Learning to Learn: Curriculum Selection via Soft Q-Learning for Sample-Efficient Language Models

Scott Viteri Damir Vrabac

1 Overview

We propose an algorithm that enables language models to learn what data to learn from. Rather than training on randomly sampled data, the model takes actions that select training examples, receiving reward based on how well it currently predicts held-out data. A Q-function, parameterized as a small network reading the language model's internal representations, learns to predict the long-run value of each candidate. The policy is derived from the Q-function via a Boltzmann distribution, eliminating the need for a separate policy network. The core hypothesis is that this learned curriculum selection will yield significantly better sample efficiency than random or heuristic curricula.

2 Technical Approach

2.1 Setup

- Language model (GPT-2) with parameters θ, hidden dimension $d = 768$
- Streaming data source, yielding fresh batches of N candidate context windows each step
- Fixed held-out evaluation set D, large; a fresh random subset $\hat{D} \subset D$ of size M is sampled each step for reward computation
- Q-network Q_ϕ with parameters ϕ, implemented as a 2-layer MLP ($d \to 32 \to 1$) reading the base model's residual stream (see Section 2.3)
- Target Q-network $Q_{\bar{\phi}}$ with parameters $\bar{\phi}$, a Polyak-averaged copy of Q_ϕ used for stable bootstrap targets
- Boltzmann temperature $\beta > 0$ controlling exploration
- Learning rate α for the LM gradient step; learning rate η for the Q-network update
- Discount factor $\gamma \in (0, 1)$ controlling the effective planning horizon
- Polyak averaging rate $\tau \in (0, 1)$ for the target network update
- Gradient clipping threshold G (max global norm; applied to both LM and Q-network gradient steps)
- Optional: TD-into-theta flag and scaling coefficient λ_{TD} for representation shaping (Section 2.7)

2.2 State, Action, and Reward

The state s_k is the current model parameters θ_k, which determine the model's representations and therefore the features available to the Q-network. In practice, s_k is never represented explicitly—it is accessed implicitly through the base model's hidden states when candidates are forwarded through it (Section 2.3).

The action $a_k \in C_k$ is the choice of which training example to train on at step k, where C_k is the batch of N candidates available at that step. Since data is streamed, each step presents a fresh candidate batch, and each candidate is seen at most once. The Q-network generalises across batches via the base model's hidden-state representation.

The reward is the held-out log-probability *after* the training update:

$$r_k = \frac{1}{M} \log p_{\theta_{k+1}}(\hat{D}_k), \tag{1}$$

where $\hat{D}_k$ is a fresh random subset of D of size M, and θ_{k+1} results from training on a_k. This measures the model's absolute predictive fitness as a consequence of the selected action. Resampling $\hat{D}_k$ each step prevents the Q-function from overfitting to a fixed evaluation subset.

The Q-function optimises the discounted return $(1-\gamma)\sum_k \gamma^k r_k$. The weights $(1-\gamma)\gamma^k$ sum to one, so this is the *exponentially weighted average* of the reward sequence: the Q-value at any state equals the typical per-step reward expected over the next $\sim 1/(1-\gamma)$ steps under the current policy. Maximising this objective favours curricula that achieve high predictive fitness quickly—each step spent with poor predictions is penalised, with a horizon controlled by γ.

2.3 Q-Network Architecture

The Q-network reads the base model's internal representations to assess candidate value. For each candidate $x \in C_k$:

1. Forward x through the base model with parameters θ_k, extracting the last-layer hidden state at the final token position: $h_x = \texttt{last\_hidden}(\theta_k, x) \in \mathbb{R}^d$.
2. Detach h_x from the computation graph (no gradients flow into θ).
3. Pass through the Q-network:

$$Q_\phi(s_k, x) = W_2\, \sigma(W_1\, h_x + b_1) + b_2, \tag{2}$$

   where $\phi = \{W_1 \in \mathbb{R}^{32\times d},\ b_1 \in \mathbb{R}^{32},\ W_2 \in \mathbb{R}^{1\times 32},\ b_2 \in \mathbb{R}\}$ and σ is ReLU.

The hidden state h_x implicitly encodes both the model's current state (through θ_k, which determines the representations) and the identity of the candidate (through the input x). By default, h_x is detached from the computation graph, ensuring that θ is updated only by the language modeling objective while ϕ is updated only by the TD loss. This prevents the co-adaptation instability that arises when value gradients distort the base model's representations. Section 2.7 describes an optional mode that relaxes this separation.

