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

Scott Viteri

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 much each selection improves prediction on held-out data. The core hypothesis is that learned curriculum selection will yield significantly better sample efficiency than random or heuristic curricula.

2 Technical Approach

2.1 Setup

- Transformer with parameters θ and L layers; write $f_\theta^{(\ell)}(x)$ for the residual-stream activation at layer ℓ and position of token x
- Off-the-shelf embedding model ϕ: text $\to \mathbb{R}^{d_\phi}$ for encoding streaming data
- Streaming data source, partitioned into training candidates and held-out evaluation
- Policy projections $W_\mu, W_\gamma \in \mathbb{R}^{d_\phi \times d}$ (output dimension matches ϕ)
- Learned state-dependent log-variance $\log \sigma^2 = W_\gamma z$
- Discount factor γ; shared learning rate α
- Scalar Q-function $Q_\theta(x) = W_Q\, u_x \in \mathbb{R}$, a linear projection of the last-token activation of x
- Gradient clipping threshold G (max global norm; all gradient steps are clipped to $\|\nabla\| \leq G$)

Simplification. In practice we optionally fix $\sigma^2 = 1$ (ignoring W_γ), reducing the policy to a learned mean in embedding space with unit-variance Gaussian weighting. This stabilises early training when the learned variance can be poorly calibrated.

2.2 Data Streaming

Data streams in continuously. For approximately every 1000 data points that arrive:

- $\sim$900 become training candidates (the set S_t at outer step t)
- $\sim$100 are reserved for held-out evaluation (the set D_t at outer step t)

Each incoming data point x is embedded via an off-the-shelf embedding model: $e_x = \phi(x)$. These embeddings are used for the selection mechanism described below.

Each element of S_t or D_t is conceptually one full transformer context window of text.

2.3 Algorithm

At each outer-loop stream step t, we refresh S_t, D_t from newly arrived data and hold them fixed for the duration of the inner loop. We then run K inner-loop steps on fixed S_t, D_t to update θ.

1. **Select a training example.** Let $z_k = f_{\theta_k}^{(L-1)}(\texttt{<s>})$ be the second-to-last residual-stream activation at the `<s>` token under the current weights θ_k. The Gaussian parameters in embedding space are:

$$\mu_k = W_\mu \, z_k \ \in \ \mathbb{R}^{d_\phi}, \tag{1}$$

$$\log \sigma_k^2 = W_\gamma \, z_k \ \in \ \mathbb{R}^{d_\phi}. \tag{2}$$

   This Gaussian $\mathcal{N}(\mu_k, \mathrm{diag}(\sigma_k^2))$ over the embedding space induces a distribution over S_t by normalising the density at each candidate's embedding:

$$\pi_{\theta_k}(x) = \frac{\mathcal{N}\big(\phi(x);\ \mu_k,\ \mathrm{diag}(\sigma_k^2)\big)}{\sum_{x' \in S_t} \mathcal{N}\big(\phi(x');\ \mu_k,\ \mathrm{diag}(\sigma_k^2)\big)}, \qquad x \in S_t. \tag{3}$$

   We sample a single example $x_k \sim \pi_{\theta_k}$ (with replacement from S_t).

2. **Language-modeling update (state transition).**

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_{\theta_k} \log p_{\theta_k}(x_k). \tag{4}$$

   This is the state transition of the MDP: the action x_k determines how the weights change, and θ_{k+1} becomes the state for the next inner step. We retain both θ_k and θ_{k+1} in memory.

3. **Reward.**

$$r_k = \frac{1}{|D_t|} \log p_{\theta_{k+1}}(D_t) - \frac{1}{|D_t|} \log p_{\theta_k}(D_t). \tag{5}$$

