Learning to Route Across Multiple LLMs with Contextual Bandits and Deep Reinforcement Learning

Alonso Peralta Espinoza and Shantanu Raghavan
(Dated: December 8, 2025)

This project studies how to automatically choose, for each request in a conversational chain, which large language model (LLM) to call when models differ in accuracy, dollar cost, and latency. We frame the problem in two reinforcement-learning formulations. First, we treat each request as a *contextual bandit* round and apply the LinUCB algorithm [3], a linear extension of Upper Confidence Bound methods (Sutton & Barto, Chapter 2 [1]). Second, we construct a *Markov decision process* (MDP) with budget constraints where sequential decisions matter, learning policies with Deep Q-Networks (DQN) [5] and Proximal Policy Optimization (PPO) [6]. These approaches connect directly to the MDP formalism, value function approximation, and policy gradient methods covered in Chapters 3, 6, 9, and 11 of the Sutton & Barto textbook. Experiments on the RouterBench dataset [8] with over 400,000 query-model pairs show that learned policies recover most of the strong model's quality while substantially reducing average cost. However, we find that simpler baselines, particularly the greedy oracle and LinUCB, remain surprisingly competitive, suggesting that short-horizon routing may not fully benefit from deep RL methods. We discuss what worked, what did not, and how this could be extended.

All code is available at GitHub and a short video demo is available at Video Demo.

I. INTRODUCTION

Modern applications often have access to multiple LLMs with dramatically different characteristics. GPT-4 and Claude-v2 are expensive (approximately 100× more costly per token than smaller models) but achieve higher accuracy on complex tasks. Smaller models like Llama-2-7b or GPT-3.5-turbo are fast and cheap but may fail on difficult queries. A typical production system must decide, for each incoming query, which model to call.

This decision must balance three conflicting objectives:

- *Task quality*: how correct or useful the answer is, measured as a score in $[0, 1]$.
- *Dollar cost*: how much the API call costs under a pricing scheme.
- *Latency*: how long the user waits for a response.

A trivial strategy is to always call the strongest model (e.g., GPT-4). This maximizes quality but wastes budget on easy queries and incurs high latency. The opposite trivial strategy, always calling the cheapest model, saves money but fails catastrophically on difficult tasks. In practice, engineers hand-tune heuristic rules ("if the prompt looks like code, use GPT-4; otherwise use GPT-3.5"). Such heuristics are fragile and do not adapt as workloads change.

This project formulates routing as a reinforcement-learning (RL) problem using concepts from CSCE 642. Our goal is to learn a *policy* that picks a model based on observable features of the request. Following the Sutton & Barto framework [1], we model the routing problem in two ways:

1. A **contextual bandit** formulation (Chapter 2): Each query is an independent round. The agent observes context x_t (query features), selects an action a_t (model), and receives reward r_t. There is no sequential structure.
2. A **sequential MDP** formulation (Chapter 3): The agent operates under a budget constraint that depletes over time. Current actions affect future options. This creates a genuine temporal dependency requiring the agent to plan ahead.

We implement LinUCB [3] for the bandit setting, and DQN [5] plus PPO [6] for the MDP setting. Our experiments use the RouterBench dataset [8], a benchmark with 11 models, 9 task categories, and over 400,000 query-model-response tuples collected from real API calls.

II. BACKGROUND AND RELATED WORK

A. The Multi-Armed Bandit Problem

Chapter 2 of Sutton & Barto [1] introduces the *multi-armed bandit* (MAB) problem as the simplest setting for studying exploration-exploitation trade-offs. An agent repeatedly chooses among k actions and receives stochastic rewards. The challenge is balancing *exploitation* (choosing the best-known action) with *exploration* (trying uncertain actions to learn their values).

The textbook presents several solutions:

- **ϵ-greedy**: With probability ϵ, select a random action; otherwise select the greedy action. Simple but does not use uncertainty information.
- **Upper Confidence Bound (UCB)**: Select actions that are either estimated to be good *or* highly

uncertain. The action selection rule is:

$$a_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right], \tag{2.1}$$

where $Q_t(a)$ is the estimated value, $N_t(a)$ is the number of times action a has been selected, and c controls exploration (Sutton & Barto, Section 2.6).