2.4 Normalised Discounted Soft Bellman Equation

The Q-function satisfies a normalised discounted soft Bellman equation:

$$Q(s_k, a_k) = (1-\gamma)\, r_k + \gamma\, V_{\bar{\phi}}(s_{k+1}), \tag{3}$$

where $\gamma \in (0,1)$ is the discount factor and $V_{\bar{\phi}}$ is the normalised soft value computed from the target network (see below). The $(1-\gamma)$ factor on the reward ensures that a constant reward r contributes r (not $r/(1-\gamma)$) to the Q-value fixed point. Combined with the normalised soft value (which subtracts $\beta \log N$), a constant reward gives $Q^* = r$ exactly. This keeps Q-values on the same scale as the reward and the MLP output in a numerically comfortable range.

The soft value function is normalised against the uniform prior:

$$V(s) = \beta \log \frac{1}{N} \sum_{a'} \exp\big(Q(s,a')/\beta\big) = \beta \left[\log \sum_{a'} \exp\big(Q(s,a')/\beta\big) - \log N \right]. \tag{4}$$

The $\log N$ subtraction removes the entropy bonus from the uniform distribution: when all Q-values are equal at q, we have $V(s) = q$ rather than $V(s) = q + \beta \log N$. This is the *normalised* logsumexp, or equivalently the log-mean-exp. Without it, the entropy bonus $\beta \log N$ would be amplified by $\gamma/(1-\gamma)$ at the Bellman fixed point, inflating Q-values far above the reward scale.

With both normalisations—$(1-\gamma)$ on the reward and $-\log N$ in the soft value—a constant reward r gives $Q^* = r$ at the fixed point. Q-values live on the same scale as the reward (around -3 to -4 for GPT-2 perplexity), and the Q-network's output is directly interpretable as the expected per-step reward over the planning horizon. At convergence, $Q(s,a)$ represents a normalised discounted sum of future rewards from choosing a in state s and following the Boltzmann policy thereafter. With $\gamma = 0.9$, the effective horizon is ~ 10 steps.

The policy $\pi(a \mid s) \propto \exp(Q(s,a)/\beta)$ is unaffected by the $\log N$ subtraction, since softmax is shift-invariant.

Target network. To stabilise the bootstrap target $V(s_{k+1})$, we maintain a target network $Q_{\bar{\phi}}$ that is a Polyak-averaged copy of the online network Q_ϕ:

$$\bar{\phi} \;\leftarrow\; \tau\,\phi + (1-\tau)\,\bar{\phi}, \qquad \tau \in (0,1). \tag{5}$$

The soft value in the TD target is computed from $Q_{\bar{\phi}}$, not Q_ϕ. This breaks the feedback loop where the network's own output serves as its training target, preventing the oscillations and divergence that can occur with function approximation. The online network Q_ϕ is still used for action selection (the Boltzmann policy) and for the prediction $\hat{Q}$ in the TD loss. With $\tau = 0.01$, the target network tracks the online network with a lag of $\sim 1/\tau = 100$ steps.

2.5 Algorithm

The key implementation insight is that each step's forward pass of candidates through the base model serves two purposes: action selection for the *current* step and the bootstrap target for the *previous* step's Q update. This avoids a redundant second forward pass.

The LM trains on a *policy-weighted mixture* of all N candidates at every step, rather than on a single sampled action. Define the Boltzmann weights:

$$\pi_i = \frac{\exp(Q_\phi(s_k, x_i)/\beta)}{\sum_{j=1}^{N} \exp(Q_\phi(s_k, x_j)/\beta)}, \qquad x_i \in C_k. \tag{6}$$

The LM loss is the policy-weighted mixture:

$$\mathcal{L}_{\text{mix}}(\theta; C_k) = \sum_{i=1}^{N} \pi_i\, \ell(\theta, x_i), \tag{7}$$

where $\ell(\theta, x_i)$ is the per-token cross-entropy loss on candidate x_i. This is precisely the *expected gradient* under the Boltzmann policy:

$$\nabla_\theta \mathcal{L}_{\text{mix}} = \mathbb{E}_{x \sim \pi(\cdot \mid s_k)}\big[\nabla_\theta \ell(\theta_k, x)\big], \tag{8}$$

computed exactly rather than estimated via a single Monte Carlo sample. This eliminates the variance from action sampling.

Q-learning and the mixture LM update. The Q-network update is unchanged by the mixture formulation: a discrete action a_k is still sampled from the Boltzmann policy, its hidden state h_{prev} is stored, and the standard TD update is applied at the next step. The Q-function continues to satisfy the discounted soft Bellman equation (3). The only difference from a single-action formulation is how the environment *transitions*: the LM parameters are updated via the full mixture rather than a single example.

