

Exploration of DyNA PPO with Dynamic Ensemble

Leihao (Eric) Lin*

*Department of Computer Science, Western University
Email: llin286@uwo.ca

Abstract—This is the abstract of the paper.

I. INTRODUCTION

Introduction of the project ...

II. BACKGROUNDS AND RELATED WORKS

Using PPO, a stable policy-gradient RL method to solve the black box optimization of biological sequence design. It proposes DyNA PPO, a variant of Proximal Policy Optimization:

- Learns a surrogate reward model through supervised regression on data collected.
- Using cross validated R2, selects from a pool of candidate regressors whose predictions are above a threshold, and uses their ensemble average as a simulator for policy updates.
- Falls back to model-free PPO when no accurate surrogate is available, avoiding model bias.
- Adds exploration bonus penalizing proposals too similar to past sequences to encourage diversity.

III. GOAL AND OBJECTIVES

- 1) Reproduce the standard DyNA PPO from the original paper.
- 2) Formulate the surrogate ensemble reward $r'(x)$ with weights w_i chosen to minimize a combination of surrogate bias and variance under cross-validation estimates.
- 3) Combine several surrogate models into one reward function:

$$r'(x) = \sum_{i=1}^K w_i f'_i(x).$$

- 4) Prove a bound on the regret of the model-based policy update step that decomposes into:
 - a) model bias terms, and
 - b) policy-optimization error,showing conditions under which weighted ensembling strictly improves sample efficiency over uniform averaging.
- 5) Define regret as the loss in reward by following the approximate surrogate-based policy update, compared to using the true fitness function at every step.
- 6) Decompose regret into:
 - How wrong the surrogate is, and
 - How imperfect our policy update on that surrogate is.

This allows us to optimally choose model weights to shrink model bias, rather than equally averaging all

models. As a result, the overall regret is smaller and fewer real samples are needed to learn an effective policy.

- 7) Implement the weighted DyNA PPO algorithm, integrating it into the existing PPO plus surrogate loop. Re-estimate weights each round automatically via a small convex optimization step.
- 8) Add an extra optimization step each round to resolve for the best ensemble weights.
- 9) Empirically compare the weighted DyNA PPO against the standard DyNA PPO on benchmark tasks.

IV. METHODOLOGY

A. Problem Formulation

1) *Sequence Optimization Task*: The core objective of this research is to develop an efficient method for optimizing discrete sequences through a combination of reinforcement learning and surrogate modeling. We formulate the problem as a sequential decision-making task where an agent learns to generate high-quality sequences by interacting with an expensive oracle function.

Let \mathcal{V} denote the vocabulary of discrete tokens. In our DNA sequence optimization task, we define $\mathcal{V} = \{0, 1, 2, 3\}$, representing the four DNA bases: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C), respectively. Each sequence $s = (s_1, s_2, \dots, s_T)$ consists of T tokens, where $s_t \in \mathcal{V}$ for all $t \in \{1, 2, \dots, T\}$. The sequence length T is a configurable parameter, with our experiments examining sequences of varying lengths (e.g., $T = 6, 10, 12$).

The optimization objective is to find sequences that maximize a reward function $R : \mathcal{V}^T \rightarrow \mathbb{R}$, which represents the oracle's evaluation of sequence quality:

$$s^* = \arg \max_{s \in \mathcal{V}^T} R(s) \quad (1)$$

In practical applications, the oracle function R represents expensive experimental evaluations (e.g., laboratory assays for DNA sequences, physical simulations, or real-world testing). The challenge lies in finding high-quality sequences while minimizing the number of oracle calls, as each evaluation incurs significant cost in terms of time, computational resources, or financial expense.

2) *Objective Function*: To simulate realistic biological sequence optimization, we design a complex DNA oracle function that captures multiple biologically relevant properties. The oracle function $R(s)$ is a composite scoring function consisting of eight components:

1. GC Content Scoring: The proportion of Guanine and Cytosine bases is a critical factor in DNA stability. The GC content score is defined as:

$$R_{GC}(s) = 4 \cdot \exp(-8(p_{GC} - 0.5)^2) \quad (2)$$

where $p_{GC} = \frac{|s|_G + |s|_C}{T}$ is the fraction of G and C bases in the sequence. This Gaussian-shaped reward function penalizes deviations from the optimal 50% GC content, reflecting biological preferences for balanced sequences.