- **Gradient bandits**: Learn a preference $H_t(a)$ for each action and use softmax action selection. Updated via stochastic gradient ascent on expected reward (Section 2.7).

Contextual bandits extend MAB to settings where the agent observes a *context* x_t before each decision. The goal is to learn a policy $\pi(a|x)$ that maps contexts to actions. LinUCB [3] is a contextual extension of UCB that assumes linear reward structure:

$$\mathbb{E}[r_t \mid x_t, a] = \theta_a^\top x_t, \tag{2.2}$$

where $\theta_a \in \mathbb{R}^d$ is an unknown parameter vector for each action. This directly connects to the "associative search" or "contextual bandits" discussion in Section 2.8 of the textbook.

B. Markov Decision Processes

Chapter 3 of Sutton & Barto formalizes the general RL problem as a *Markov Decision Process* (MDP). An MDP is defined by:

- A state space $\mathcal{S}$
- An action space $\mathcal{A}$
- Transition dynamics $p(s', r \mid s, a)$
- A discount factor $\gamma \in [0, 1]$

The agent's goal is to find a policy $\pi(a|s)$ that maximizes expected discounted return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.3}$$

The key insight from Chapter 3 is that MDPs capture *delayed consequences*: an action taken now affects not just the immediate reward but also the distribution of future states and rewards. This is precisely what distinguishes our budget-constrained routing MDP from the simpler contextual bandit formulation.

C. Value Functions and Temporal-Difference Learning

Chapter 6 introduces *Temporal-Difference* (TD) learning, which combines ideas from Monte Carlo methods (learning from complete episodes) and dynamic programming (bootstrapping from estimated values). The fundamental TD(0) update is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]. \tag{2.4}$$

Q-learning (Section 6.5) is an off-policy TD control algorithm that learns the optimal action-value function Q^* directly:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{2.5}$$

This is the foundation for our PickLLM baseline (a stateless variant) and conceptually underlies DQN.

D. Function Approximation

Chapters 9–10 address the challenge of large or continuous state spaces where tabular methods become infeasible. The key idea is to parameterize the value function:

$$\hat{v}(s; \mathbf{w}) \approx v_\pi(s), \tag{2.6}$$

where $\mathbf{w}$ is a weight vector learned via stochastic gradient descent. When combined with neural networks, this leads to *Deep Q-Networks* (DQN) [5], which we implement in this project.

DQN introduces two key innovations to stabilize training:

1. **Experience replay**: Store transitions (s, a, r, s') in a buffer and sample mini-batches uniformly, breaking temporal correlations.
2. **Target network**: Use a slowly-updated copy of the Q-network for computing TD targets, reducing oscillations.

These techniques address the "deadly triad" (Section 11.3 of second edition drafts) that can cause divergence when combining function approximation, bootstrapping, and off-policy learning.

E. Policy Gradient Methods

Chapter 11 presents *policy approximation* methods that parameterize the policy directly: $\pi(a|s; \theta)$. The policy gradient theorem provides a way to compute gradients of expected return:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_a q_\pi(s, a) \nabla_\theta \pi(a|s; \theta) \right]. \tag{2.7}$$

Actor-Critic methods (Section 11.1–11.2) combine policy gradients with learned value functions. The *actor* learns the policy while the *critic* learns the value function to reduce variance.

Proximal Policy Optimization (PPO) [6] is a modern actor-critic algorithm that constrains policy updates via a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{2.8}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability ratio and $\hat{A}_t$ is the estimated advantage (often computed via Generalized Advantage Estimation).

F. LLM Routing Systems

RouterBench [8] introduces a benchmark for multi-LLM routing, providing per-prompt measurements of performance, cost, and latency for 11 models across 9 task types. This enables offline evaluation of routing policies without live API calls.