4. **Q-value of selected example.** The state in this MDP is θ_k and the action is x_k. The Q-function estimates the *discounted future expected return* from selecting x_k in state θ_k and following the policy thereafter. We obtain u_k as a byproduct of the forward pass on x_k under θ_{k+1} used to compute $\log p_{\theta_{k+1}}(D_t)$ in step 3 (specifically, $u_k = f_{\theta_{k+1}}^{(L-1)}(x_k^{\mathrm{last}})$, the second-to-last residual-stream activation at the last token of x_k). Then:

$$Q_{\theta_k}(x_k) = W_Q \, u_k \ \in \ \mathbb{R}. \tag{6}$$

5. **Next-state value (Bellman target).** We sample a batch of M candidates $\{x^{(j)}\}_{j=1}^{M} \sim \pi_{\theta_{k+1}}$ from S_t and estimate:

$$V_{k+1} \approx \frac{1}{M} \sum_{j=1}^{M} Q_{\theta_{k+1}}(x^{(j)}). \tag{7}$$

6. **Baseline value.** We sample a batch of M candidates $\{x^{(j)}\}_{j=1}^{M} \sim \pi_{\theta_k}$ from S_t and estimate:

$$V_k \approx \frac{1}{M} \sum_{j=1}^{M} Q_{\theta_k}(x^{(j)}). \tag{8}$$

   Since W_Q is a single linear projection, evaluating Q on each sampled candidate is cheap.

7. **Combined loss and update.** The Bellman regression loss for the Q-function is:

$$y_k = r_k + \gamma \ \mathrm{sg}(V_{k+1}), \tag{9}$$

$$L_Q = \big(Q_{\theta_k}(x_k) - y_k\big)^2, \tag{10}$$

where $\mathrm{sg}(\cdot)$ denotes stop-gradient: we do not back-propagate through θ_{k+1} or the Q values in the target, but we *do* back-propagate into W_Q (and the backbone that produces u_k) on the left-hand side.

The policy-gradient loss uses the advantage $A_k = \mathrm{sg}\big(Q_{\theta_k}(x_k) - V_k\big)$:

$$L_\pi = -A_k \log \pi_{\theta_k}(x_k). \tag{11}$$

Gradients of L_π flow through $\pi_{\theta_k}(x_k)$ into W_μ, W_γ, and the backbone (via z_k).

Implementation note. Computing L_π requires $\nabla_{\theta_k} \log \pi_{\theta_k}(x_k)$, but A_k is only available after the LM step (step 2) has advanced parameters to θ_{k+1}. The implementation resolves this by snapshotting θ_k before the LM step. L_Q is computed at θ_{k+1} (since $u_k = f_{\theta_{k+1}}^{(L-1)}(x_k^{\mathrm{last}})$), then θ_k is temporarily restored to compute $\nabla_{\theta_k} L_\pi$ exactly, and finally θ_{k+1} is restored before the combined gradient step.

The total loss for one inner step is:

$$\boxed{L_k = L_Q + L_\pi} \tag{12}$$

The gradient ∇L_Q is evaluated at θ_{k+1} and ∇L_π at θ_k; both are summed and applied as a single update to θ_{k+1}. The language-modeling update (step 2) is applied separately as the MDP state transition.

2.4 Architecture

Let L be the number of transformer layers. All activations used by the policy and Q-function are read from the second-to-last residual-stream layer ($\ell = L - 1$).

- **Policy state:** $z = f_\theta^{(L-1)}(\texttt{<s>}) \in \mathbb{R}^d$, the activation at the start token.
- **Policy head:** $\mu = W_\mu z \in \mathbb{R}^{d_\phi}$, $\log \sigma^2 = W_\gamma z \in \mathbb{R}^{d_\phi}$. The Gaussian $\mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$ in embedding space is normalised over S_t to give $\pi_\theta(x)$.
- **Q head:** $Q_\theta(x) = W_Q u_x \in \mathbb{R}$, where $u_x = f_\theta^{(L-1)}(x^{\mathrm{last}})$ is the activation at the last token of context window x.

The state-value baseline V is estimated by sampling M candidates from π and averaging their Q values, yielding the advantage $A = Q_\theta(x) - V$.

2.5 Alternative: Unified Meta-Learning Objective

The algorithm in Section 2.3 treats the language-modeling update as an opaque MDP transition and uses a learned Q-function to estimate the value of selecting each training example. An alternative is to *differentiate through* the LM update itself, yielding a single objective that subsumes both the data-selection policy and the training step.