Off-policy interpretation. From the Q-learner's perspective, the situation is analogous to off-policy learning. The Q-learner observes: "I selected action a_k, the environment transitioned to state s_{k+1}, and I received reward r_k." It does not need to know that the transition was caused by a mixture-weighted gradient step rather than a single-example step. Q-learning is inherently off-policy—the Bellman backup $Q(s, a) = (1 - \gamma)\, r + \gamma\, V(s')$ is valid regardless of the behavior policy that generated the transition, as long as r and s' are the actual observed reward and next state.

Over many steps, the Q-function converges to predict the value of upweighting a given candidate within the mixture—which is exactly the marginal contribution signal needed to set the mixture weights well.

Effective batch size. The mixture-weighted update is equivalent to training on a soft batch of $\sim 1/\|\pi\|_2^2$ effective examples per step (the inverse of the collision probability of the policy distribution). Early in training, when Q-values are similar, this approaches N; as the policy sharpens, the effective batch size decreases toward 1.

Computational cost per step. One forward pass of N candidates through the base model (batched, no gradient) for Q-scoring; N forward-backward passes through the full model for the mixture LM loss (processed in mini-batches with gradient accumulation); one forward pass of M held-out examples for reward; one MLP forward-backward for the Q update; one MLP forward pass through the target network (negligible). The candidate extraction forward passes dominate; the mixture LM step adds roughly a $2\times$ factor to total step time compared to a single-example LM step.

Staleness. The Q update at step k uses the target network $Q_{\bar{\phi}}$ for the bootstrap value V_k, and the online network Q_ϕ for action selection. The target network lags the online network by $\sim 1/\tau$ steps, which is by design: this lag stabilises the TD target. The online Q-values q_k used for action selection have one-step staleness (computed before the ϕ update), which is standard in online Q-learning and negligible given the small MLP learning rate η.

2.6 Uniform Mixture Variant (No Q-Weighting)

As an ablation, the flag `--no_q_weighting` replaces the Boltzmann policy weights with uniform weights:

$$\pi_i = \frac{1}{N}, \qquad i = 1, \ldots, N. \tag{9}$$

The LM loss becomes the unweighted average over all candidates:

$$\mathcal{L}_{\text{uniform}}(\theta; C_k) = \frac{1}{N} \sum_{i=1}^{N} \ell(\theta, x_i). \tag{10}$$

This decouples the LM training step from the Q-function entirely: the LM trains on an equal mixture of all candidates regardless of their Q-values. The Q-network still trains normally (receiving rewards, computing TD updates), but its output does not influence which data the LM learns from. This serves as a controlled ablation to isolate the effect of Q-weighted curriculum selection on sample efficiency.

2.7 TD Gradients into Theta (Representation Shaping)

By default, the hidden states h_x used by the Q-network are detached from θ, so the TD loss $\frac{1}{2}(\hat{q}-y)^2$ only updates the Q-network parameters ϕ. This creates a one-directional dependency: θ determines the representations that ϕ reads, but ϕ's learning signal never shapes θ. If θ changes quickly (e.g. under a high LM learning rate), the Q-head may struggle to track the shifting representation landscape.

The flag `--td_into_theta` enables a bidirectional coupling: during the TD update, the previous action's hidden state is recomputed through the current θ *with* gradients, so that $\nabla_\theta \mathcal{L}_{\text{TD}}$ is nonzero and flows into θ via the LM optimizer. This gives θ a secondary objective: produce representations that are informative for Q-value prediction.

Concretely, at step k the Q update becomes:

1. Re-forward the previous action's tokens a_{k-1} through the current model θ_k *with gradients*:

$$h_{\text{prev}} = \texttt{last\_hidden}(\theta_k, a_{k-1}) \in \mathbb{R}^d. \qquad \text{(grad-connected to } \theta_k\text{)}$$

2. Compute the TD loss as before:

$$\mathcal{L}_{\text{TD}} = \tfrac{1}{2}\big(Q_\phi(h_{\text{prev}}) - y\big)^2, \qquad y = (1-\gamma)\, r_{\text{prev}} + \gamma\, V_{\bar{\phi}}(s_k).$$

3. Backpropagate: $\nabla_\phi \mathcal{L}_{\text{TD}}$ updates ϕ as usual; $\lambda_{\text{TD}}\, \nabla_\theta \mathcal{L}_{\text{TD}}$ updates θ via the LM optimizer.

The coefficient λ_{TD} (`--td_lambda`, default 1.0) controls the relative strength of the TD signal on θ. Setting $\lambda_{\text{TD}} < 1$ downweights the TD contribution to avoid overwhelming the language modeling gradient. The LM optimizer thus receives gradients from two sources each step: the LM loss (from the mixture or retrieved-example training) and the scaled TD loss.