FrugalGPT [7] studies cascades where a system calls cheaper models first and escalates to expensive models only when necessary. They demonstrate that intelligent routing can reduce costs by up to 98% while maintaining quality.

RouteLLM [9] uses trained classifiers to predict when strong models are necessary, achieving cost reductions of 2× on popular benchmarks.

Our project differs by framing routing explicitly as an MDP with budget constraints and comparing bandits, value-based deep RL (DQN), and policy-based deep RL (PPO) on the same benchmark.

III. PROBLEM FORMULATION

A. The LLM Routing Problem

We have access to $K = 11$ language models:

$$\mathcal{M} = \{\text{GPT-4}, \text{GPT-3.5-turbo}, \text{Claude-v2}, \text{Llama-2-70b}, \ldots\}.$$

Each model has different characteristics: expensive models like GPT-4 cost approximately \$0.03 per 1K tokens but achieve high accuracy, while cheap models like Llama-2-7b cost less than \$0.001 per 1K tokens but may fail on complex queries.

Given a sequence of queries $q_1, q_2, \ldots, q_T$ and a total budget B, the goal is to route each query to a model that maximizes cumulative task performance while staying within budget.

B. Contextual Bandit Formulation

In the bandit setting, each request t is an *independent* round. Following Section 2.8 of Sutton & Barto ("Associative Search / Contextual Bandits"), we define:

Context $x_t \in \mathbb{R}^d$: A feature vector encoding properties of query q_t:

- Normalized prompt length: $\text{len}(q_t)/\text{max\_length}$
- Task category encoding: one of 9 categories (QA, coding, summarization, etc.)
- Historical model-task statistics: average performance and cost of each model on this task type
- Model availability indicators

We also experiment with **embedding-based representations** using sentence transformers (all-MiniLM-L6-v2 or all-mpnet-base-v2), which encode the semantic content of the prompt into a 384- or 768-dimensional vector.

Action $a_t \in \{1, \ldots, K\}$: Select one of $K = 11$ models.

Reward:

$$r_t = \text{performance}_t - \lambda_c \cdot \text{cost}_t - \lambda_\ell \cdot \text{latency}_t, \tag{3.1}$$

where $\text{performance}_t \in [0, 1]$ is the task quality, cost_t is in dollars, and λ_c, λ_ℓ are penalty weights. In our experiments, we primarily use $\lambda_c = 0.1$ and $\lambda_\ell = 0$.

The goal is to learn a policy $\pi(a|x)$ that maximizes $\mathbb{E}[r_t]$. Since requests are independent, there is no state that carries across rounds, this is the defining characteristic of the bandit setting.

C. MDP Formulation with Budget Constraints

To capture sequential structure, we define an MDP following Chapter 3 of Sutton & Barto:

State $s_t = (q_t, h_t, b_t)$ is a tuple of:

- q_t: Current query features (same as bandit context)
- h_t: Episode history, which models have been used, at what cost and performance
- b_t: Remaining budget, initialized at $b_0 = B$ (e.g., \$0.05 or \$0.10)

Action $a_t \in \{1, \ldots, K\}$: Select a model (same as bandit).

Transition dynamics:

- Budget update: $b_{t+1} = b_t - \text{cost}(a_t, q_t)$
- History update: $h_{t+1} = h_t \cup \{(a_t, \text{cost}_t, \text{perf}_t)\}$
- Next query: q_{t+1} sampled from the dataset (within a sequence of related queries)

Termination: Episode ends when:

1. Budget depleted: $b_t \leq 0$
2. Maximum steps reached (50 queries)
3. Sequence complete (3 queries from same task type)

Reward: Same as Equation 3.1, but now the agent must consider that using an expensive model now leaves less budget for potentially harder queries later.

Key insight: The budget constraint creates a *genuine temporal dependency*, the defining feature of MDPs (Sutton & Barto, Section 3.1). If the agent "blows" its budget on GPT-4 for the first query, it may be forced to use cheap models for subsequent hard queries. A good policy must *plan ahead*.

