapter 10)
- Exposure floors (strategic products): `exposure_floors` in `zoosim/core/config.py:233` (defined in config; enforcement is introduced in Chapter 10)

These are surfaced in `ActionConfig` and wired into downstream modules as they mature.

1.2.2 1.2.1 The Role of Engagement in Reward Design

In practice, search objectives are **hierarchically structured**, not flat:

1. **Viability constraints** (must satisfy or system is unusable): CTR > 0, latency < 500ms, uptime $> 99.9\%$
2. **Business outcomes** (what we optimize): GMV, profitability (CM2), strategic positioning
3. **Strategic nudges** (tiebreakers for long-term value): exploration, new product exposure, brand building

Engagement (clicks, dwell time, add-to-cart actions) **straddles this hierarchy**: it is partly viability (zero clicks $\Rightarrow$ dead search, users abandon platform), partly outcome (clicks signal incomplete attribution—mobile browse, desktop purchase), and partly strategic (exploration value—today's clicks reveal preferences for tomorrow's sessions).

Why include $\delta \cdot$ CLICKS in the reward?

We include $\delta \cdot \text{CLICKS}$ as a **soft viability term** in the reward function (1.2). This serves three purposes:

1. **Incomplete attribution**: E-commerce has imperfect conversion tracking. A user clicks product p on mobile, adds to cart, completes purchase on desktop 3 days later. We observe the click, but GMV attribution goes to a different session (or is lost entirely in cross-device gaps). The click is a **leading indicator** of future GMV not captured in ω.

2. **Exploration value**: Clicks reveal user preferences even without immediate purchase. If user u clicks on premium brand products but doesn't convert, we learn u is exploring that segment—valuable for future sessions. This is **information acquisition**: clicks are samples from the user's latent utility function.

3. **Platform health**: A search system with high GMV but near-zero CTR is **brittle**—one price shock or inventory gap causes catastrophic user abandonment. Engagement is a **leading indicator of retention**: users who click regularly have higher lifetime value (LTV) than those who occasionally convert high-value purchases but otherwise ignore search results.

The clickbait risk. However, δ **must be carefully bounded**. If δ/α is too large, the agent learns **"clickbait" strategies**: optimize CTR at the expense of conversion rate (CVR = purchases/clicks). The pathological case: show irrelevant but visually attractive products (e.g., cute cat toys for dog owners), achieve high clicks but zero sales, and still get rewarded due to $\delta \cdot \text{CLICKS} \gg 0$.

Practical guideline: Set $\delta/\alpha \in [0.01, 0.10]$—engagement is a *tiebreaker*, not the primary objective. We want clicks to be a **soft regularizer** that prevents GMV-maximizing policies from collapsing engagement, not a dominant term that drives the optimization.

Diagnostic metric: Monitor **revenue per click (RPC)**

$$\text{RPC}_t = \frac{\sum_{i=1}^{t} \text{GMV}_i}{\sum_{i=1}^{t} \text{CLICKS}_i}$$

(cumulative GMV per click up to episode t). If RPC_t drops $> 10\%$ below baseline while CTR rises during training, the agent is learning clickbait—reduce δ immediately.

Control-theoretic analogy: This is similar to LQR with **state and control penalties**: $c(x, u) = x^\top Q x + u^\top R u$. We penalize both deviation from target state (GMV, CM2) and control effort (engagement as "cost" of achieving GMV). The relative weights Q, R encode the tradeoff. In our case, $\alpha, \beta, \gamma, \delta$ play the role of Q, and we're learning the optimal policy $\pi^*(x)$ under this cost structure. See Appendix B (and Section 1.10) for deeper connections to classical control.

Multi-episode perspective (Chapter 11 preview): In a **Markov Decision Process (MDP)** with inter-session dynamics, engagement enters *implicitly* through its effect on retention and lifetime value:

$$V^\pi(s_0) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \text{GMV}_t \mid s_0 \right] \tag{1.2'}$$

If today's clicks increase the probability that user u returns tomorrow (state transition $s_{t+1} = f(s_t, \text{clicks}_t, ...)$), then maximizing (1.2.2)e automatically incentivizes engagement. We wouldn't need $\delta \cdot \text{CLICKS}$ in the single-step reward—it would be *derived* from optimal long-term value.

However, the **single-step contextual bandit** (our MVP formulation) cannot model inter-session dynamics. Each search is treated as independent: user arrives, we rank, user interacts, episode terminates. No s_{t+1}, no retention modeling. Including $\delta \cdot \text{CLICKS}$ is a **heuristic proxy** for the missing LTV component—mathematically imperfect, but empirically essential for search systems.

The honest assessment: This is a **theory-practice tradeoff**. The "correct" formulation is (1.2.2)e (multi-episode MDP), but it requires modeling complex user dynamics (churn, seasonality, cross-session preferences) that are expensive to simulate and hard to learn from. The single-step approximation (1.2) with $\delta \cdot \text{CLICKS}$ is **pragmatic**: it captures 80% of the value with 20% of the complexity. For the MVP, this is the right tradeoff. Chapter 11 extends to multi-episode settings where engagement is properly modeled as state dynamics.

Cross-reference — Chapter 11

The full multi-episode treatment and implementation live in `Chapter 11 --- Multi-Episode Inter-Session MDP` (see `docs/book/syllabus.md`). There we add `zoosim/multi_episode/session_env.py` and `zoosim/multi_episode/retention.py` to operationalize (1.2.2)e with a retention/hazard state and validate that engagement raises long-term value without needing an explicit $\delta \cdot \text{CLICKS}$ term.

Code $\leftrightarrow$ Config (reward weights)

Business weights in `RewardConfig` (`MOD-zoosim.config`) implement (1.2) parameters and must satisfy engagement bounds from this section:

- α (GMV): Primary objective, normalized to 1.0 by convention
- β/α (CM2 weight): Profit sensitivity, typically $\in [0.3, 0.8]$ (higher $\Rightarrow$ prioritize margin over revenue)
- γ/α (STRAT weight): Strategic priority, typically $\in [0.1, 0.3]$ (house brands, new products, clearance)
- δ/α **(CLICKS weight): Bounded** $\in [0.01, 0.10]$ **to prevent clickbait strategies**

Validation (enforced in code): see `zoosim/dynamics/reward.py:56` for an assertion on δ/α in the production reward path. The numerical range $[0.01, 0.10]$ is an engineering guardrail motivated by clickbait failure modes; Appendix C provides the duality background for constrained optimization, not a derivation of this specific bound.
Diagnostic: Compute $\text{RPC}_t = \sum \text{GMV}_i / \sum \text{CLICKS}_i$ after each policy update. If RPC drops $> 10\%$ while CTR rises, reduce δ by 30–50%.