Cross-step graph avoidance. A naive implementation would require keeping the full computation graph from step $k-1$ alive until step k, which is prohibitively expensive. Instead, we store the *token IDs* of the previous action (not the hidden state), and re-forward them through the current θ_k at the next step. This costs one extra single-example transformer forward pass per step—negligible compared to the N-candidate or RAG forward passes—and avoids any cross-step graph retention.

Relation to auxiliary tasks. This is analogous to auxiliary-task representation learning in RL (Jaderberg et al., 2017; UNREAL): the TD loss acts as an auxiliary objective that encourages the base model to maintain representations useful for value prediction. Unlike standard auxiliary tasks, here the auxiliary signal is the Q-learning TD error itself, creating a closed loop between the value function and the representations it operates on.

Note on the bootstrap target. Only the Q-prediction side $Q_\phi(h_{\text{prev}})$ carries gradients into θ. The bootstrap target $y = (1-\gamma)\, r + \gamma\, V_{\bar{\phi}}(s')$ remains stop-gradient with respect to both ϕ and θ, as is standard in Q-learning. The target-network value $V_{\bar{\phi}}$ uses detached hidden states.

3 Langevin-RAG Curriculum Selection

The discrete-Q approach (Section 2.5) scores a fixed batch of N candidates per step. This limits the search to the N examples that happen to appear in the current batch—a tiny fraction of the training corpus. To search over a much larger space of possible training data, we introduce a two-stage pipeline that uses the Q-network as an energy function for continuous optimisation via Stochastic Gradient Langevin Dynamics (SGLD), then retrieves real training examples from a large indexed corpus via Retrieval-Augmented Generation (RAG).

3.1 Overview

The pipeline proceeds in four stages per training step:

1. **Langevin sampling in embedding space** (Section 3.2): Initialise K parallel chains of continuous embeddings and run SGLD using the Q-network as an energy function. After burn-in, collect samples that represent "what the Q-network wants to train on."
2. **Snap to tokens** (Section 3.3): Map each continuous embedding sample to a discrete token sequence via nearest-neighbor lookup against the model's token embedding matrix.
3. **RAG retrieval** (Section 3.4): Embed the discrete query sentences with a sentence-transformer model and retrieve the top-k most similar real training examples from a pre-built FAISS index of the corpus.
4. **Unweighted LM training**: Train the language model on the retrieved examples with uniform loss (no Q-weighting), since the curriculum signal is already embedded in which examples were retrieved.

The key insight is that the Q-network encodes a preference over text via the base model's representations: $Q_\phi(\text{transformer}(x))$ assigns a scalar value to any input. By treating Q as an energy function and sampling in the continuous embedding space, we can search over an effectively infinite space of "virtual" inputs, unconstrained by which examples happen to be in the current batch.

3.2 SGLD in Embedding Space

Let $\mathbf{e} \in \mathbb{R}^{L \times d}$ be a continuous embedding sequence of length L in the GPT-2 embedding space ($d = 768$). Define the energy function:

$$E(\mathbf{e}) = \frac{1}{\tau} Q_\phi\big(\text{transformer}(\mathbf{e})\big), \tag{11}$$

where transformer($\mathbf{e}$) feeds $\mathbf{e}$ as `inputs_embeds` (bypassing the token embedding lookup), extracts the last-layer hidden state at the final position, and passes it through the Q-network Q_ϕ. The temperature $\tau > 0$ controls how peaked the resulting distribution is around high-Q regions.

We seek samples from $p(\mathbf{e}) \propto \exp(E(\mathbf{e}))$—the Boltzmann distribution that concentrates on embeddings with high Q-values. SGLD produces approximate samples via the Langevin update:

$$\mathbf{e}_{t+1} = \mathbf{e}_t + \frac{\varepsilon}{2} \nabla_{\mathbf{e}} E(\mathbf{e}_t) + \sqrt{\varepsilon}\, \boldsymbol{\eta}_t, \qquad \boldsymbol{\eta}_t \sim \mathcal{N}(0, \sigma^2 I), \tag{12}$$

where ε is the step size and σ controls the noise scale. The gradient $\nabla_{\mathbf{e}} E$ is computed by back-propagating from the Q-value through the full transformer into the input embeddings. Gradient checkpointing is used to keep memory bounded during this backward pass.

Initialisation. Each of the K chains is initialised by sampling random token IDs uniformly from the vocabulary and looking up their embeddings via the model's `wte` matrix. This places the initial points in a realistic region of embedding space.