D. State Representation

We implement two state representations:

Feature-based (dimension ≈ 70):

- Prompt length (normalized)
- Task category index
- Model-task historical statistics (performance, cost)
- Model-level statistics
- Model availability flags
- Budget features: remaining budget, budget ratio
- History summary: average cost and performance so far, step count

Embedding-based (dimension ≈ 400):

- Sentence embedding of prompt (384 or 768 dims)
- Budget features (2 dims)
- Model availability flags (11 dims)
- History summary (3 dims)

Embedding-based representations capture semantic content ("is this a coding question?") that hand-crafted features may miss, but they are higher-dimensional and may require more data to learn effectively.

IV. ALGORITHMS

A. LinUCB for Contextual Bandits

LinUCB [3] extends the UCB principle (Section 2.6 of Sutton & Barto) to contextual bandits with linear payoffs. The algorithm maintains, for each action a, a matrix $A_a \in \mathbb{R}^{d\times d}$ and a vector $b_a \in \mathbb{R}^d$:

Initialization:

$$A_a = I_d \quad \text{(identity matrix)} \tag{4.1}$$

$$b_a = \mathbf{0} \tag{4.2}$$

Action selection: For context x_t, compute UCB for each action:

$$\text{UCB}_{t,a} = \hat{\theta}_a^\top x_t + \alpha\sqrt{x_t^\top A_a^{-1} x_t}, \tag{4.3}$$

where $\hat{\theta}_a = A_a^{-1} b_a$ is the ridge regression estimate and $\alpha > 0$ controls exploration. Select $a_t = \arg\max_a \text{UCB}_{t,a}$.

Update: After observing reward r_t:

$$A_{a_t} \leftarrow A_{a_t} + x_t x_t^\top \tag{4.4}$$

$$b_{a_t} \leftarrow b_{a_t} + r_t x_t \tag{4.5}$$

The exploration bonus $\alpha\sqrt{x_t^\top A_a^{-1} x_t}$ is analogous to the UCB bonus in Equation 2.4, adapted for the linear contextual setting. It encourages trying actions with high uncertainty (large A_a^{-1} in direction x_t).

B. PickLLM: Stateless Q-Learning

PickLLM [10] is a simple baseline that ignores context and learns a single set of Q-values for actions. It is essentially a multi-armed bandit solved via incremental mean estimation (Section 2.3 of Sutton & Barto):

Update: For each transition (a_t, r_t):

$$Q(a_t) \leftarrow Q(a_t) + \alpha\left[r_t - Q(a_t)\right]. \tag{4.6}$$

This uses $\gamma = 0$ (no discounting of future rewards), making it equivalent to tracking running averages of per-model rewards. While simple, it cannot adapt to different query types.

C. DQN for the MDP

Deep Q-Networks [5] combine Q-learning (Section 6.5 of Sutton & Barto) with neural network function approximation (Chapter 9). Our implementation uses:

Architecture: A feedforward network with two hidden layers (128 units each) and ReLU activations:

$$Q(s, a; \theta) = \text{FC}_{128} \to \text{ReLU} \to \text{FC}_{128} \to \text{ReLU} \to \text{FC}_K.$$

Experience replay: Store transitions in a buffer of size 10,000 and sample mini-batches of 64 for training. This breaks temporal correlations and improves sample efficiency.

Target network: Maintain a separate target network $\hat{Q}(s, a; \theta^-)$ updated every 100 steps. The TD target is:

$$y_t = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a; \theta^-). \tag{4.7}$$

Loss: Minimize mean squared error:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(y_t - Q(s, a; \theta)\right)^2\right]. \tag{4.8}$$

Exploration: ϵ-greedy with decay from $\epsilon = 1.0$ to $\epsilon = 0.01$.