2. Biological Motifs: Functional DNA sequences often contain specific motifs (subsequences) that serve biological functions. We reward the presence of known restriction sites and regulatory elements:

$$R_{\text{motif}}(s) = \sum_{m \in \mathcal{M}} w_m \cdot n_m(s) \quad (3)$$

where \mathcal{M} is the set of recognized motifs, w_m is the weight assigned to motif m , and $n_m(s)$ counts non-overlapping occurrences of motif m in sequence s . Key motifs include:

- GAATTC (EcoRI restriction site): $w = 4.0$
- GGATCC (BamHI restriction site): $w = 3.5$
- GGCC (generic 4-cutter): $w = 2.0$
- TATA (TATA box): $w = 2.0$
- ATCG (generic motif): $w = 1.0$
- GC (dinucleotide pair): $w = 0.3$

3. Position-Dependent Scoring: Base preferences vary by position within the sequence. For each position $t \in \{1, \dots, T\}$, we compute:

$$R_{\text{pos}}(s) = \sum_{t=1}^T \begin{cases} 0.5 \cdot \left(1 - 4 \left(\frac{t-1}{T-1} - 0.5\right)^2\right) & \text{if } s_t \in \{G, C\} \\ 0.3 \cdot \left(1 - 4 \left(\frac{t-1}{T-1} - 0.5\right)^2\right) & \text{if } s_t \in \{A, T\} \end{cases} \quad (4)$$

This parabolic function favors GC bases near the center of the sequence, mimicking real biological constraints.

4. Repetitive Sequence Penalty: Excessive repetition reduces sequence quality. We penalize tandem repeats:

$$P_{\text{repeat}}(s) = \sum_{\ell=1}^3 \sum_{i=1}^{T-2\ell+1} \mathbb{1}_{[\text{repeat}(s, i, \ell) \geq 3]} \cdot (\text{repeat}(s, i, \ell) - 2) \cdot \ell \cdot 0.3 \quad (5)$$

where $\text{repeat}(s, i, \ell)$ counts consecutive occurrences of the ℓ -length unit starting at position i .

5. Local Complexity: We measure local sequence complexity using a sliding window of size $w = 4$:

$$R_{\text{complex}}(s) = 3 \cdot \bar{c} - 2 \cdot \text{Var}(c) \quad (6)$$

where $c_i = \frac{|\{s_i, s_{i+1}, \dots, s_{i+w-1}\}|}{4}$ measures the fraction of unique bases in window i , \bar{c} is the mean complexity, and $\text{Var}(c)$ is the variance across all windows. This rewards high average complexity while penalizing high variance.

6. Dinucleotide Preferences: Certain adjacent base pairs have favorable or unfavorable biochemical properties:

$$R_{\text{dinuc}}(s) = \sum_{t=1}^{T-1} w_{s_t s_{t+1}} \quad (7)$$

where weights include: CG (+0.2), GC (+0.2), GA (+0.1), TC (+0.1), AA (-0.2), TT (-0.2).

7. Melting Temperature: DNA melting temperature is approximated using a simplified formula:

$$T_m = 4 \cdot (|s|_G + |s|_C) + 2 \cdot (|s|_A + |s|_T) \quad (8)$$

The normalized temperature contributes:

$$R_{T_m}(s) = 2 \cdot \frac{T_m - 20}{40} \quad (9)$$

8. Experimental Noise: To simulate real-world measurement uncertainty, we add Gaussian noise:

$$R_{\text{noise}} \sim \mathcal{N}(0, 0.15) \quad (10)$$

The total oracle reward is computed as:

$$R(s) = \max(0, R_{GC}(s) + R_{\text{motif}}(s) + R_{\text{pos}}(s) - P_{\text{repeat}}(s) + R_{\text{complex}}(s) + R_{\text{noise}}) \quad (11)$$

This multi-component oracle function creates a complex, non-convex optimization landscape with multiple local optima, making it challenging for gradient-free optimization methods. The stochastic noise component further complicates the optimization process, requiring robust learning algorithms that can handle uncertainty and generalize from limited evaluations.