Gradient clipping. Per-chain gradient norms are clipped to a threshold G_{embed} to prevent instability from large gradients in early Langevin steps.

Collection. The first B steps are discarded as burn-in. After burn-in, every T-th sample is collected (thinning) to reduce autocorrelation. Collection stops once the desired number of samples S has been gathered. With K parallel chains, this requires $\lceil S/K \rceil$ collection events.

3.3 Snap to Tokens

Each collected embedding sample $\mathbf{e} \in \mathbb{R}^{L \times d}$ is a continuous vector that does not correspond to any real token sequence. We map it to discrete tokens via nearest-neighbor cosine similarity against the model's token embedding matrix $W_e \in \mathbb{R}^{V \times d}$:

$$t_i = \arg\max_{v \in \{1,\ldots,V\}} \frac{\mathbf{e}_i \cdot W_e[v]}{\|\mathbf{e}_i\| \, \|W_e[v]\|}, \qquad i = 1, \ldots, L. \tag{13}$$

This produces a discrete token sequence $\mathbf{t} = (t_1, \ldots, t_L)$ that is typically incoherent natural language—a bag of loosely related tokens—since the Langevin dynamics optimises for Q-value rather than fluency. However, this sequence serves only as a *query* for RAG retrieval, not as training data itself; the sentence-transformer embedding (Section 3.4) is robust to such noise.

3.4 RAG Retrieval

Index construction (once, at startup). We stream through the training corpus and extract I fixed-length token windows. Each window is decoded to text and embedded with a pretrained sentence-transformer model (all-MiniLM-L6-v2, $d_{\text{st}} = 384$), producing normalised embedding vectors. These are stored in a FAISS inner-product index for efficient nearest-neighbor search.

Query and retrieval (each step). The S discrete query sequences from the snap step are decoded to text and embedded with the same sentence-transformer. For each query, the top-k nearest neighbours in the FAISS index are retrieved by cosine similarity. Results are deduplicated across all queries to produce $R \leq S \cdot k$ unique training examples.

Why RAG works despite gibberish queries. The sentence-transformer maps the incoherent token sequence to a dense vector that captures its topical content—the model is trained to be robust to paraphrasing and noise. Empirically, the cosine similarity between query embeddings and retrieved documents is ~ 0.44 (moderate similarity), indicating that the retrieval is coarse enough to tolerate the snap artifacts while still being meaningfully directed by the Q-network's preferences.

3.5 Langevin-RAG Algorithm

LM training is unweighted. Unlike the discrete-Q approach (Section 2.5), the LM trains on the retrieved examples with *uniform* loss. The curriculum signal is not in the *weights* on examples but in *which* examples are retrieved. The Q-network's preferences are expressed through the SGLD

$\rightarrow$ snap $\rightarrow$ RAG pipeline, which acts as a soft argmax: the Langevin dynamics finds high-Q regions of embedding space, these are projected to queries, and the queries pull relevant real data from the corpus.

Q-learning is off-policy. The Q-network update is identical to the discrete-Q version: observe reward r_k, compute the TD target from the target network, and minimise the squared TD error. The Q-learner does not need to know how the training data was selected—it only observes the resulting reward. This makes the Langevin-RAG pipeline compatible with the same normalised discounted soft Bellman equation (Eq. (3)).

Computational cost. SGLD dominates the per-step cost ($\sim 56\%$ of wall time), as it requires backpropagating through the full transformer at each Langevin step. RAG retrieval is negligible ($<1\%$). LM training accounts for $\sim 13\%$, and held-out reward evaluation for $\sim 26\%$. On an H100, each step takes ~ 3.7s with the default parameters ($K = 8$ chains, $T_{\max} = 100$ steps, $L = 64$ tokens, batch size 16).

3.6 SGLD Diagnostics

Monitoring SGLD health is critical for tuning the Langevin hyperparameters. We track:

- **Q gain** (ΔQ): The difference between Q-values of collected samples and Q-values at initialisation. Positive ΔQ indicates that SGLD is climbing the Q landscape. Empirically, ΔQ grows from ~ 0.2 early in training to ~ 0.8 as the Q-network develops more structured gradients.
- **Random-Q baseline**: Q-values of fresh random embeddings, evaluated each step. This provides an *unbiased* reference—the Q-values of collected samples are biased high by construction. The gap between SGLD Q and random Q measures the informativeness of the Langevin search.
- **Diversity**: Pairwise cosine similarity of collected embedding vectors (lower = more diverse), fraction of unique token sequences, and pairwise Jaccard similarity of token sets. With well-tuned parameters, all samples should be distinct (unique ratio = 1.0) and nearly orthogonal (cosine ~ 0.01).
- **Gradient health**: Mean gradient norm and fraction of steps where gradient clipping activates. High clip fractions ($>50\%$) suggest the step size ε or clip threshold G_{embed} may need adjustment.
- **Per-phase timing**: Wall-clock seconds for SGLD, RAG, LM training, and reward evaluation, shown as both absolute values and percentages.