2.5.1 Myopic formulation ($\gamma = 0$)

Define the one-step *meta-objective*:

$$J(\theta) \;=\; \mathbb{E}_{x\sim\pi_\theta}\Big[\tfrac{1}{|D_t|}\,\log p_{\theta'}(D_t)\Big], \qquad \theta' \;=\; \theta + \alpha\,\nabla_\theta \log p_\theta(x). \tag{13}$$

This says: select x to maximise the held-out log-probability of the model obtained after one gradient step on x. Because θ' is a differentiable function of θ (and of the choice of x), we can compute $\nabla_\theta J$ by back-propagating through the inner gradient step.

The gradient decomposes into two terms. Because x is drawn from a discrete distribution over S_t, we cannot reparameterise through the sampling; the score-function (REINFORCE) estimator gives:

$$\nabla_\theta J \;=\; \mathbb{E}_{x\sim\pi_\theta}\Big[\underbrace{\nabla_\theta\Big(\tfrac{1}{|D_t|}\log p_{\theta'}(D_t)\Big)\Big|_{\text{through }\theta'}}_{\text{pathwise (chain-rule) term}} \;+\; \underbrace{\Big(\tfrac{1}{|D_t|}\log p_{\theta'}(D_t) - b\Big)\,\nabla_\theta \log \pi_\theta(x)}_{\text{score-function (REINFORCE) term}}\Big], \tag{14}$$

where b is any baseline that does not depend on x (e.g. a running mean of the held-out log-prob). The pathwise term differentiates the held-out evaluation through the transition $\theta' = \theta + \alpha\,\nabla_\theta \log p_\theta(x)$, requiring a Hessian-vector product $\nabla^2_\theta \log p_\theta(x)$.

Relation to the current algorithm. Setting $\gamma = 0$ in the actor-critic formulation of Section 2.3 recovers the REINFORCE term of (14) (with Q playing the role of the held-out improvement and V as the baseline), but *discards* the pathwise term. The unified formulation therefore strictly dominates the $\gamma = 0$ actor-critic: it uses the same REINFORCE signal plus an additional low-variance gradient that flows through the LM transition.

What the unified formulation eliminates. Under (13) the Q-function, value baseline, Bellman loss, and the θ_k-snapshot machinery are all unnecessary. The entire RL apparatus (steps 4–7) is replaced by a single backward pass through the meta-objective. This substantially simplifies the algorithm and removes the need for the two-phase gradient computation described in the implementation note of Section 2.3.

2.5.2 Multi-step extension ($\gamma > 0$)

The myopic objective (13) optimises only the immediate one-step improvement. To recover multi-step lookahead, one can unroll K inner steps and optimise the cumulative discounted held-out improvement:

$$J_K(\theta_0) \;=\; \mathbb{E}\left[\sum_{k=0}^{K-1} \gamma^k \,\tfrac{1}{|D_t|}\,\log p_{\theta_{k+1}}(D_t)\right], \qquad \theta_{k+1} = \theta_k + \alpha\,\nabla_{\theta_k}\log p_{\theta_k}(x_k), \quad x_k \sim \pi_{\theta_k}. \tag{15}$$

Differentiating through the full unrolling is the MAML-style approach (Finn et al., 2017). This is exact but requires $O(K)$ memory for the intermediate parameter snapshots and $O(K)$ Hessian-vector products during the backward pass.

A practical middle ground is the **hybrid** approach: use the differentiable one-step objective for the immediate reward (capturing the pathwise gradient through the LM transition), and retain the Q-function only for estimating the discounted future return beyond one step:

$$L_{\text{hybrid}} \;=\; -\,\tfrac{1}{|D_t|}\,\log p_{\theta'}(D_t) \;-\; \mathrm{sg}(A_k)\,\log \pi_\theta(x_k), \tag{16}$$

where $\theta' = \theta + \alpha\,\nabla_\theta \log p_\theta(x_k)$ and A_k is the multi-step advantage from the Q-function. The first term provides the exact one-step pathwise gradient; the second provides the REINFORCE correction for long-horizon effects.