Code $\leftrightarrow$ Simulator Layout

- `zoosim/core/config.py` (`MOD-zoosim.config`): SimulatorConfig/RewardConfig with seeds, guardrails, and reward weights
- `zoosim/world/{catalog,users,queries}.py`: deterministic catalog + segment + query generation (Chapter 4)
- `zoosim/ranking/{relevance,features}.py`: base relevance and boost feature engineering (Chapter 5)
- `zoosim/dynamics/{behavior,reward}.py` (`MOD-zoosim.behavior`, `MOD-zoosim.reward`): click/abandonment dynamics + reward aggregation for (1.2)
- `zoosim/envs/{search_env.py,gym_env.py}` (`MOD-zoosim.env`): single-step environment and Gym wrapper wiring the simulator together
- `zoosim/multi_episode/{session_env.py,retention.py}`: Chapter 11's retention-aware MDP implementing (1.2.2)e

1.2.3 Verifying the Reward Function

Before diving into theory, let's implement (1.2) and see what it does:

```python
# Minimal implementation of #EQ-1.2 (full version: Lab 1.3 in exercises_labs.md)
def compute_reward(gmv, cm2, strat, clicks, alpha=1.0, beta=0.5, gamma=0.2, delta=0.1):
    """R = alpha*GMV + beta*CM2 + gamma*STRAT + delta*CLICKS"""
    return alpha * gmv + beta * cm2 + gamma * strat + delta * clicks

# Strategy A (GMV-focused): gmv=120, cm2=15, strat=1, clicks=3
# Strategy B (Balanced):    gmv=100, cm2=35, strat=3, clicks=4
R_A = compute_reward(120, 15, 1, 3)  # = 128.00
R_B = compute_reward(100, 35, 3, 4)  # = 118.50
```

Strategy	GMV	CM2	STRAT	CLICKS	Reward
A (GMV-focused)	120	15	1	3	**128.00**
B (Balanced)	100	35	3	4	118.50

Wait—Strategy A won? With profitability-focused weights ($\alpha = 0.5, \beta = 1.0, \gamma = 0.5, \delta = 0.1$), the result flips: Strategy A scores 75.80, Strategy B scores **86.90**. The optimal strategy depends on business weights—this is a multi-objective tradeoff, not a fixed optimization. See **Lab 1.3–1.4** for full implementations and weight sensitivity analysis.

Revenue-per-click diagnostic (clickbait detection): Strategy A gets fewer clicks (3 vs 4) but 60% higher GMV per click (EUR 40 vs EUR 25)—*quality over quantity*. The metric RPC = GMV/CLICKS monitors for clickbait: if RPC drops while CTR rises, reduce δ immediately. See **Lab 1.5** for the full implementation with alerting thresholds.

The bound $\delta/\alpha = 0.10$ is at the upper limit. We recommend starting with $\delta/\alpha = 0.05$ and monitoring RPC over time. If RPC degrades, the agent has learned to exploit the engagement term.

Code $\leftrightarrow$ Simulator

The minimal example above mirrors the simulator's reward path. In production, `RewardConfig` (`MOD-zoosim.config`) in `zoosim/core/config.py` holds the business weights, and `compute_reward` (`MOD-zoosim.reward`) in `zoosim/dynamics/reward.py` implements (1.2) aggregation with a detailed breakdown. Keeping these constants in configuration avoids magic numbers in code and guarantees reproducibility across experiments.
RPC monitoring (for production deployment): Log $\text{RPC}_t = \sum_{i=1}^{t} \text{GMV}_i / \sum_{i=1}^{t} \text{CLICKS}_i$ as a running average per Section 1.2.1. Alert if RPC drops $> 10\%$ below baseline. See Chapter 10 (Robustness) for drift detection and automatic δ adjustment.

Key observation: The **optimal strategy depends on business weights** $(\alpha, \beta, \gamma, \delta)$. This is not a fixed optimization problem—it's a **multi-objective tradeoff** that requires careful calibration. In practice, these weights are set by business stakeholders, and the RL system must respect them.

1.3 1.3 The Context Problem: Why Static Boosts Fail

Current production systems use **fixed boost weights** $\mathbf{w}_{\text{static}}$ for all queries. Let's see why this fails.

1.3.1 Experiment: User Segment Heterogeneity

Simulate two user types with different preferences. See `zoosim/dynamics/behavior.py` for the production click/abandonment model; the toy model in **Lab 1.6** is simplified for exposition.

Code $\leftrightarrow$ Behavior (production click model)

Production (`MOD-zoosim.behavior`, concept `CN-ClickModel`) implements an examination–click–purchase process with position bias:

- Click probability: `click_prob = sigmoid(utility)` in `zoosim/dynamics/behavior.py`
- Position bias: `_position_bias()` using `BehaviorConfig.pos_bias` in `zoosim/core/config.py`
- Purchase: `sigmoid(buy_logit)` in `zoosim/dynamics/behavior.py`

Chapter 2 formalizes click models and position bias; Chapter 5 connects these to off-policy evaluation.

Experiment results (full implementation: **Lab 1.6** in `exercises_labs.md`):
With static discount boost $w = 2.0$, user segments respond dramatically differently:

User Type	Expected Clicks	Relative Performance
Price hunter	0.997	Baseline
Premium shopper	0.428	**57% fewer clicks**

Analysis: The static weight is **over-optimized for price hunters** and **under-performs for premium shoppers**. Ideally, we'd adapt per segment: price hunters get $w_{\text{discount}} \approx 2.0$, premium shoppers get $w_{\text{discount}} \approx 0.5$. But production systems use **one global w**—this is wasteful.

> **Note (Toy vs. Production Models):** The toy model uses linear utility and multiplicative position bias. Production uses sigmoid probabilities, calibrated position bias from `BehaviorConfig`, and an examination–click–purchase cascade. The toy suffices to show **user heterogeneity**; Chapter 2 develops the full PBM/DBN click model with measure-theoretic foundations.

1.3.2 The Context Space

Define **context** x as the information available at ranking time:

$$x = (u, q, h, t) \in \mathcal{X} \tag{1.4}$$

where: - u: User features (segment, past purchases, location) - q: Query features (tokens, category, specificity) - h: Session history (coarse, not full trajectory—this is a bandit) - t: Time features (seasonality, day-of-week)

Key insight: The optimal boost weights $\mathbf{w}^*(x)$ should be a **function of context**. This transforms our problem from:

$$\text{Static optimization:} \quad \max_{\mathbf{w} \in \mathbb{R}^K} \mathbb{E}_{x \sim \rho, \omega \sim P(\cdot|x,\mathbf{w})}[R(\mathbf{w}, x, \omega)] \tag{1.5}$$

to:

$$\text{Contextual optimization:} \quad \max_{\pi: \mathcal{X} \to \mathbb{R}^K} \mathbb{E}_{x \sim \rho, \omega \sim P(\cdot|x,\pi(x))}[R(\pi(x), x, \omega)] \tag{1.6}$$

where we've made the conditioning explicit: ω is drawn **after** observing context x and choosing action $a = \pi(x)$, consistent with the causal graph $x \to a \to \omega \to R$.

This is **no longer a static optimization problem**—it's a **function learning problem**. We must learn a **policy** π that maps contexts to actions. Welcome to reinforcement learning.

1.4 1.4 Contextual Bandits: The RL Formulation

Let's formalize the RL setup. We'll start with the **single-step (contextual bandit)** framing, then preview the full MDP extension.

1.4.1 Building Intuition: Why "Bandit" Not "MDP"?

In traditional RL, an agent interacts with an environment over multiple timesteps, and actions affect future states (e.g., a robot's position determines what it can reach next). In search ranking, each query is **independent**—showing User A a certain ranking doesn't change what User B sees when they search later. There's no "state" that evolves over time within a single session. This simplification is called a **contextual bandit**: one-shot decisions conditioned on context, with no sequential dependencies.

Let's build up the components incrementally:

Context $\mathcal{X}$: "What do we observe before choosing boosts?" - User features: segment (price_hunter, premium, litter_heavy, pl_lover), purchase history, location - Query features: tokens, category match, query specificity - Session context: time of day, device type, recent browsing - In our pet supplies example: $(u = \text{premium}, q = \text{"cat food"}, h = \text{empty cart}, t = \text{evening})$

Action $\mathcal{A}$: "What do we control?" - Boost weights $\mathbf{w} \in [-a_{\max}, +a_{\max}]^K$ for K features (discount, margin, private label, bestseller, recency) - Bounded to prevent catastrophic behavior: $|w_k| \leq a_{\max}$ (typically $a_{\max} \in [0.3, 1.0]$) - Continuous space—not discrete arms like classic bandits

Reward R: "What do we optimize?" - Scalar combination from (1.2): $R = \alpha \cdot \text{GMV} + \beta \cdot \text{CM2} + \gamma \cdot \text{STRAT} + \delta \cdot \text{CLICKS}$ - Stochastic—depends on user behavior ω (clicks, purchases) - Observable after each search session

Distribution ρ: "How are contexts sampled?" - Real-world query stream from users - We don't control this—contexts arrive from the environment - Must generalize across the distribution of contexts

Now we can formalize this as a mathematical object.