3.7 Interpretation as Free-Energy Minimisation

The Boltzmann policy is the solution to a free-energy minimisation problem at each state:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \Big[-\mathbb{E}_{x\sim\pi}\big[Q_\phi(s,x)\big] + \beta\, \mathrm{KL}\big(\pi \,\|\, p_0\big) \Big], \tag{14}$$

where p_0 is the uniform distribution over C_k. Since p_0 is uniform, $\mathrm{KL}(\pi\|p_0) = \log N - H(\pi)$, and the objective reduces (up to constants) to:

$$\pi^*(\cdot \mid s) = \arg\min_{\pi} \Big[-\mathbb{E}_{x\sim\pi}\big[Q_\phi(s,x)\big] - \beta\, H(\pi) \Big]. \tag{15}$$

The first term encourages selecting high-value candidates; the second encourages exploration. The temperature β controls the tradeoff.

ELBO interpretation. Define a latent variable model where the "evidence" is the event that the current trajectory is optimal, with likelihood $p(\text{optimal} \mid x) \propto \exp(Q(s,x)/\beta)$. The policy π acts as a variational posterior over actions, and the free energy is the negative ELBO:

$$\log p(\text{optimal} \mid s) \;\geq\; \mathbb{E}_{x\sim\pi}\left[\frac{Q(s,x)}{\beta}\right] - \mathrm{KL}(\pi \,\|\, p_0). \tag{16}$$

The Boltzmann policy makes this bound tight. The normalised soft value $V(s) = \beta[\log \sum_x \exp(Q(s,x)/\beta) - \log N]$—which appears as the bootstrap term in the TD target—is the log-evidence minus $\beta \log N$, measuring how much value the current candidate batch offers above the uniform baseline.

4 Evaluation

4.1 Primary Metric: Sample Efficiency

The central question is: how many training examples does the model need to reach a given performance level?

We will measure:

- Perplexity on a fixed evaluation set as a function of training examples seen
- Performance on downstream tasks (e.g., MMLU, HellaSwag) as a function of training examples

4.2 Baselines

- **Random curriculum:** Uniform sampling from candidates
- **Loss-based curriculum:** Prioritise high-loss examples
- **Uncertainty-based curriculum:** Prioritise examples with high model uncertainty
- **Competence-based curriculum:** Examples ordered by difficulty (requires difficulty labels)

4.3 Analysis

Beyond aggregate metrics, we will examine:

- **Curriculum structure:** Does the learned curriculum exhibit interpretable patterns? Developmental stages? Topic clustering?
- **Exploration dynamics:** How does the policy entropy evolve over training? Does the temperature β produce reasonable exploration?
- **Q-function interpretability:** Which candidates receive high Q-values at different stages of training? Does the Q-function learn to identify examples that are valuable given the model's current state?
- **Q-value dynamics:** How do Q-values evolve over training? Convergence of Q-values indicates the value function has stabilised; the spread (standard deviation) reflects how strongly the Q-function discriminates between candidates.

5 Relation to Prior Work

5.1 Curriculum Learning

Graves et al. (2017) use multi-armed bandits to select tasks; Jiang et al. (2018) train a separate "mentor" network to weight examples. Our approach differs by using the language model's own

representations as features for the Q-function and applying soft Q-learning with temporal credit assignment, rather than treating selection as a stateless bandit problem.

5.2 Meta-Learning

MAML (Finn et al., 2017) learns initializations for fast adaptation. We share the computational structure—reasoning about the effect of gradient updates—but learn *what* to train on rather than *where* to start.

5.3 RL as Inference

The control-as-inference framework (Levine, 2018; Rawlik et al., 2013) casts optimal control as probabilistic inference, with the Q-function serving as an energy function and the optimal policy as the corresponding Boltzmann distribution. Our algorithm instantiates this framework in the curriculum selection setting: the Q-network provides unnormalized log-probabilities over actions, and the Boltzmann policy is derived analytically without requiring a separate policy network. This is the soft Q-learning approach of Haarnoja et al. (2017), applied to curriculum selection with a standard discount factor for Bellman contraction.