Formulation	**Q / V needed?**	**Second-order?**	**Multi-step?**
Actor-critic (current, §2.3)	Yes	No	Yes (via γ, Q)
Unified myopic (13)	No	Yes (Hessian-vec)	No
Hybrid (16)	Yes (future only)	Yes	Yes
Full unrolling (15)	No	Yes ($K\times$)	Yes (exact)

Table 1: Comparison of objective formulations. "Second-order" refers to the need for Hessian-vector products from differentiating through the LM gradient step. For GPT-2 small, one Hessian-vector product costs roughly $2\times$ a standard backward pass; PyTorch supports this via `create_graph=True`.

2.5.3 Tradeoffs

The unified myopic formulation is the simplest and eliminates all auxiliary RL machinery. Its cost is one Hessian-vector product per inner step ($\sim 2\times$ backward) and the loss of multi-step lookahead. Whether the pathwise gradient or the multi-step Q-estimate contributes more to learning quality is an empirical question; the current implementation uses the actor-critic formulation, with the unified objective as a planned future experiment.

3 Evaluation

3.1 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
- Comparison to human learning efficiency as an aspirational benchmark

3.2 Baselines

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

3.3 Analysis

Beyond aggregate metrics, we will examine:

- **Curriculum structure:** Does the learned curriculum exhibit interpretable patterns? Developmental stages? Topic clustering?
- **Exploration dynamics:** How does σ evolve over training? Does the learned variance produce reasonable exploration?
- **Q-function interpretability:** What does $Q_\theta(x)$ learn to predict? Can we understand what makes a training example "valuable"?

4 Relation to Prior Work

4.1 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 embedding selection in the model's forward pass and using RL to optimize for held-out improvement directly.

4.2 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.

4.3 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 streaming held-out data as a proxy for predictive success.

4.4 Connection to My Prior Work

This proposal builds directly on my recent work on Markovian Transformers for Informative Language Modeling (`https://arxiv.org/abs/2404.18988`). That work introduces a framework for training language models to generate Chain-of-Thought reasoning that is *causally load-bearing*—the CoT must contain all information needed to predict the answer, as the model cannot attend back to the original question.

Both projects share a common structure: use RL to learn intermediate representations that improve prediction on held-out data. In the Markovian Transformers work, we learn CoTs:

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

In the current proposal, we learn curricula:

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

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

5 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.

5.1 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. Understanding what this simple case yields will inform more ambitious formulations.

6 Timeline and Resources

Phase	**Duration**
Infrastructure and baseline implementation	6 weeks
Core algorithm implementation and debugging	8 weeks
Initial experiments and hyperparameter tuning	6 weeks
Scaling experiments on larger models	4 weeks
Analysis and writeup	4 weeks
Total	**6 months**

6.1 Compute Resources

- **Initial experiments:** Stanford Sherlock cluster (covered by existing allocation)
- **Final scaling experiments:** Cloud compute (Runpod or similar), approximately $10K for H100/H200 time to validate results on larger models (1B+ parameters)

7 Expected Outcomes

1. Empirical demonstration of sample efficiency gains (or well-documented negative result)
2. Open-source implementation of the algorithm
3. Analysis of learned curriculum structure
4. Foundation for future work on agency, homeostasis, and learned exploration in language models

8 About the Applicant

Scott Viteri is a final-year PhD student at Stanford working on machine learning and reinforcement learning, with a focus on continual learning in transformers. His relevant prior work includes:

- **Markovian Transformers for Informative Language Modeling** (`https://arxiv.org/abs/2404.18988`): Introduces a framework for training language models with RL to produce causally load-bearing Chain-of-Thought reasoning. Demonstrates large gains on reasoning benchmarks (GSM8K, ARC-Challenge, MMLU). This work establishes the technical foundation for the current proposal.

- **Epistemic Phase Transitions in Mathematical Proofs** (Cognition, 2022): Studies how belief formation operates in mathematical reasoning using network models, showing how modular structure and bidirectional inference enable certainty to emerge despite local error rates.
- **ARC Eliciting Latent Knowledge Prize winner**: Recognized for work on methods to extract truthful information from language models.
- Experience with category-theoretic approaches to machine learning, collaborating with researchers at the Topos Institute.