1.4.2 Problem Setup

Working Definition 1.4.1 (Contextual Bandit for Search Ranking).

A contextual bandit is a tuple $(\mathcal{X}, \mathcal{A}, R, \rho)$ where:

1. **Context space** $\mathcal{X} \subset \mathbb{R}^{d_x}$: Compact space of user-query features (see DEF-1.1.0, EQ-1.4)
2. **Action space** $\mathcal{A} = [-a_{\max}, +a_{\max}]^K \subset \mathbb{R}^K$: Compact set of bounded boost weights
3. **Reward function** $R : \mathcal{X} \times \mathcal{A} \times \Omega \to \mathbb{R}$: Measurable in ω, integrable (see EQ-1.2, Standing Assumption)
4. **Context distribution** ρ: Probability measure on $\mathcal{X}$ (the query/user arrival distribution)

This is a working definition. The measure-theoretic formalization (Borel σ-algebras on $\mathcal{X}$ and $\mathcal{A}$, probability kernel $P(\omega \mid x, a)$, measurable policy class) appears in Chapter 2.

At each round $t = 1, 2, ...$: - Observe context $x_t \sim \rho$ - Select action $a_t = \pi(x_t)$ (boost weights) - Rank products using score $s_i = r_{\text{ES}}(q, p_i) + a_t^\top \phi(p_i, u, q)$ - User interacts with ranking, generates outcome ω_t - Receive reward $R_t = R(x_t, a_t, \omega_t)$ - Update policy π

Objective: Maximize expected cumulative reward:

$$\max_{\pi} \mathbb{E}_{x\sim\rho,\omega}\left[\sum_{t=1}^{T} R(x_t, \pi(x_t), \omega_t)\right] \tag{1.7}$$

subject to constraints (1.3a-c).

Now the structure is clear: we make a **single decision** per context (choose boost weights), observe a **stochastic outcome** (user behavior), receive a **scalar reward**, and move to the next **independent context**. No sequential state transitions—that's what makes it a "bandit" rather than a full MDP.

1.4.3 The Value Function

Define the **value** of a policy π as:

$$V(\pi) = \mathbb{E}_{x\sim\rho}[Q(x, \pi(x))] \tag{1.8}$$

where $Q(x, a) = \mathbb{E}_{\omega}[R(x, a, \omega)]$ is the **expected reward** for context x and action a. The **optimal value** is:

$$V^* = \max_{\pi} V(\pi) = \mathbb{E}_{x\sim\rho}\left[\max_{a\in\mathcal{A}} Q(x, a)\right] \tag{1.9}$$

and the **optimal policy** is:

$$\pi^*(x) = \arg\max_{a\in\mathcal{A}} Q(x, a) \tag{1.10}$$

Key observation: If we knew $Q(x, a)$ for all (x, a), we'd simply evaluate it on a grid and pick the max. But Q is **unknown and expensive to estimate**—each evaluation requires a full search session with real users. This is the **exploration-exploitation tradeoff**:

- **Exploration**: Try diverse actions to learn $Q(x, a)$
- **Exploitation**: Use current Q estimate to maximize reward

1.4.4 Action Space Structure: Bounded Continuous

Unlike discrete bandits (finite arms), our action space $\mathcal{A} = [-a_{\max}, +a_{\max}]^K$ is **continuous and bounded**. This introduces both challenges and opportunities:

Challenges: - Cannot enumerate all actions - Need continuous optimization (gradient-based or derivative-free) - Exploration is harder (infinite actions to try)

Opportunities: - Smoothness: Nearby actions have similar rewards (we hope!) - Function approximation: Learn $Q(x, a)$ as a neural network - Gradient information: If Q is differentiable in a, use $\nabla_a Q$ to find $\arg\max$

Bounded actions are critical: Without bounds, the RL agent could set $w_{\text{discount}} = 10^6$ (destroying relevance) or $w_{\text{margin}} = -10^6$ (promoting loss-leaders indefinitely). Bounds enforce **safety**:

$$|a_k| \le a_{\max} \quad \forall k \in \{1, \ldots, K\} \tag{1.11}$$

Typical range: $a_{\max} \in [0.3, 1.0]$ (determined by domain experts).

1.4.5 Implementation: Bounded Action Space

The key operation is **clipping** uncalibrated policy outputs to the bounded space $\mathcal{A}$:

```python
import numpy as np

# Project action onto A = [-a_max, +a_max]^K (full class: Lab 1.7)
def clip_action(a, a_max=0.5):
    """Enforce #EQ-1.11 bounds. Critical for safety."""
    return np.clip(a, -a_max, a_max)

# Neural policy might output unbounded values
a_bad = np.array([1.2, -0.3, 0.8, -1.5, 0.4])
a_safe = clip_action(a_bad)  # -> [0.5, -0.3, 0.5, -0.5, 0.4]
```

Action	Before	After Clipping
a_1	1.2	0.5
a_2	-0.3	-0.3
a_3	0.8	0.5
a_4	-1.5	-0.5
a_5	0.4	0.4

Key takeaway: Always **clip actions before applying** them to the scoring function. Neural policies can output unbounded values; we must project them onto $\mathcal{A}$. Align `a_max` with `SimulatorConfig.action.a_max` in `zoosim/core/config.py` to ensure consistency. See **Lab 1.7** for the full `ActionSpace` class with sampling, validation, and volume computation.

Code $\leftrightarrow$ Env (clipping)

The production simulator (`MOD-zoosim.env`) enforces (1.4.4)1 action space bounds at ranking time.

- Action clipping: `np.clip(..., -a_max, +a_max)` in `zoosim/envs/search_env.py:50`
- Bound parameter: `SimulatorConfig.action.a_max` in `zoosim/core/config.py:229`
- Feature standardization toggle: `standardize_features` in `zoosim/core/config.py:231` (applied in env when enabled)

Keeping examples consistent with these guards avoids silent discrepancies between notebooks and the simulator.

1.4.6 Minimal End-to-End Check: One Step in the Simulator

Tie the concepts together by running a single simulated step with a bounded action.

```python
import numpy as np
from zoosim.core import config
from zoosim.envs import ZooplusSearchEnv

cfg = config.load_default_config()  # uses SimulatorConfig.seed at `zoosim/core/config.py:2
env = ZooplusSearchEnv(cfg, seed=cfg.seed)
```

```
state = env.reset()

# Zero action of correct dimensionality; env will clip if needed (see `zoosim/envs/search_e
action = np.zeros(cfg.action.feature_dim, dtype=float)
_, reward, done, info = env.step(action)

print(f"Reward: {reward:.3f}, done={done}")
print("Top-k ranking indices:", info["ranking"])  # shape aligns with `SimulatorConfig.top_
```

This verifies the scoring path: base relevance + bounded boosts → ranking → behavior simulation → reward aggregation.

Output:

```
Reward: 0.100, done=True
Top-k ranking indices: [5458, 4421, 1512, 9366, 3414, 8953, 4523, 4260, 4520, 4831]
```

Using the Gym Wrapper For RL loops and baselines, use the Gymnasium wrapper which exposes standard `reset/step` and action/observation spaces consistent with configuration.

```
import numpy as np
from zoosim.core import config
from zoosim.envs import GymZooplusEnv

cfg = config.load_default_config()
env = GymZooplusEnv(cfg, seed=cfg.seed)

obs, info = env.reset()
print("obs dim:", obs.shape)  # |categories| + |query_types| + |segments|

# Sample a valid bounded action (env clips incoming actions internally as well)
action = env.action_space.sample()
obs2, reward, terminated, truncated, info2 = env.step(action)

print(f"reward={reward:.3f}, terminated={terminated}, truncated={truncated}")
```

This interface is used in tests and ensures actions stay within $[-a_{\max}, +a_{\max}]^K$ with observation encoding derived from configuration.