5.4 Soft Q-Learning

Soft Q-learning (Haarnoja et al., 2017) augments the standard Bellman equation with an entropy bonus, yielding a Boltzmann policy without requiring a separate policy network. SAC (Haarnoja et al., 2018) extends this to continuous actions with a learned temperature. Our setting is simpler: the action space is a finite set of N candidates, so the Boltzmann policy and soft value can be computed exactly via softmax and logsumexp. Following standard practice in deep Q-learning (Mnih et al., 2015; Lillicrap et al., 2016), we use a Polyak-averaged target network for the bootstrap value, normalise the reward by $(1 - \gamma)$, and subtract $\log N$ from the soft value to keep Q-values on the same scale as the reward.

5.5 Intrinsic Motivation

Our objective relates to compression progress (Schmidhuber) and active inference, but avoids the memory requirements of the former and the dark room problem of the latter by using a fixed held-out set as a proxy for predictive success. The absolute log-probability reward (1) provides a direct measure of predictive fitness, connecting to the epistemic homeostasis motivation: the agent is penalised for *being* in a state of poor prediction, not merely for failing to improve.

5.6 Connection to Prior Work

This proposal builds directly on Markovian Transformers for Informative Language Modeling (`https://arxiv.org/abs/2404.18988`), which introduces a framework for training language models with RL to produce causally load-bearing Chain-of-Thought reasoning. Both projects share a common structure: use RL to learn intermediate representations that improve prediction on held-out data. In the Markovian Transformers work, the learned object is a CoT:

$$\text{Question} \rightarrow \text{CoT} \rightarrow \text{Answer}$$

In the current proposal, the learned object is a curriculum:

$$\text{Model State} \rightarrow \text{Selected Data} \rightarrow \text{Improved Predictions}$$

The Markovian Transformers work demonstrates that this general approach is tractable and yields large gains on reasoning benchmarks (e.g., GSM8K: 19.6% $\rightarrow$ 57.1%). The current proposal extends this framework from learning *what to say* to learning *what to study*.

6 Broader Motivation

Language models trained to predict text develop remarkable capabilities from a simple objective. Yet they require extensive post-training to behave agentically and arguably lack a kind of global coherence. One hypothesis: the training process is purely observational—the model never takes actions that affect what it observes.

This project is a stepping stone toward studying whether learned curriculum selection produces qualitatively different agents. The immediate goal is demonstrating sample efficiency gains. The longer-term question is whether controlling one's own learning process contributes to the coherence and agency that current models seem to lack.

6.1 Long-Term Direction: Formalizing Homeostasis

Biological agents are shaped by survival pressures. Hunger, pain, and fatigue are not arbitrary reward signals—they are tied to the organism's continued existence. Current approaches to intrinsic motivation (curiosity, empowerment, compression progress) capture aspects of adaptive behavior but lack this grounding in self-preservation.

A long-term goal of this research program is to formalize homeostasis and survival into a simple, biologically plausible metric that could serve as a foundation for intrinsic motivation in artificial systems. The current project—learning to select data that improves future prediction—is a minimal step in this direction: the agent takes actions that maintain its predictive capacity, a kind of epistemic homeostasis. The absolute reward formulation makes this connection explicit: the agent is directly penalised for poor predictive fitness at every moment, not merely for failing to improve.

Algorithm 1 Curriculum Selection via Normalised Discounted Soft Q-Learning

Initialise: LM parameters θ; Q-network parameters ϕ; target parameters $\bar{\phi} \leftarrow \phi$

// Bootstrap step (no Q update)
$C_0 \leftarrow \textsc{NextBatch}(\text{stream}, N)$
$H_0 \leftarrow \textsc{DetachedHidden}(\theta, C_0)$ ▷ $[N, d]$ — no grad into θ
$q_0 \leftarrow Q_\phi(H_0)$ ▷ $[N, 1]$
$\pi_0 \leftarrow \textsc{Softmax}(q_0/\beta)$ ▷ Boltzmann policy weights
$a_0 \leftarrow \textsc{Sample}(\pi_0)$
$h_{\text{prev}} \leftarrow H_0[a_0]$ ▷ Store hidden state of selected action
$\theta \leftarrow \theta - \alpha \nabla_\theta \sum_i \pi_{0,i}\, \ell(\theta, C_0[i])$ ▷ Mixture LM update, clipped to $\|\nabla\| \le G$
$r_{\text{prev}} \leftarrow \frac{1}{M} \log p_\theta(\hat{D}_0)$ ▷ Reward after update

for $k = 1, 2, \ldots$ **do** *// Main loop*
  $C_k \leftarrow \textsc{NextBatch}(\text{stream}, N)$
  $H_k \leftarrow \textsc{DetachedHidden}(\theta, C_k)$ ▷ Serves dual purpose below
  $q_k \leftarrow Q_\phi(H_k)$