B. DyNA-PPO Framework

1) Overview: The DyNA-PPO (Dynamic Model-Based Proximal Policy Optimization) framework integrates model-free reinforcement learning with model-based surrogate optimization to efficiently solve expensive discrete sequence optimization problems. The key innovation lies in combining the exploration capabilities of PPO with the sample efficiency of surrogate models, enabling the algorithm to learn high-quality policies while minimizing oracle evaluations.

The framework operates through an iterative two-phase training approach:

Phase 1: Oracle-Based Learning. In this phase, the policy network π_θ generates a batch of sequences by sampling from its learned distribution. These sequences are evaluated using the true oracle function $R(s)$, providing ground-truth feedback. The policy is then updated using the PPO algorithm to maximize expected rewards. This phase ensures that the policy learns from accurate signals but is limited by the cost of oracle evaluations.

Phase 2: Model-Based Virtual Training. To amplify learning without additional oracle calls, we train an ensemble of surrogate models $\{f_1, f_2, \dots, f_K\}$ on all previously collected oracle evaluations. These models approximate the oracle function: $f_k(s) \approx R(s)$. The policy network then

generates additional virtual sequences, which are evaluated by the surrogate ensemble rather than the expensive oracle. These predicted rewards guide further policy updates, enabling extensive exploration at minimal cost.

The algorithm alternates between these two phases across N experiment rounds. In round n , after collecting oracle feedback on B sequences (batch size), the surrogate models are retrained on the cumulative dataset $\mathcal{D} = \{(s_i, R(s_i))\}_{i=1}^{\lfloor \mathcal{D} \rfloor}$. If the models achieve sufficient accuracy (measured by R^2 score exceeding threshold τ), they guide M virtual training rounds. This hybrid approach balances exploration (via diverse sequence generation), exploitation (via policy optimization), and sample efficiency (via surrogate modeling).

The mathematical formulation of the overall objective is:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{s \sim \pi_{\theta}} [R(s)] \quad (12)$$

where the expectation is approximated through a combination of oracle-evaluated sequences and surrogate-predicted sequences. The policy $\pi_{\theta}(a_t | s_{<t})$ represents the probability distribution over the next token $a_t \in \mathcal{V}$ given the sequence prefix $s_{<t} = (s_1, \dots, s_{t-1})$.

2) *Key Hyperparameters*: The DyNA-PPO framework is controlled by several critical hyperparameters that govern the balance between exploration, exploitation, and model-based learning:

Reinforcement Learning Parameters:

- **Batch size (B)**: Number of sequences generated per round. In our experiments, we use $B \in \{32, 64\}$. Larger batch sizes provide more diverse exploration but increase computational cost per round.
- **Learning rate (α)**: Step size for gradient descent optimization. We employ an adaptive learning rate schedule with base value $\alpha_{\text{base}} = 3 \times 10^{-4}$ and minimum value $\alpha_{\text{min}} = 1 \times 10^{-5}$. The learning rate follows a cosine annealing schedule with warm restarts every 5 rounds.
- **Discount factor (γ)**: Controls the importance of future rewards. Set to $\gamma = 0.99$, emphasizing long-term sequence quality over immediate rewards.
- **PPO clip ratio (ϵ_{clip})**: Constrains policy updates to prevent destructive changes. This parameter is adaptive:

$$\epsilon_{\text{clip}} = \begin{cases} 0.3 & \text{if } n \leq 3 \text{ (early exploration)} \\ 0.1 & \text{if } \sigma_R > 2.0 \text{ (high variance)} \\ 0.2 & \text{otherwise (standard)} \end{cases} \quad (13)$$

where n is the round number and σ_R is the reward standard deviation.