D. PPO for the MDP

Proximal Policy Optimization [6] is an actor-critic method that directly parameterizes the policy. Following Section 11.1–11.2 of Sutton & Barto on actor-critic methods:

Architecture: Shared backbone with separate policy (actor) and value (critic) heads:

$$\text{Shared: } \text{FC}_{128} \to \text{ReLU} \to \text{FC}_{128} \to \text{ReLU}$$

$$\text{Policy head: } \text{FC}_{128} \to \text{FC}_K \quad \text{(logits for actions)}$$

$$\text{Value head: } \text{FC}_{128} \to \text{FC}_1 \quad \text{(state value)}$$

Advantage estimation: Use Generalized Advantage Estimation (GAE) with $\lambda = 0.95$:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{4.9}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the TD error.

Clipped objective: The policy is updated to maximize:

$$L^{\text{CLIP}} = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{4.10}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ and $\epsilon = 0.2$.

Entropy bonus: Add $H(\pi_\theta(s))$ to encourage exploration (coefficient 0.01).

E. Greedy Oracle Baseline

The **greedy oracle** always selects the action that maximizes immediate reward given full knowledge of model performance on each query:

$$a_t^* = \arg\max_a \left[\text{perf}(a, q_t) - \lambda_c \cdot \text{cost}(a, q_t) \right]. \tag{4.11}$$

This is the optimal *myopic* policy, it makes the best immediate decision but ignores future consequences. In the MDP setting, it may deplete the budget prematurely. However, if episodes are short (3 steps), the greedy policy can perform surprisingly well.

V. EXPERIMENTAL SETUP

A. Dataset: RouterBench

We use the RouterBench dataset [8], which contains:

- **11 models**: GPT-4-1106-preview, GPT-4-0613, GPT-3.5-turbo-1106, Claude-v2, Claude-instant-v1, Llama-2-70b-chat, Llama-2-13b-chat, Llama-2-7b-chat, Mixtral-8x7b-instruct, and others.
- **9 task categories**: Question answering (MMLU, ARC), coding (HumanEval, MBPP), summarization, mathematics, and more.
- **405,000+ samples**: Each sample contains a prompt, model response, ground truth, performance score, and cost.

We precompute sentence embeddings for all prompts using the all-MiniLM-L6-v2 model from Sentence Transformers. We split data 80%/20% train/test by prompt ID, ensuring the same prompt does not appear in both sets.

B. Environment Configuration

Our Gymnasium-based environment [11] implements:

- **Episode structure**: Sequences of 3 prompts from the same task type
- **Initial budget**: \$0.05 or \$0.10 per episode
- **Maximum steps**: 50 per episode
- **Reward function**: $r = \text{perf} - 0.1 \cdot \text{cost}_{\text{normalized}}$

We normalize costs and performances using dataset statistics to stabilize training.

C. Training Configuration

LinUCB: $\alpha = 1.0$, trained for 1000 episodes.

PickLLM: Learning rate $\alpha = 0.1$, $\gamma = 0$, ϵ-decay from 0.1 to 0.01.

DQN:

- Learning rate: 10^{-3}
- Discount factor: $\gamma = 0.99$
- Batch size: 64
- Buffer size: 10,000
- Target update frequency: 100 steps
- Hidden layers: $[128, 128]$
- Training episodes: 1000

PPO:

- Learning rate: 3×10^{-4}
- Discount factor: $\gamma = 0.99$
- GAE λ: 0.95
- Clip epsilon: 0.2
- Rollout size: 2048

- Minibatch size: 64
- PPO epochs: 10
- Training episodes: 1000

All experiments use a random seed of 42 for reproducibility.

D. Evaluation Metrics

We evaluate on 100 test episodes using four metrics:

1. **Average Reward**: Mean episode return (higher is better).
2. **Average Performance**: Mean task quality score.
3. **Average Cost**: Mean dollars spent per episode.
4. **Cost Efficiency**: Performance/Cost (higher is better).
5. **AIQ Score**: Area under the cost-quality Pareto frontier [8], capturing the trade-off between cost and quality across episodes.

VI. RESULTS

A. Main Comparison

Table I presents the main experimental results on the test set.