Output:

```
obs dim: (11,)
reward= 0.100, terminated=True, truncated=False
```

1.5 1.5 From Optimization to Learning: Why RL?

At this point, one might ask: **Why not just optimize equation (1.6) directly?** If we can evaluate $R(a, x)$ for any (a, x), can we not use gradient descent?

1.5.1 The Sample Complexity Bottleneck

Problem: Evaluating $R(a, x)$ requires **running a live search session**: 1. Apply boost weights a to score products 2. Show ranked results to user 3. Wait for clicks/purchases 4. Compute $R = \alpha \cdot \text{GMV} + ...$

This is **expensive and risky**: - **Expensive**: Each evaluation takes seconds (user interaction) and costs money (potential lost sales) - **Risky**: Trying bad actions (a) can hurt user experience and revenue - **Noisy**: User behavior is stochastic—one sample has high variance

Sample complexity estimate: Suppose we have $|\mathcal{X}| = 100$ contexts (user segments $\times$ query types), $|\mathcal{A}| = 10^5$ discretized actions (gridding $K = 5$ boost features into 10 bins each), and need $G = 10$ gradient samples per action to estimate $\nabla_a R$ with low variance.

Naive grid search: Evaluate $R(x, a)$ for all (x, a) pairs: - Cost: $|\mathcal{X}| \cdot |\mathcal{A}| = 100 \cdot 10^5 = 10^7$ search sessions - At 1 session/second, this takes **116 days**

Gradient descent: Estimate $\nabla_a R$ for one context via finite differences: - Cost per iteration: $2K \cdot G = 2 \cdot 5 \cdot 10 = 100$ sessions (forward differences in K dimensions, G samples each) - For $T = 1000$ iterations to converge: $100 \cdot 1000 = 10^5$ sessions per context - Total: $|\mathcal{X}| \cdot 10^5 = 100 \cdot 10^5 = 10^7$ sessions (same as grid search!)

RL with exploration: Learn $Q(x, a)$ via bandits with $\sim \sqrt{T}$ regret: - Cost: $T \sim 10^4$ sessions total (across all contexts, amortized) - Wallclock: **3 hours** at 1 session/second

This **1000x speedup** is why we use RL for search ranking.

Gradient-based optimization would require: - Thousands of evaluations per context x - Directional derivatives $\nabla_a R(a, x)$ via finite differences - No safety guarantees (could try catastrophically bad a)

This is **not feasible** in production.

1.5.2 RL as Sample-Efficient, Safe Exploration

Reinforcement learning provides:

1. **Off-policy learning**: Train on historical data (past search logs) without deploying new policies
2. **Exploration strategies**: Principled methods (UCB, Thompson Sampling) that balance exploration vs. exploitation
3. **Safety constraints**: Enforce bounds (1.11) and constraints (1.3a-c) during learning
4. **Function approximation**: Learn $Q(x, a)$ or $\pi(x)$ as neural networks, generalizing across contexts
5. **Continual learning**: Adapt to distribution shift (seasonality, new products) via online updates

The RL framework transforms our problem from: - **Black-box optimization** (expensive, unsafe, no generalization)

to: - **Function learning with feedback** (sample-efficient, safe, generalizes)

1.6 1.6 Roadmap: From Bandits to Deep RL

Chapter 0 provided an informal, code-first toy example; Chapters 1–3 now build the mathematical foundations that justify and generalize it. This section provides a **roadmap through the book** (the 4-part structure and what each chapter accomplishes). For the **roadmap through this chapter** specifically, see the section headers below.

1.6.1 Part I: Foundations (Chapters 1-3)

We've established the business problem and contextual bandit formulation (Chapter 1). To evaluate policies safely without online experiments, we need **measure-theoretic foundations** for off-policy evaluation (Chapter 2: absolute continuity, Radon-Nikodym derivatives). To extend beyond bandits to multi-step sessions, we need **Bellman operators and convergence theory** (Chapter 3: contractions, fixed points).

Chapter 1 (this chapter): Formulate search ranking as contextual bandit **Chapter 2**: Probability, measure theory, and click models (position bias, abandonment) **Chapter 3**: Operators and contractions (Bellman equation, convergence)

1.6.2 Part II: Simulator (Chapters 4-5)

Before implementing RL algorithms, we need a **realistic environment** to test them. Chapters 4-5 build a production-quality simulator with synthetic catalogs, users, queries, and behavior models. This enables safe offline experimentation before deploying to real search traffic.

Chapter 4: Catalog, users, queries—generative models for realistic environments **Chapter 5**: Position bias and counterfactuals—why we need off-policy evaluation (OPE)

1.6.3 Part III: Policies (Chapters 6-8)

With a simulator, we can now develop **algorithms**: discrete template bandits (Chapter 6), continuous action Q-learning (Chapter 7), and policy gradients (Chapter 8).

Constraint enforcement for CM2 floors and rank stability appears in Chapter 10 (Guardrails), with the underlying duality theory in Appendix C.

Chapter 6 develops bandits with formal regret bounds; Chapter 7 establishes convergence under realizability (but no regret guarantees for continuous actions); Chapter 8 proves the Policy Gradient Theorem and analyzes the theory-practice gap. All three provide PyTorch implementations.

Chapter 6: Discrete template bandits (LinUCB, Thompson Sampling over fixed strategies) **Chapter 7**: Continuous actions via $Q(x, a)$ regression (neural Q-functions) **Chapter 8**: Policy gradient methods (REINFORCE, PPO, theory-practice gap)

1.6.4 Part IV: Evaluation, Robustness & Multi-Episode MDPs (Chapters 9-11)

Before production deployment, we need **safety guarantees**: off-policy evaluation to test policies on historical data (Chapter 9), robustness checks and guardrails with production monitoring (Chapter 10), and the extension to multi-episode dynamics where user engagement compounds across sessions (Chapter 11).

Chapter 9: Off-policy evaluation (IPS, SNIPS, DR—how to test policies safely) **Chapter 10**: Robustness and guardrails (drift detection, rank stability, CM2 floors, A/B testing, monitoring) **Chapter 11**: Multi-episode MDPs (inter-session retention, hazard modeling, long-term user value)

1.6.5 The Journey Ahead

By the end of this chapter, we will: - **Prove** convergence of bandit algorithms under general conditions - **Implement** production-quality deep RL agents (NumPy/PyTorch) - **Understand** when theory applies and when it breaks (the deadly triad, function approximation divergence) - **Deploy** RL systems safely (OPE, constraints, monitoring)

Let's begin.

How to read this chapter on first pass.

Sections 1.1–1.6, 1.9, and 1.10 form the core path: they set up search as a constrained contextual bandit and explain why we use RL rather than static tuning. Sections 1.7–1.8 are advanced/optional previews of measure-theoretic foundations, the Bellman operator, and off-policy evaluation. Skim or skip these on first reading and return after Chapters 2–3.

1.7 (Advanced) Optimization Under Uncertainty and Off-Policy Evaluation

This section is optional on a first reading.

Readers primarily interested in the **contextual bandit formulation** and motivation for RL can safely skim this section and jump to §1.9 (Constraints) or Chapter 2. Here we take an early, slightly more formal look at:

* **Expected reward and well-posedness** (why the expectations we write down actually exist) * **Off-policy evaluation (OPE)** at a *conceptual* level * Two preview results: **existence of an optimal policy** and a **regret lower bound**

The full measure-theoretic machinery (Radon–Nikodym, conditional expectation) lives in **Chapter 2**. The full OPE toolbox (IPS, SNIPS, DR, FQE, SWITCH, MAGIC) lives in **Chapter 9**.

