

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 : $\text{text} \rightarrow \mathbb{R}^{d_{\phi}}$ 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

2.2 Data Streaming

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

- ~ 900 become training candidates (the set S_t at outer step t)
- ~ 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. Candidates are sampled with replacement, so $|S_t|$ stays fixed throughout the inner loop. We then run K inner-loop steps (one per data point in Section 2.2) on fixed S_t, D_t to update θ .

All three losses (language-modeling, L_Q , and policy gradient) share the same learning rate α and update the shared parameters θ (including the transformer backbone and the projection heads W_μ, W_γ, W_Q).

1. **Compute selection vector.** Let $z_k = f_{\theta_k}^{(L-1)}(\langle \mathbf{s} \rangle)$ be the second-to-last residual-stream activation at the $\langle \mathbf{s} \rangle$ (start) token under the current weights θ_k . The policy parameters in embedding space are:

$$\mu_k = W_\mu z_k \in \mathbb{R}^{d_\phi} \quad (1)$$

$$\log \sigma_k^2 = W_\gamma z_k \in \mathbb{R}^{d_\phi}. \quad (2)$$

We draw a single sample via the reparameterisation trick ($\varepsilon \sim \mathcal{N}(0, I)$):

$$h_k = \mu_k + \sigma_k \odot \varepsilon \in \mathbb{R}^{d_\phi}. \quad (3)$$

Because μ_k and σ_k live in the same space as the embeddings $\phi(x)$, the vector h_k directly parameterises a distribution over candidates (step 2).

2. **Select from candidates.**

$$\pi_{\theta_k}(x \mid h_k) = \text{softmax}(h_k^\top E_{S_t}) \quad \text{where } E_{S_t} = [\phi(x)]_{x \in S_t} \quad (4)$$

$$x_k \sim \pi_{\theta_k}(\cdot \mid h_k). \quad (5)$$

Sampling is with replacement from S_t .

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

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta_k} \log p_{\theta_k}(x_k). \quad (6)$$

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.

4. **Reward.**

$$r_k = \frac{1}{|D_t|} \log p_{\theta_{k+1}}(D_t) - \frac{1}{|D_t|} \log p_{\theta_k}(D_t). \quad (7)$$

In practice we cache $\frac{1}{|D_t|} \log p_{\theta_k}(D_t)$ before the update and compute only $\frac{1}{|D_t|} \log p_{\theta_{k+1}}(D_t)$ after, so we never need both θ_k and θ_{k+1} in memory simultaneously.

5. **Q-function update.** 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 reuse the forward pass that computed $\log p_{\theta_k}(x_k)$ in step 3: let $u_k = f_{\theta_k}^{(L-1)}(x_k^{\text{last}})$ be the second-to-last residual-stream activation at the last token of x_k . Then:

$$Q_{\theta_k}(x_k) = W_Q u_k \in \mathbb{R}. \quad (8)$$

To form the Bellman target we need the next-state value. We sample a batch of M candidates $\{x^{(j)}\}_{j=1}^M \sim \pi_{\theta_{k+1}}(\cdot \mid h_{k+1})$ from S_t and estimate:

$$V_{k+1} \approx \frac{1}{M} \sum_{j=1}^M Q_{\theta_{k+1}}(x^{(j)}). \quad (9)$$

The Bellman target and regression loss are:

$$y_k = r_k + \gamma \text{sg}(V_{k+1}) \quad (10)$$

$$L_Q = (Q_{\theta_k}(x_k) - y_k)^2, \quad (11)$$

where $\text{sg}(\cdot)$ 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.

6. **Policy-gradient update.** We compute a state-value baseline by sampling M candidates $\{x^{(j)}\}_{j=1}^M \sim \pi_{\theta_k}(\cdot | h_k)$ from S_t :

$$V_k \approx \frac{1}{M} \sum_{j=1}^M Q_{\theta_k}(x^{(j)}). \quad (12)$$

Since W_Q is a single linear projection applied to already-available activations, evaluating Q on each sampled candidate is cheap. The advantage and policy gradient are:

$$A_k = Q_{\theta_k}(x_k) - V_k \quad (13)$$

$$\nabla_{\theta_k} J_\pi = A_k \nabla_{\theta_k} \log \pi_{\theta_k}(x_k | h_k). \quad (14)$$

Gradients flow through the reparameterised $h_k = \mu_k + \sigma_k \odot \varepsilon$ into W_μ , W_γ , and the backbone.

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:** $z = f_\theta^{(L-1)}(\langle \mathbf{s} \rangle) \in \mathbb{R}^d$, the activation at the start token.
- **Policy head:** $\mu = W_\mu z \in \mathbb{R}^{d_\phi}$
- **Variance head:** $\log \sigma^2 = W_\gamma z \in \mathbb{R}^{d_\phi}$
- **Q head:** $Q_\theta(x) = W_Q u_x \in \mathbb{R}$, where $u_x = f_\theta^{(L-1)}(x^{\text{last}})$ is the activation at the last token of context window x . This activation is obtained as a byproduct of the forward pass used for the language-modeling loss.

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

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% \rightarrow 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 agentially 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.