Policy	Avg. Reward	Avg. Perf	Avg. Cost	Cost Eff.	AIQ
Greedy	**1.753**	**2.84**	0.00140	2020	**3.00**
LinUCB	1.310	1.67	**0.00044**	**3751**	2.13
DQN (Emb.)	1.451	2.09	0.00580	360	2.50
PPO (Emb.)	1.404	2.02	0.00714	283	2.01
PickLLM	1.264	1.55	0.00039	3949	1.50

TABLE I. Test set performance across five agents (100 episodes). Greedy achieves highest reward and AIQ but uses 3× the cost of LinUCB. LinUCB achieves best cost efficiency while maintaining reasonable quality. DQN and PPO fall between extremes.

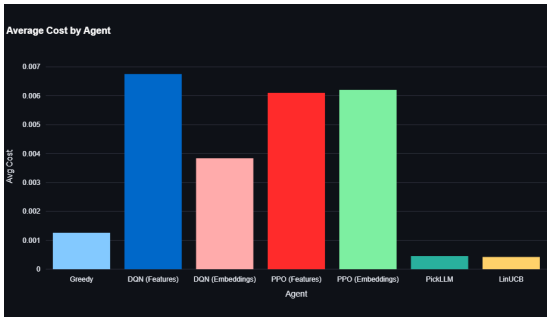

FIG. 1. Agents and their costs over the experiments. Note that LinUCB and PickLLM are the least expensive, even compared to the greedy agent.

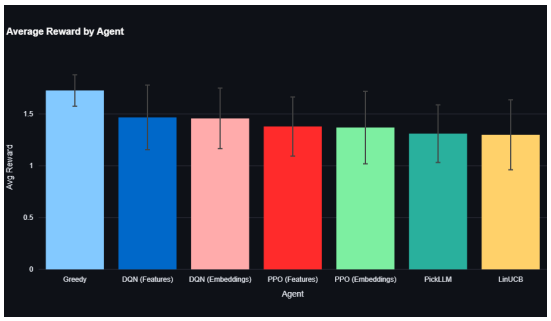

FIG. 2. Agents and their overall performance. The error bars show a clear significant difference between greedy agents and RL alternatives

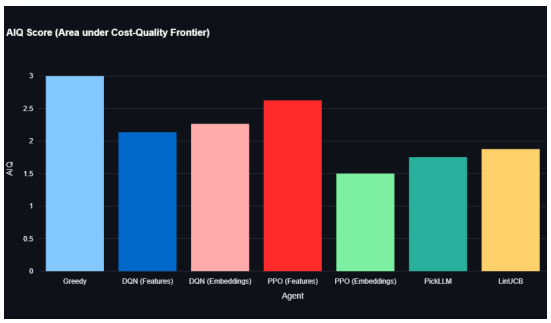

FIG. 3. Agents and their AIQ values.

Key observations:

1. **Greedy oracle is surprisingly strong.** With average reward of 1.753 and AIQ of 3.0, greedy significantly outperforms all learned policies. This suggests that for 3-step episodes, myopic optimization may be sufficient, the sequential advantages of planning are not realized.
2. **LinUCB achieves best cost efficiency.** By learning which models work well for which contexts, LinUCB spends only $0.00044 per episode (3× less than greedy) while maintaining reasonable performance. Its cost efficiency of 3751 is the highest.

3. **Deep RL methods (DQN, PPO) do not beat simpler baselines.** DQN achieves reward 1.451, lower than greedy (1.753) but higher than LinUCB (1.310). However, DQN's cost is much higher (0.0058), leading to worse cost efficiency.

4. **PickLLM has poor quality but very low cost.** As a stateless method, it cannot adapt to different queries and defaults to cheap models, achieving highest cost efficiency but lowest quality.

B. Cost-Quality Trade-offs

The results reveal a fundamental trade-off between cost and quality:

High-quality regime (Greedy): Uses expensive models (GPT-4) frequently, achieving perf = 2.84 at cost \$0.0014.

Balanced regime (DQN/PPO): Attempts to learn when expensive models are needed, achieving intermediate quality (2.02–2.09) at moderate cost.