1.6.6 1.7.1 Expected Utility and Well-Defined Rewards

Up to now we've treated expressions like $\mathbb{E}[R(\mathbf{w}, x, \omega)]$ as if they were obviously meaningful. Let's make that explicit.

Recall the stochastic reward from section 1.2:

- Context $x \in \mathcal{X}$ (user, query, etc.)
- Action $a \in \mathcal{A}$ (boost weights $\mathbf{w}$)
- Outcome $\omega \in \Omega$ (user behavior: clicks, purchases, abandonment)
- Reward $R(x, a, \omega) \in \mathbb{R}$ (scalarized GMV/CM2/STRAT/CLICKS)

We define the **expected utility** (Q-function) of action a in context x as

$$Q(x,a) := \mathbb{E}_{\omega \sim P(\cdot|x,a)}[R(x,a,\omega)]. \tag{1.12}$$

Here $P(\cdot \mid x,a)$ is the outcome distribution induced by showing the ranking determined by (x,a) in our click model.

To even *define* $Q(x,a)$, we need basic regularity conditions. These are made rigorous in Chapter 2 (Assumption 2.6.1); we preview them informally here:

Regularity conditions for well-defined rewards (informal):

1. **Measurability**: $R(x,a,\omega)$ is measurable as a function of ω.
2. **Integrability**: Rewards have finite expectation: $\mathbb{E}[|R(x,a,\omega)|] < \infty$.
3. **Coverage / overlap**: If the evaluation policy ever plays an action in a context, the logging policy must have taken that action with positive probability there. This is **absolute continuity** $\pi_{\text{eval}} \ll \pi_{\log}$, ensuring importance weights $\pi_{\text{eval}}(a \mid x)/\pi_{\log}(a \mid x)$ are finite.

Conditions (1)–(2) ensure $Q(x,a) = \mathbb{E}[R(x,a,\omega) \mid x,a]$ is a well-defined finite Lebesgue integral. Condition (3) is critical for **off-policy evaluation** (§1.7.2 below): when we estimate the value of a new policy using data from an old policy, we reweight observations by likelihood ratios. Absolute continuity guarantees these ratios exist (the denominator is never zero where the numerator is positive).

Continuous-action remark. Our action space $\mathcal{A} = [-a_{\max}, +a_{\max}]^K$ is continuous. In this case, $\pi_e(a \mid x)$ and $\pi_b(a \mid x)$ should be read as *densities* (Radon–Nikodym derivatives) with respect to Lebesgue measure on $\mathcal{A}$, and importance weights are density ratios. The coverage condition becomes a support condition: $\text{supp}(\pi_e(\cdot \mid x)) \subseteq \text{supp}(\pi_b(\cdot \mid x))$.

Chapter 2, §2.6 formalizes these conditions as **Assumption 2.6.1 (OPE Probability Conditions)** and proves that the IPS estimator is unbiased under these assumptions. For now, note that our search setting satisfies all three: rewards are bounded (GMV and CM2 are finite), and we'll use exploration policies (e.g., ε-greedy with $\varepsilon > 0$) that ensure coverage.

1.6.7 1.7.2 The Offline Evaluation Problem (Toy Cat-Food Example)

In §1.4 we defined the value of a policy π as

$$V(\pi) = \mathbb{E}_{x \sim \rho,\, \omega}[R(x, \pi(x), \omega)], \tag{1.7a}$$

where ρ is the context distribution (query/user stream).

So far we implicitly assumed we can just *deploy* any candidate policy π to estimate $V(\pi)$ online:

1. Pick boost weights $a = \pi(x)$.
2. Show ranking to users.
3. Observe reward $R(x,a,\omega)$.
4. Average over many sessions.

In a real search system, this is often **too risky**:

- Every exploratory policy hits **GMV** and **CM2**.
- It affects **real users** and competes with other experiments.
- It may violate **constraints** (CM2 floor, rank stability, strategic exposure).

This is where **off-policy evaluation (OPE)** enters:

> Can we estimate $V(\pi_e)$ for a new evaluation policy π_e using only logs collected under an old behavior policy π_b?

Formally, suppose we have a log

$$\mathcal{D} = (x_i, a_i, r_i)_{i=1}^n,$$

where $x_i \sim \rho$, $a_i \sim \pi_b(\cdot \mid x_i)$, and $r_i = R(x_i, a_i, \omega_i)$.
A **naïve idea** is to just average rewards in the logs:

$$\hat{V}_{\text{naive}} := \frac{1}{n} \sum_{i=1}^{n} r_i.$$

This clearly estimates $V(\pi_b)$, not $V(\pi_e)$.

Toy example: two cat-food templates Take a single query type "cat food" and two ranking templates:

- a_{GMV} — aggressive discount boosts (high GMV, risky CM2),
- a_{SAFE} — conservative boosts (lower GMV, safer CM2).

Consider:

- Logging policy π_b: always uses a_{SAFE},
- Evaluation policy π_e: would always use a_{GMV}.

The log contains n sessions:

$$\mathcal{D} = (x_i, a_i, r_i)_{i=1}^n, \quad a_i \equiv a_{\text{SAFE}}.$$

The empirical average

$$\hat{V}_{\text{naive}} = \frac{1}{n} \sum_{i=1}^{n} r_i$$

tells us how good $a_{\textbf{SAFE}}$ is. It tells us **nothing** about a_{GMV}, because that action was never taken: there are *no facts in the data* about what would have happened under a_{GMV}.

> **Key lesson:** OPE cannot recover counterfactuals from **purely deterministic logging** that never explores the actions of interest.

We need a way to reuse logs from π_b while "pretending" they came from π_e. That is exactly what **importance sampling** does.

A tiny NumPy sketch makes this concrete:

```
import numpy as np

# Logged under pi_b: always choose SAFE (action 0)
logged_actions = np.array([0, 0, 0, 0, 0])
logged_rewards = np.array([0.8, 1.1, 0.9, 1.0, 1.2])  # CM2-safe template

# New policy pi_e: always choose GMV-heavy (action 1)
def pi_b(a):
```

```
    return 1.0 if a == 0 else 0.0  # deterministic SAFE
def pi_e(a):
    return 1.0 if a == 1 else 0.0  # deterministic GMV

naive = logged_rewards.mean()
print(f"Naive log average: {naive:.3f}  (this is V(pi_b), not V(pi_e))")
```

There is *no* way to estimate what would have happened under action 1 from this dataset: the required probabilities and rewards simply do not appear.

1.6.8 1.7.3 Importance Sampling at a High Level

To estimate $V(\pi_e)$ from data generated under π_b, we reweight logged samples by how much more (or less) likely they would be under π_e.

For a logged triplet (x, a, r), define the **importance weight**

$$w(x, a) = \frac{\pi_e(a \mid x)}{\pi_b(a \mid x)}. \tag{1.7b}$$

Intuition:

- If $\pi_e(a \mid x) > \pi_b(a \mid x)$, then $w > 1$: this sample should count **more**, because π_e would have produced it more often.
- If $\pi_e(a \mid x) < \pi_b(a \mid x)$, then $w < 1$: it should count **less**.

The **inverse propensity scoring (IPS)** estimator for the value of π_e is

$$\hat{V}_{\mathrm{IPS}}(\pi_e) = \frac{1}{n} \sum_{i=1}^{n} w(x_i, a_i) \cdot r_i = \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_e(a_i \mid x_i)}{\pi_b(a_i \mid x_i)} \cdot r_i. \tag{1.7c}$$

Under the regularity conditions (measurability, integrability) and an additional **coverage** condition (next subsection), IPS is **unbiased**: in expectation, it recovers the true value $V(\pi_e)$ from data logged under π_b. Chapter 2, §2.6 formalizes these as **Assumption 2.6.1** and proves unbiasedness rigorously.