- **Entropy coefficient (β_H)**: Regularization term encouraging exploration. We use an adaptive schedule:

$$\beta_H = \begin{cases} 0.02 & \text{if } n \leq 3 \\ 0.01 & \text{if } 3 < n \leq 7 \\ 0.005 & \text{if } n > 7 \end{cases} \quad (14)$$

Higher values in early rounds promote diversity; lower values later focus on exploitation.

Model-Based Learning Parameters:

- **Number of experiment rounds (N)**: Total training iterations with oracle access. We experiment with $N \in \{10, 50, 100, 200\}$ to study convergence behavior.
- **Number of model-based rounds (M)**: Virtual training iterations per experiment round. Set to $M = 5$, but adaptively reduced based on model quality:

$$M_{\text{actual}} = \begin{cases} \min(M, 5) & \text{if } R_{\text{max}}^2 > 0.3 \\ \min(M, 3) & \text{if } R_{\text{max}}^2 > 0 \\ 2 & \text{if } R_{\text{max}}^2 > -0.3 \\ 1 & \text{otherwise} \end{cases} \quad (15)$$

where R_{max}^2 is the highest R^2 score among surrogate models.

- **Model quality threshold (τ)**: Minimum R^2 score required for a surrogate model to be included in the ensemble. We investigate both fixed thresholds ($\tau = 0.2$) and dynamic thresholds that increase linearly during training:

$$\tau(n) = \begin{cases} -0.3 & \text{if } n < \lfloor N/10 \rfloor \\ \min(\tau_{\text{final}}, -0.3 + (n - \lfloor N/10 \rfloor) \cdot 0.01) & \text{otherwise} \end{cases} \quad (16)$$

where τ_{final} is the target final threshold. This allows lenient model acceptance early in training when data is scarce, gradually increasing standards as more data accumulates.

Diversity Control Parameters:

- **Diversity penalty weight (λ)**: Controls the strength of penalties for generating similar sequences. Set to $\lambda = 0.1$. During model-based training, this is further modulated:

$$\lambda_{\text{virtual}}(m) = \lambda \cdot \max\left(0.2, 1 - \frac{m}{M}\right) \quad (17)$$

where m is the current virtual training iteration, reducing diversity pressure as virtual training progresses.

- **Diversity radius (ϵ_d)**: Defines the edit distance threshold for considering sequences as similar. Set to $\epsilon_d = 3$, meaning sequences differing by fewer than 3 edits incur diversity penalties.

Exploration Parameters:

- **Exploration rate ($\epsilon_{\text{explore}}$)**: Probability of selecting exploratory actions. Follows an adaptive decay schedule:

$$\epsilon_{\text{explore}}(n) = \begin{cases} 0.3 & \text{if } n \leq 3 \\ \max(0.15, 0.4 - 0.03n) & \text{if } 3 < n \leq 7 \\ \max(0.05, 0.2 - 0.015n) & \text{if } n > 7 \end{cases} \quad (18)$$

- **Temperature (T_{sample})**: Controls the randomness of sequence generation. During guided exploration, temperature is set to:

$$T_{\text{sample}} = \begin{cases} 1.2 & \text{if } n \leq 5 \\ 1.0 & \text{otherwise} \end{cases} \quad (19)$$

Higher temperature increases diversity; lower temperature sharpens the distribution toward high-probability tokens.

Training Configuration:

- **Policy update epochs:** Number of gradient steps per batch. Adaptively set as:

$$E_{\text{policy}} = \begin{cases} 8 & \text{if } n \leq 3 \\ 4 & \text{if } 3 < n \leq 7 \\ 2 & \text{if } n > 7 \end{cases} \quad (20)$$

More epochs early to establish good initial policy; fewer epochs later to prevent overfitting.

- **Virtual training epochs:** For model-based updates, we use fewer epochs:

$$E_{\text{virtual}} = \begin{cases} 2 & \text{if } R_{\text{max}}^2 > 0 \\ 1 & \text{otherwise} \end{cases} \quad (21)$$

Reduced epochs prevent over-reliance on potentially inaccurate surrogate predictions.