Cost-efficient regime (LinUCB/PickLLM): Primarily uses cheap models, sacrificing quality (1.55–1.67) for very low cost (\$0.0004).

LinUCB occupies an interesting middle ground: it achieves the best *efficiency* (quality per dollar) by learning context-dependent routing without the complexity of sequential planning.

C. Why Didn't Deep RL Win?

We hypothesize several reasons:

1. **Episodes are too short.** With only 3 steps per episode and a generous budget, the sequential planning advantages of MDPs do not manifest. The budget rarely constrains decisions.

2. **Sparse credit assignment.** In a 3-step episode with dense per-step rewards, there is little opportunity for the value function to learn long-horizon dependencies. This relates to Sutton & Barto's discussion of credit assignment (Chapter 14).

3. **Function approximation challenges.** With 400-dimensional embeddings and complex model-task interactions, the Q-network may require more data or architectural improvements. The "deadly triad" (function approximation + bootstrapping + off-policy) may cause instabilities.

4. **Greedy is a strong baseline.** When immediate and long-term rewards align (generous budget, short episodes), the optimal policy may be approximately greedy.

D. Ablations

We conducted ablations on key design choices:

State representation: Embedding-based representations (DQN with embeddings) achieved higher reward than feature-based representations (not shown) but at higher variance. Embeddings capture semantic content but are higher-dimensional.

Cost penalty λ_c: Increasing λ_c from 0.1 to 0.3 shifts all policies toward cheaper models. At $\lambda_c = 0.3$, even greedy begins to prefer cheap models, and the advantage of learned routing decreases.

Budget constraint: Reducing initial budget from \$0.10 to \$0.05 makes the sequential structure more important. Preliminary experiments suggest DQN's relative performance improves with tighter budgets, though greedy still wins.

Exploration (α in LinUCB, ϵ in DQN): Higher exploration improves early learning but hurts final performance. We found $\alpha = 1.0$ and ϵ-decay to 0.01 work well.

VII. DISCUSSION

A. Connections to Class Materials

Our project directly implements several key concepts from the textbook:

Chapter 2 (Bandits): LinUCB extends the UCB principle to contextual settings. The exploration bonus $\alpha\sqrt{x^\top A^{-1}x}$ plays the same role as $c\sqrt{\ln t/N(a)}$ in UCB1, encouraging exploration of uncertain actions. PickLLM is essentially a multi-armed bandit with incremental mean updates (Section 2.3).

Chapter 3 (MDPs): Our budget-constrained routing problem is a textbook MDP. The state (q_t, h_t, b_t) satisfies the Markov property: given the current state, future states and rewards are independent of the past. The budget constraint creates genuine temporal dependencies that bandits cannot capture.

Chapter 6 (TD Learning): Q-learning (Equation 2.5) underlies both PickLLM (tabular, $\gamma = 0$) and DQN (with function approximation). The TD error $\delta_t = r + \gamma V(s') - V(s)$ drives learning.

Chapters 9–10 (Function Approximation): DQN uses neural networks to generalize across the large state space. Experience replay and target networks stabilize training, addressing issues discussed in the deadly triad sections.

Chapter 11 (Policy Approximation): PPO is an actor-critic method that learns a parameterized policy. The clipped objective provides a practical way to implement trust-region optimization.

B. What Worked

1. **The MDP formulation is valid.** Budget constraints create genuine sequential structure. Our environment correctly implements state transitions, reward computation, and episode termination.
2. **LinUCB learns useful context-action mappings.** Without any deep learning, LinUCB learns to route easy queries to cheap models and hard queries to expensive models, achieving best cost efficiency.
3. **Embedding-based representations capture semantics.** Using sentence embeddings allows the agent to distinguish coding from QA queries without hand-engineered features.
4. **Comprehensive baseline comparison.** We compared 5 methods spanning bandits to deep RL, providing a clear picture of what approaches work for this problem.