We will:

- Prove the general change-of-measure identity behind (1.7c) in **Chapter 2** (Radon–Nikodym).
- Implement IPS, SNIPS, DR, FQE, and friends in **Chapter 9**, including variance and diagnostics.

For this chapter, the only thing to remember is:

OPE ≈ "reweight logged rewards by importance weights."

1.6.9 1.7.4 Coverage / Overlap and Logging Design

The formula (1.7b) only makes sense when the denominator is non-zero whenever the numerator is:

$$\pi_e(a \mid x) > 0 \quad \Rightarrow \quad \pi_b(a \mid x) > 0. \tag{1.7d}$$

This is the **coverage** or **overlap** condition: any action that π_e might take in context x must have been tried with *some* positive probability by π_b.

If $\pi_b(a \mid x) = 0$ but $\pi_e(a \mid x) > 0$, the weight $w(x, a)$ would be infinite. Informally:

> If the logging policy never took action a in context x, the data contains **no information** about that counterfactual.

For real systems, this translates into concrete requirements: avoid fully deterministic logging (use ε-greedy or mixture policies with $\varepsilon \in [0.01, 0.10]$), store propensities $\pi_b(a \mid x)$ alongside each interaction, and design logging with future evaluation policies in mind—if we plan to evaluate aggressive boost strategies later, we must explore them occasionally now.

Chapter 9, §9.5 develops these requirements into a full logging protocol with formal assumptions (common support, propensity tracking), diagnostics (effective sample size), and production implementation guidance. The key intuition: **if we never explore an action, we can never evaluate it offline**.

1.6.10 1.7.5 Preview: Existence of an Optimal Policy

In §1.4 we wrote

$$\pi^*(x) = \arg\max_{a \in \mathcal{A}} Q(x, a), \qquad V^* = \max_{\pi} V(\pi),$$

as if the maximizer always existed and was a nice measurable function. In continuous spaces, this is surprisingly non-trivial.

Roughly, we need:

- A **compact** and nicely behaved action space $\mathcal{A}$,
- A **measurable** and upper semicontinuous $Q(x, a)$,
- A bit of measure-theory to ensure the argmax can be chosen **measurably** in x.

Existence guarantee. Under mild topological conditions (compact action space, upper semicontinuous Q), a measurable optimal policy $\pi^*(x) = \arg\max_a Q(x, a)$ exists via measurable selection theorems—see **Chapter 2, §2.8.2 (Advanced: Measurable Selection)** for the Kuratowski–Ryll–Nardzewski theorem (Theorem 2.8.3). For our search setting—where $\mathcal{A} = [-a_{\max}, a_{\max}]^K$ is a compact box and scoring functions are continuous—this guarantees the optimization problem in §1.4 is well-posed: there exists a best policy π^*, and our learning algorithms will be judged by how close they get to it.

1.6.11 1.7.6 Preview: Regret and Fundamental Limits

The last concept we preview is **regret**—how far a learning algorithm falls short of the optimal policy over time.

For a fixed policy π and the optimal policy π^*, define the **instantaneous regret** at round t:

$$\text{regret}_t = Q(x_t, \pi^*(x_t)) - Q(x_t, \pi(x_t)), \tag{1.13}$$

and the **cumulative regret** over T rounds:

$$\text{Regret}_T = \sum_{t=1}^{T} \text{regret}_t. \tag{1.14}$$

We say an algorithm has **sublinear regret** if

$$\lim_{T\to\infty} \frac{\text{Regret}_T}{T} = 0, \tag{1.15}$$

i.e., average per-round regret goes to zero.

Information-theoretic lower bounds for stochastic bandits establish a fundamental limit on learning speed. **Theorem 6.0** (Minimax Lower Bound) in Chapter 6 states: for any learning algorithm and any time horizon T, there exists a K-armed bandit instance such that

$$\mathbb{E}[\text{Regret}_T] \geq c\sqrt{KT}$$

for a universal constant $c > 0$. No algorithm can do better than $\Omega(\sqrt{KT})$ regret uniformly over all bandit problems. We must pay at least this price to discover which arms are good. UCB and Thompson Sampling are "optimal" because they match this bound up to logarithmic factors. For contextual bandits, the lower bound becomes $\Omega(d\sqrt{T})$ where d is the feature dimension; see Chapter 6 and **Appendix D** for the complete treatment via Fano's inequality and the data processing inequality. Here, the message is simply:

> **There is a built-in price for exploration**, and even the best algorithm cannot beat it asymptotically.

1.6.12 1.7.7 Where the Real Math Lives

This section deliberately kept things at a **preview** level:

- We introduced **expected utility** $Q(x, a)$ and stated regularity conditions (measurability, integrability, coverage) to make expectations like (1.12) well-posed; these are formalized in Chapter 2 as Assumption 2.6.1.
- We sketched **off-policy evaluation** via importance sampling and stressed the coverage condition (1.7d).
- We previewed two structural results: existence of an optimal policy (discussed in §1.7.5, rigorous treatment in Ch2 §2.8.2 via measurable selection) and a fundamental regret lower bound ([THM-6.0] in Chapter 6, proof via Fano's inequality in Appendix D).

The full story is split across later chapters:

- **Chapter 2** builds the measure-theoretic foundation (probability spaces, conditional expectation, Radon–Nikodym), and proves the change-of-measure identities that justify importance weights.
- **Chapter 6** develops bandit algorithms (LinUCB, Thompson Sampling) and proves regret upper bounds that match this lower-bound rate up to logs.
- **Chapter 9** turns IPS into a full-blown OPE toolbox, with model-based and doubly-robust estimators, variance analysis, and production diagnostics.

For the rest of this chapter, only the high-level picture is needed:

> We treat search as a contextual bandit with a well-defined expected reward, we will sometimes need to evaluate policies **offline** via importance weights, and there are fundamental limits on how quickly any algorithm can learn.

1.7 1.8 (Advanced) Preview: Neural Q-Functions and Bellman Operators

How do we represent $Q(x, a)$ for high-dimensional $\mathcal{X}$ (user embeddings, query text) and continuous $\mathcal{A}$? Answer: **neural networks**.

Define a parametric Q-function:

$$Q_\theta(x, a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R} \tag{1.16}$$

where $\theta \in \mathbb{R}^p$ are neural network weights. We'll learn θ to approximate the true $Q(x, a)$ via **regression**:

$$\min_\theta \mathbb{E}_{(x,a,r)\sim\mathcal{D}} \left[(Q_\theta(x, a) - r)^2\right] \tag{1.17}$$

where $\mathcal{D}$ is a dataset of (x, a, r) triples from past search sessions.

If the number of contexts and actions were tiny, we could represent Q as a **table** $Q[x, a]$ and fit it directly by regression on observed rewards. Chapter 7 begins with such a tabular warm-up example before moving to neural networks that can handle high-dimensional $\mathcal{X}$ and continuous $\mathcal{A}$.

1.7.1 Preview: The Bellman Operator (Chapter 3)

We've focused on contextual bandits—single-step decision making where each episode terminates after one action. But what if we extended to **multi-step reinforcement learning (MDPs)**? This preview provides the vocabulary for Exercise 1.5 and sets up Chapter 3.

In an MDP, actions have consequences that ripple forward: today's ranking affects whether the user returns tomorrow, builds a cart over multiple sessions, or churns. The value function must account for **future rewards**, not just immediate payoff.