- **Warm-up samples:** Initial dataset size before policy training begins. Set to 50 diverse sequences generated using multiple strategies (25% random, 25% high-GC, 25% high-AT, 25% repetitive patterns). This provides surrogate models with sufficient initial data for meaningful predictions.

These hyperparameters collectively define the learning dynamics, with many featuring adaptive schedules that respond to training progress, data availability, and model quality. This adaptivity is crucial for maintaining stable learning while maximizing sample efficiency across the full training trajectory.

C. Policy Network Architecture

1) *Network Design:* The policy network implements an autoregressive architecture that generates sequences token-by-token, conditioning each decision on the previously generated tokens and the current position. The network serves dual purposes: (1) as an actor that outputs a probability distribution over the next token, and (2) as a critic that estimates the value of the current sequence state. This actor-critic architecture is fundamental to the PPO algorithm, enabling both policy improvement and variance reduction through baseline subtraction.

Input Representation:

The network processes two types of input at each generation step t :

- **Sequence context:** The last W tokens generated so far, where W is the context window size (set to $W = 8$ in our implementation).
- **Position information:** The current position t within the sequence, encoded to capture temporal dependencies.

For a sequence $s = (s_1, s_2, \dots, s_T)$ at position t , the network observes the context window $s_{t-W:t-1} = (s_{t-W}, \dots, s_{t-1})$. If $t < W$, the context is padded with zeros: $s_{\text{context}} = (\underbrace{0, \dots, 0}_{\text{padding}}, s_1, \dots, s_{t-1})$.

Token Embedding Layer:

Each token $s_i \in \mathcal{V}$ is mapped to a dense vector representation through an embedding lookup table:

$$\mathbf{e}_i = \text{Embed}(s_i) \in \mathbb{R}^{d_e} \quad (22)$$

where $d_e = 64$ is the embedding dimension. This layer is implemented as $\text{Embed} : \mathbb{Z}_{|\mathcal{V}|} \rightarrow \mathbb{R}^{d_e}$, parameterized by a learnable matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d_e}$. For our DNA task with $|\mathcal{V}| = 4$, this creates a 4-by-64 embedding matrix.

The context window of W tokens is embedded and concatenated:

$$\mathbf{c}_{\text{embed}} = [\mathbf{e}_{t-W}, \mathbf{e}_{t-W+1}, \dots, \mathbf{e}_{t-1}] \in \mathbb{R}^{W \cdot d_e} \quad (23)$$

resulting in a flattened vector of dimension $8 \times 64 = 512$.

Positional Encoding:

To inject information about the absolute position within the sequence, we use sinusoidal positional encodings as introduced in the Transformer architecture [?]. For position $t \in \{0, 1, \dots, T-1\}$, the positional encoding is a d_e -dimensional vector computed as:

$$\text{PE}(t, 2i) = \sin\left(\frac{t}{10000^{2i/d_e}}\right) \quad (24)$$

$$\text{PE}(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d_e}}\right) \quad (25)$$

for $i \in \{0, 1, \dots, \lfloor d_e/2 \rfloor - 1\}$. These sinusoidal functions with varying frequencies enable the network to learn relative positions and extrapolate to sequence lengths beyond those seen during training. The positional encoding for position t is denoted as $\mathbf{p}_t \in \mathbb{R}^{d_e}$.

Combined Input Representation:

The final input to the policy and value networks is the concatenation of the context embedding and positional encoding:

$$\mathbf{x}_t = [\mathbf{c}_{\text{embed}}; \mathbf{p}_t] \in \mathbb{R}^{d_{\text{input}}} \quad (26)$$

where $d_{\text{input}} = W \cdot d_e + d_e = 8 \times 64 + 64 = 576$ is the total input dimension. This representation captures both what tokens have been generated (via context embeddings) and where we are in the sequence (via positional encoding).