  // Q update for previous transition (target network for bootstrap)
  $\bar{q}_k \leftarrow Q_{\bar{\phi}}(H_k)$ ▷ Target network Q-values, no gradient
  $V_k \leftarrow \beta\big[\log \sum_{a'} \exp\big(\bar{q}_k[a']/\beta\big) - \log N\big]$ ▷ Normalised soft value from target network
  $y \leftarrow (1-\gamma)\, r_{\text{prev}} + \gamma\, V_k$ ▷ Normalised TD target, stop-gradient
  $h_{\text{prev}} \leftarrow \textsc{Hidden}(\theta, a_{\text{prev}})$ ▷ Detached by default; with-grad if `--td_into_theta` (Sec. 2.7)
  $\hat{q} \leftarrow Q_\phi(h_{\text{prev}})$ ▷ Re-evaluate previous action with current ϕ
  Update ϕ (and optionally θ) to minimise $\frac{1}{2}(\hat{q} - y)^2$, clipped to $\|\nabla\| \le G$
  $\bar{\phi} \leftarrow \tau\, \phi + (1-\tau)\, \bar{\phi}$ ▷ Polyak update

  // Action selection for current step (for Q-learning backup)
  $\pi_k \leftarrow \textsc{Softmax}(q_k/\beta)$ ▷ Boltzmann policy weights
  $a_k \leftarrow \textsc{Sample}(\pi_k)$
  $h_{\text{prev}} \leftarrow H_k[a_k]$

  // LM mixture update and reward
  $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_i \pi_{k,i}\, \ell(\theta, C_k[i])$ ▷ Mixture LM update, clipped to $\|\nabla\| \le G$
  $r_k \leftarrow \frac{1}{M} \log p_\theta(\hat{D}_k)$ ▷ Fresh $\hat{D}_k \subset D$, reward after update
  $r_{\text{prev}} \leftarrow r_k$
end for

Algorithm 2 Curriculum Selection via Langevin-RAG

Initialise: LM parameters θ; Q-network ϕ; target $\bar{\phi} \leftarrow \phi$; FAISS index over I corpus windows

for $k = 0, 1, 2, \ldots$ **do**

// Stage 1: SGLD in embedding space

Initialise K chains: $\mathbf{e}_0^{(j)} \leftarrow \texttt{wte}(\text{random tokens}),\ j = 1, \ldots, K$

for $t = 0, 1, \ldots, T_{\max}$ **do**

$\mathbf{e}_{t+1}^{(j)} \leftarrow \mathbf{e}_t^{(j)} + \frac{\varepsilon}{2} \operatorname{clip}\big(\nabla_{\mathbf{e}} E(\mathbf{e}_t^{(j)}),\, G_{\text{embed}}\big) + \sqrt{\varepsilon}\, \boldsymbol{\eta}_t$ ▷ Eq. (12)

if $t \geq B$ and $(t - B) \bmod T = 0$ **then**

Collect $\mathbf{e}_t^{(1)}, \ldots, \mathbf{e}_t^{(K)}$

end if

end for

// Stage 2: Snap to discrete tokens

For each collected sample $\mathbf{e}$, compute $\mathbf{t} = \text{snap}(\mathbf{e})$ via Eq. (13)

// Stage 3: RAG retrieval

Embed query texts $\{\text{decode}(\mathbf{t})\}$ with sentence-transformer

Retrieve R unique training examples from FAISS index

// Stage 4: Q update (for previous transition, if $k > 0$)

$H_k \leftarrow \textsc{DetachedHidden}(\theta, \text{retrieved examples})$ ▷ $[R, d]$

Compute V_k from $Q_{\bar{\phi}}(H_k)$ as in Eq. (3)

$h_{\text{prev}} \leftarrow \textsc{Hidden}(\theta, a_{\text{prev}})$ ▷ Detached by default; with-grad if `--td_into_theta`

Update ϕ (and optionally θ) to minimise $\frac{1}{2}(Q_\phi(h_{\text{prev}}) - [(1-\gamma)\, r_{\text{prev}} + \gamma\, V_k])^2$

$\bar{\phi} \leftarrow \tau\, \phi + (1 - \tau)\, \bar{\phi}$

// Stage 5: LM training (unweighted)

$\theta \leftarrow \theta - \alpha\, \nabla_\theta \frac{1}{R} \sum_{i=1}^{R} \ell(\theta, x_i)$ ▷ Uniform loss over retrieved examples

// Stage 6: Reward

$r_k \leftarrow \frac{1}{M} \log p_\theta(\hat{D}_k)$ ▷ Held-out evaluation

end for