The Bellman equation for an MDP value function is:

$$V(x) = \max_a \left\{ R(x, a) + \gamma \mathbb{E}_{x'\sim P(\cdot|x,a)}[V(x')] \right\} \tag{1.18}$$

where: - $P(x'|x, a)$ is the **transition probability** to next state x' given current state x and action a - $\gamma \in [0, 1]$ is a **discount factor** (future rewards are worth less than immediate ones) - The expectation is over the stochastic next state x'

Compact notation: We can write this as the **Bellman operator** $\mathcal{T}$:

$$(\mathcal{T}V)(x) := \max_a \{R(x,a) + \gamma \mathbb{E}_{x'}[V(x')]\} \tag{1.19}$$

The operator $\mathcal{T}$ takes a value function $V : \mathcal{X} \to \mathbb{R}$ and produces a new value function $\mathcal{T}V$. The optimal value function V^* is the **fixed point** of $\mathcal{T}$:

$$V^* = \mathcal{T}V^* \quad \Leftrightarrow \quad V^*(x) = \max_a \{R(x,a) + \gamma \mathbb{E}_{x'}[V^*(x')]\} \tag{1.20}$$

How contextual bandits fit: In our single-step formulation, there is **no next state**—the episode ends after one search. Mathematically, this means $\gamma = 0$ (no future) or equivalently $P(x'|x,a) = \delta_{\text{terminal}}$ (deterministic transition to a terminal state with zero value). Then:

$$V(x) = \max_a \{R(x,a) + 0 \cdot \mathbb{E}[V(x')]\} = \max_a Q(x,a)$$

This is exactly equation (1.9)! **Contextual bandits are the $\gamma = 0$ special case of MDPs.**

Why the operator formulation matters: In Chapter 3, we'll prove that $\mathcal{T}$ is a **contraction mapping** in $\|\cdot\|_\infty$, which guarantees: 1. **Existence and uniqueness** of V^* (Banach fixed-point theorem) 2. **Convergence** of iterative algorithms: $V_{k+1} = \mathcal{T}V_k$ converges to V^* geometrically 3. **Robustness**: Small errors in R or P lead to small errors in V^*

For now, just absorb the vocabulary: **Bellman operator**, **fixed point**, **discount factor**. These are the building blocks of dynamic programming and RL theory.

Looking ahead: Chapter 11 extends our search problem to **multi-episode MDPs** where user retention and session dynamics create genuine state transitions. There, we'll need the full Bellman machinery. But for the MVP (Chapters 1-8), contextual bandits suffice.

1.8 1.9 Constraints and Safety: Beyond Reward Maximization

Real-world RL requires **constrained optimization**. Maximizing (1.2) alone can lead to: - **Negative CM2**: Promoting loss-leaders to boost GMV - **Ignoring strategic products**: Optimizing short-term revenue at the expense of long-term goals - **Rank instability**: Reordering the top-10 drastically between queries, confusing users

We enforce constraints via **Lagrangian methods** (Chapter 10, with theory in Appendix C) and **rank stability penalties**.

1.8.1 Lagrangian Formulation

Transform constrained problem:

$$\begin{aligned} \max_\pi \quad & \mathbb{E}[R(\pi(x))] \\ \text{s.t.} \quad & \mathbb{E}[\text{CM2}(\pi(x))] \geq \tau_{\text{CM2}} \\ & \mathbb{E}[\text{STRAT}(\pi(x))] \geq \tau_{\text{STRAT}} \end{aligned} \tag{1.21}$$

into unconstrained:

$$\max_\pi \min_{\lambda \geq 0} \mathcal{L}(\pi, \lambda) = \mathbb{E}[R(\pi(x))] + \lambda_1(\mathbb{E}[\text{CM2}] - \tau_{\text{CM2}}) + \lambda_2(\mathbb{E}[\text{STRAT}] - \tau_{\text{STRAT}}) \tag{1.22}$$

where $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2_+$ are Lagrange multipliers. This is a **saddle-point problem**: maximize over π, minimize over λ.

Theorem 1.9.1 (Slater's Condition, informal). If there exists at least one policy that strictly satisfies all constraints (e.g., a policy with CM2 and exposure above the required floors and acceptable rank stability), then the **Lagrangian saddle-point problem**

$$\min_{\lambda \geq 0} \max_{\pi} \mathcal{L}(\pi, \lambda)$$

is equivalent to the original constrained optimization problem: they have the same optimal value.

Interpretation. Under mild convexity assumptions, we can treat Lagrange multipliers λ as "prices" for violating constraints and search for a saddle point instead of solving the constrained problem directly. **Appendix C** proves this rigorously (Theorem C.2.1) for the contextual bandit setting using the theory of randomized policies and convex duality ([@boyd:convex_optimization:2004, Section 5.2.3]). Chapter 10 exploits this to implement **primal–dual RL** for search: we update the policy parameters to increase reward and constraint satisfaction (primal step) while adapting multipliers that penalize violations (dual step).

What this tells us:

Strong duality means we can solve the constrained problem (1.8.1)1 by solving the unconstrained Lagrangian #EQ-1.22—no duality gap. Practically, this justifies **primal-dual algorithms**: alternate between improving the policy (primal) and adjusting constraint penalties (dual), confident that convergence to the saddle point yields the constrained optimum.

The strict feasibility requirement ($\exists \tilde{\pi}$ with slack in the CM2 constraint) is typically easy to verify: the baseline production policy usually satisfies constraints with margin. If no such policy exists, the constraints may be infeasible—one is asking for profitability floors that no ranking can achieve. **Appendix C, §C.4.3** discusses diagnosing infeasible constraint configurations: diverging dual variables, Pareto frontiers below constraint thresholds, and ε-relaxation remedies.

Implementation preview: In **Chapter 10 (§10.4.2)**, we implement constraint-aware RL using primal-dual optimization (theory in **Appendix C, §C.5**): 1. **Primal step**: $\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{L}(\theta, \lambda)$ (improve policy toward higher reward and constraint satisfaction) 2. **Dual step**: $\lambda \leftarrow \max(0, \lambda - \eta' \nabla_\lambda \mathcal{L}(\theta, \lambda))$ (tighten constraints if violated, relax if satisfied)

The saddle-point (θ^*, λ^*) satisfies the Karush-Kuhn-Tucker (KKT) conditions for the constrained problem (1.8.1). For now, just note that **constraints require dual variables** λ—we're not just learning a policy, but also learning how to trade off GMV, CM2, and strategic exposure dynamically.

1.9 1.10 Summary and Looking Ahead

We've established the foundation:

What we have: - **Business problem**: Multi-objective search ranking with constraints - **Mathematical formulation**: Contextual bandit with $Q(x, a)$ to learn - **Action space**: Continuous bounded $\mathcal{A} = [-a_{\max}, +a_{\max}]^K$ - **Objective**: Maximize $\mathbb{E}[R]$ subject to CM2/exposure/stability constraints - **Regret limits (preview)**: Bandit algorithms incur a $\tilde{\Omega}(\sqrt{KT})$ exploration cost; later bandit chapters formalize this lower bound and show algorithms that match it up to logs - **Implementation**: Tabular Q-table (baseline), preview of neural Q-function - **OPE foundations (conceptual)**: Why absolute continuity and importance sampling matter for safe policy evaluation (full measure-theoretic treatment in Chapters 2 and 9)

What we need: - **Probability foundations** (Chapter 2): Measure theory for OPE reweighting; position bias models (PBM/DBN) for realistic user simulation; counterfactual reasoning to test "what if?" scenarios safely - **Convergence theory** (Chapter 3): Bellman

operators, contraction mappings, fixed-point theorems for proving algorithm correctness - **Simulator** (Chapters 4-5): Realistic catalog/user/query/behavior models that mirror production search environments - **Algorithms** (Chapters 6-8): LinUCB, neural bandits, Lagrangian constraints for safe exploration and constrained optimization - **Evaluation** (Chapter 9): Off-policy evaluation (IPS, SNIPS, DR) for testing policies before deployment - **Deployment** (Chapters 10-11): Robustness, A/B testing, production ops for real-world systems

> **For Readers with Control Theory Background.** Readers familiar with LQR, HJB equations, or optimal control will find **Appendix B** provides a detailed bridge to RL: we show how the discrete Bellman equation arises as a discretization of HJB, how policy gradients relate to Riccati solutions, and how Lyapunov analysis informs convergence proofs. That appendix also traces the lineage from classical control to modern deep RL algorithms (DDPG, PPO, SAC). Readers new to control theory may skip it for now and return when these connections appear in Chapters 8, 10, and 11.