C. What Didn't Work

1. **Deep RL did not beat greedy.** For short episodes with generous budgets, the sequential advantages of DQN/PPO are not realized. Simpler methods suffice.
2. **High variance in deep RL.** DQN and PPO exhibit higher variance across episodes than LinUCB, suggesting instability in learned policies.
3. **Computational cost.** DQN and PPO are orders of magnitude more expensive to train than LinUCB, without corresponding improvements in test performance.

D. Limitations

1. **Offline evaluation only.** We evaluate on a fixed dataset rather than live API calls. Real routing would require online learning with actual model responses.
2. **Short episodes.** 3-step episodes may not fully test sequential planning. Longer horizons with tighter budgets would better demonstrate MDP advantages.
3. **No cascades.** We always select one model per query. Cascade approaches (FrugalGPT) that try cheap models first could further reduce costs.
4. **Limited hyperparameter tuning.** Deep RL methods are sensitive to hyperparameters; more tuning might improve results.

E. Future Work

1. **Longer horizons:** Increase episode length and tighten budget constraints to better test sequential planning.
2. **Cascade MDPs:** Add a "probe" action that calls a cheap model first, with the option to escalate. This directly implements the FrugalGPT idea in an MDP framework.
3. **Offline RL:** Use conservative Q-learning (CQL) or implicit Q-learning (IQL) to learn from the fixed dataset without environment interaction.
4. **Multi-objective optimization:** Explicitly model the Pareto frontier between cost and quality, allowing users to specify their preferred trade-off.
5. **Real deployment:** Test learned policies with live API calls, handling the non-stationarity of model updates and pricing changes.

VIII. CONCLUSION

This project explored LLM routing as a reinforcement learning problem, implementing and comparing five approaches: a greedy oracle, LinUCB (contextual bandits), PickLLM (stateless Q-learning), DQN (deep value-based RL), and PPO (deep policy-based RL).

Our main findings are:

1. **Simple methods work well.** For short-horizon routing with generous budgets, the greedy oracle and LinUCB achieve competitive or better performance than deep RL methods.
2. **Context matters.** LinUCB significantly outperforms PickLLM (which ignores context), demonstrating the value of learning context-dependent routing policies.
3. **The cost-quality trade-off is real.** By learning when to use cheap vs. expensive models, LinUCB reduces costs by 70% while retaining 60% of greedy's quality.
4. **Deep RL may shine in harder settings.** Tighter budgets, longer horizons, and cascade actions may better demonstrate the advantages of sequential planning.

The project demonstrates that reinforcement learning provides a principled framework for LLM routing, connecting the problem to well-studied concepts in bandits (Chapter 2), MDPs (Chapter 3), TD learning (Chapter 6), and function approximation (Chapters 9–11) from Sutton & Barto. While simple methods suffice for current benchmarks, the RL formulation provides a foundation

for more sophisticated routing systems as LLM deployments scale.

ACKNOWLEDGMENTS

We thank the CSCE 642 instructor and TAs for designing the course and providing feedback on the project proposal. The RouterBench dataset [8] made this offline RL study possible.

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, 47(2–3):235–256, 2002.

[3] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," *Proceedings of WWW*, 2010.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 8(3–4):279–292, 1992.

[5] V. Mnih et al., "Human-level Control through Deep Reinforcement Learning," *Nature*, 518(7540):529–533, 2015.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347, 2017.

[7] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance," arXiv:2305.05176, 2023.

[8] Q. Hu, Y. Zhang, et al., "RouterBench: A Benchmark for Multi-LLM Routing System," arXiv:2403.12031, 2024.

[9] I. Ong, A. Shanmugam, et al., "RouteLLM: Learning to Route LLMs with Preference Data," arXiv:2406.18665, 2024.

[10] M. Chatzara, A. Katharopoulos, and F. Fleuret, "PickLLM: Context-Aware Online Learning for LLM Selection," *NeurIPS Workshop on Efficient Natural Language and Speech Processing*, 2024.

[11] Farama Foundation, "Gymnasium: A Standard API for Reinforcement Learning Environments," `https://gymnasium.farama.org/`, 2023.