1.9.1 Why Chapter 2 Comes Next

We've formulated search ranking as contextual bandits, but left two critical gaps unresolved:

1. **User behavior is a black box.** Section 1.3's illustrative click model (position bias = 1/k) was helpful pedagogically, but production search requires **rigorous click models** that capture examination, clicks, purchases, and abandonment. We need to formalize "How do users interact with rankings?" at the level of **probability measures and stopping times**, not heuristics. Without this, our simulator won't reflect real user behavior, and algorithms trained in simulation will fail in production.

2. **We can't afford online-only learning.** Evaluating each policy candidate with real users (Section 1.5's "sample complexity bottleneck") is too expensive and risky. We need **off-policy evaluation (OPE)** to test policies on historical data logged under old policies. But OPE requires reweighting probabilities across different policies (importance sampling)—the weights $w(x, a) = \pi_{\text{eval}}(a|x)/\pi_{\log}(a|x)$ are only well-defined when both policies are absolutely continuous w.r.t. a common measure (the **coverage condition** in **Assumption 2.6.1** of Chapter 2, §2.6). This is **measure theory**, and it's not optional.

Chapter 2 addresses both gaps: We'll build **position-biased click models (PBM/DBN)** that mirror real user behavior with examination, relevance-dependent clicks, and session abandonment. Then we'll develop the **measure-theoretic foundations** (Radon-Nikodym derivatives, change of measure, importance sampling) that make OPE sound. This is not abstract mathematics for its own sake—it's the **foundation of safe RL deployment**.

By the end of Chapter 2, we will be able to: - Simulate realistic user sessions with position bias and abandonment - Formalize "what would have happened if we'd shown a different ranking?" (counterfactuals) - Understand why naive off-policy estimates are biased and how to correct them

Let's build.

1.10 Exercises

Note. Readers who completed Chapter 0's toy bandit experiment should: (i) compare the regret curves from Exercise 0.3 to the $\tilde{\Omega}(\sqrt{KT})$ lower bound discussed in §1.7.6 (and revisited

in Chapter 6); (ii) restate the Chapter 0 environment in this chapter's notation by identifying $(\mathcal{X}, \mathcal{A}, \rho, R)$.

Companion files for Chapter 1: - Labs and tasks: `docs/book/ch01/exercises_labs.md` - Written solutions: `docs/book/ch01/ch01_lab_solutions.md` - Runnable reference implementation: `scripts/ch01/lab_solutions.py` - Regression tests for chapter snippets: `tests/ch01/test_reward_e`

Production Checklist (Chapter 1)

- **Seed deterministically**: `SimulatorConfig.seed` in `zoosim/core/config.py:252` and module-level RNGs.
- **Align action bounds**: `SimulatorConfig.action.a_max` in `zoosim/core/config.py:229`; examples should respect the same value.
- **Use config-driven weights**: `RewardConfig` for $(\alpha, \beta, \gamma, \delta)$; avoid hard-coded numbers.
- **Validate engagement weight**: Assert $\delta/\alpha \in [0.01, 0.10]$ in `zoosim/dynamics/reward.py:56` (see Section 1.2.1).
- **Monitor RPC**: Log $\text{RPC}_t = \sum \text{GMV}_i / \sum \text{CLICKS}_i$; alert if drops $> 10\%$ (clickbait detection).
- **Enforce constraints early**: CM2 and exposure floors via Lagrange multipliers (**Chapter 10, §10.4.2**; theory in **Appendix C**).
- **Ensure reproducible ranking**: Enable `ActionConfig.standardize_features` in `zoosim/core/config.py:231`.

Exercise 1.1 (Reward Function Sensitivity). [20 min] (a) Implement equation (1.2) with $(\alpha, \beta, \gamma, \delta) = (1, 0, 0, 0)$ (GMV-only) and $(0.3, 0.6, 0.1, 0)$ (profit-focused). Generate 1000 random outcomes and plot the reward distributions. (b) Compute the correlation between GMV and CM2 in the simulated data. Are they aligned or conflicting? (c) Find business weights that make the two strategies from Section 1.2 achieve equal reward.

Exercise 1.2 (Action Space Geometry). [30 min] (a) For $K = 2$ and $a_{\max} = 1$, plot the action space $\mathcal{A}$ as a square $[-1, 1]^2$. (b) Sample 1000 random actions uniformly. How many are within the ℓ_2 ball $\|a\|_2 \leq 1$? (c) Modify `ActionSpace.sample()` to sample from the ℓ_∞ ball (current) vs. the ℓ_2 ball. Does this change the coverage of boost strategies?

Exercise 1.3 (Regret Bounds). [extended: 45 min] (a) Implement a naive **uniform exploration** policy that samples $a_t \sim \text{Uniform}(\mathcal{A})$ for T rounds. (b) Assume true $Q(x, a) = \mathbf{1}^\top x + \mathbf{1}^\top a + \epsilon$ where $\mathbf{1}^\top v := \sum_i v_i$ denotes the sum of components of vector v, and $\epsilon \sim \mathcal{N}(0, 0.1)$. Compute empirical regret Regret_T for $T = 100, 1000, 10000$. (c) Verify that $\text{Regret}_T/T \to \Delta$ where $\Delta = \max_a Q(x, a) - \mathbb{E}_a[Q(x, a)]$ (constant regret rate—suboptimal!). (d) **Challenge**: Implement ε-greedy (with $\varepsilon = 0.1$) and compare regret curves. Does it achieve sublinear regret?

Exercise 1.4 (Constraint Feasibility). [30 min] (a) Generate synthetic outcomes where CM2 is correlated with GMV: $\text{CM2} = 0.25 \cdot \text{GMV} + \text{noise}$. (b) Find the minimum CM2 floor τ_{CM2} such that $\geq 90\%$ of sampled actions satisfy the constraint. (c) Plot the **Pareto frontier**: GMV vs. CM2 for different action distributions. Is it convex?

Exercise 1.5 (Bellman Equation for Bandits). [20 min] Show that the contextual bandit value function (equation 1.9) satisfies:

$$V(x) = \max_a Q(x, a) = \max_a \mathbb{E}_\omega[R(x, a, \omega)]$$

Prove this is a special case of the Bellman optimality equation:

$$V(x) = \max_a \{R(x, a) + \gamma \mathbb{E}_{x'}[V(x')]\}$$

when $\gamma = 0$ (no future states). What happens if $\gamma > 0$?

Hint for MDP extension: In the MDP Bellman equation, the term $\gamma \mathbb{E}_{x'}[V(x')]$ represents expected future value starting from next state x' (sampled from transition dynamics $P(x' \mid x, a)$). For contextual bandits, there is no next state—the episode terminates after one action. Setting $\gamma = 0$ eliminates future rewards, reducing to the bandit case. When $\gamma > 0$, we get multi-step RL with inter-session dynamics (Chapter 11).

Next Chapter: We'll develop the **measure-theoretic foundations** needed for off-policy evaluation, position bias models, and counterfactual reasoning.