Policy Network (π_θ):

The policy network transforms the input representation into a probability distribution over the vocabulary. It consists of a feedforward neural network with two hidden layers:

$$\mathbf{h}_1^\pi = \text{ReLU}(\mathbf{W}_1^\pi \mathbf{x}_t + \mathbf{b}_1^\pi) \quad (27)$$

$$\mathbf{h}_2^\pi = \text{ReLU}(\mathbf{W}_2^\pi \mathbf{h}_1^\pi + \mathbf{b}_2^\pi) \quad (28)$$

$$\mathbf{z}_t = \mathbf{W}_3^\pi \mathbf{h}_2^\pi + \mathbf{b}_3^\pi \quad (29)$$

where:

- $\mathbf{W}_1^\pi \in \mathbb{R}^{h \times d_{\text{input}}}$, $\mathbf{W}_2^\pi \in \mathbb{R}^{h \times h}$, $\mathbf{W}_3^\pi \in \mathbb{R}^{|\mathcal{V}| \times h}$ are weight matrices
- $\mathbf{b}_1^\pi, \mathbf{b}_2^\pi \in \mathbb{R}^h$, $\mathbf{b}_3^\pi \in \mathbb{R}^{|\mathcal{V}|}$ are bias vectors
- $h = 256$ is the hidden dimension
- $\text{ReLU}(x) = \max(0, x)$ is the rectified linear unit activation function

The logits $\mathbf{z}_t \in \mathbb{R}^{|\mathcal{V}|}$ are converted to probabilities via the softmax function:

$$\pi_\theta(a_t|s_{<t}) = \frac{\exp(z_t^{(a)})}{\sum_{a=0}^{|\mathcal{V}|-1} \exp(z_t^{(a)})} \quad (30)$$

where $z_t^{(a)}$ is the logit corresponding to action a . This probability distribution $\pi_\theta(a_t|s_{<t})$ represents the policy’s belief about which token should come next, given the sequence context and position.

Value Network (V_ϕ):

The value network estimates the expected cumulative reward from the current state, providing a baseline for advantage estimation in PPO. It shares the same input representation \mathbf{x}_t but has a separate set of parameters:

$$\mathbf{h}_1^V = \text{ReLU}(\mathbf{W}_1^V \mathbf{x}_t + \mathbf{b}_1^V) \quad (31)$$

$$\mathbf{h}_2^V = \text{ReLU}(\mathbf{W}_2^V \mathbf{h}_1^V + \mathbf{b}_2^V) \quad (32)$$

$$V_\phi(s_{<t}) = \mathbf{w}_3^V \mathbf{h}_2^V + b_3^V \quad (33)$$

where the architecture mirrors the policy network but outputs a single scalar value $V_\phi(s_{<t}) \in \mathbb{R}$ instead of a probability distribution. The value network parameters ϕ are trained jointly with the policy parameters θ to minimize the temporal difference error.

The complete policy network has approximately:

$$N_{\text{params}} = 2 \times (d_{\text{input}} \cdot h + h \cdot h) + (h \cdot |\mathcal{V}| + h \cdot 1) + |\mathcal{V}| \cdot d_e \quad (34)$$

parameters. For our configuration ($d_{\text{input}} = 576$, $h = 256$, $|\mathcal{V}| = 4$, $d_e = 64$), this yields approximately 360,000 trainable parameters.

2) *Autoregressive Generation*: The policy network generates sequences autoregressively, producing one token at a time conditioned on all previously generated tokens. This process implements the factorization:

$$\pi_\theta(s) = \prod_{t=1}^T \pi_\theta(s_t | s_{1:t-1}) \quad (35)$$

Generation Process:

Starting with an empty sequence $s_0 = \emptyset$, the generation proceeds as follows:

- 1) **Initialize**: Set $t = 1$ and $s = \square$ (empty sequence)
- 2) **For each position** $t = 1, 2, \dots, T$:
 - a) Prepare context: $s_{\text{context}} = s_{t-W:t-1}$ (with zero-padding if $t < W$)
 - b) Compute embeddings: $\mathbf{c}_{\text{embed}} = \text{Flatten}([\text{Embed}(s_i)]_{i=t-W}^{t-1})$
 - c) Add positional encoding: $\mathbf{x}_t = [\mathbf{c}_{\text{embed}}; \mathbf{p}_t]$
 - d) Forward pass: $\pi_\theta(\cdot | s_{<t}) = \text{Softmax}(\text{PolicyNet}(\mathbf{x}_t))$
 - e) Sample action: $s_t \sim \text{Categorical}(\pi_\theta(\cdot | s_{<t}))$
 - f) Append to sequence: $s \leftarrow s \cup \{s_t\}$
- 3) **Return**: Complete sequence $s = (s_1, s_2, \dots, s_T)$

Sampling Strategies:

The framework supports multiple sampling strategies for action selection at step (2e):

- **Stochastic sampling**: Draw s_t from the categorical distribution:

$$s_t \sim \text{Categorical}(\pi_\theta(\cdot | s_{<t})) \quad (36)$$

This introduces stochasticity for exploration during training.

- **Deterministic (greedy)**: Select the most probable token:

$$s_t = \arg \max_{a \in \mathcal{V}} \pi_\theta(a | s_{<t}) \quad (37)$$

This maximizes exploitation of learned patterns, typically used during evaluation.

- **ϵ -greedy**: With probability ϵ , select a random token; otherwise use stochastic sampling:

$$s_t = \begin{cases} \text{Uniform}(\mathcal{V}) & \text{with probability } \epsilon \\ \text{Categorical}(\pi_\theta(\cdot | s_{<t})) & \text{with probability } 1 - \epsilon \end{cases} \quad (38)$$

This balances exploration and exploitation, particularly useful in early training.

3) *Temperature-Controlled Sampling*: To control the diversity of generated sequences, we implement temperature scaling, a technique that modulates the sharpness of the probability distribution. Given policy logits \mathbf{z}_t before softmax, temperature $T_{\text{sample}} > 0$ is applied as:

$$\pi_\theta^T(a_t | s_{<t}) = \frac{\exp(z_t^{(a_t)} / T_{\text{sample}})}{\sum_{a=0}^{|\mathcal{V}|-1} \exp(z_t^{(a)} / T_{\text{sample}})} \quad (39)$$

The temperature parameter affects generation behavior:

- $T_{\text{sample}} > 1$: **High temperature** flattens the distribution, increasing randomness and diversity. As $T_{\text{sample}} \rightarrow \infty$, the distribution approaches uniform.
- $T_{\text{sample}} = 1$: **Standard temperature** uses the original policy distribution without modification.
- $T_{\text{sample}} < 1$: **Low temperature** sharpens the distribution, making the policy more confident and deterministic. As $T_{\text{sample}} \rightarrow 0$, sampling approaches greedy selection.

In our implementation, temperature is adaptively adjusted based on training progress:

$$T_{\text{sample}}(n) = \begin{cases} 1.5 & \text{if } n \leq 2 \text{ (early exploration)} \\ 1.2 & \text{if } n \leq 5 \text{ (moderate exploration)} \\ 1.0 & \text{if } n > 5 \text{ (standard exploitation)} \end{cases} \quad (40)$$

where n is the training round number.

4) *Top-k Filtering*: To further control generation quality, we optionally apply top-k filtering, which restricts sampling to only the k most probable tokens. The filtered distribution is computed as:

$$\pi_\theta^{\text{top-}k}(a_t | s_{<t}) = \begin{cases} \frac{\pi_\theta(a_t | s_{<t})}{\sum_{a \in \mathcal{T}_k} \pi_\theta(a | s_{<t})} & \text{if } a_t \in \mathcal{T}_k \\ 0 & \text{otherwise} \end{cases} \quad (41)$$

where \mathcal{T}_k is the set of top- k tokens:

$$\mathcal{T}_k = \{a \in \mathcal{V} : \pi_\theta(a | s_{<t}) \text{ is among the } k \text{ highest probabilities}\} \quad (42)$$

In our experiments, we use $k = 3$ when top-k filtering is enabled. This prevents the model from selecting nonsensical low-probability tokens while maintaining sufficient diversity for exploration. The combination of temperature scaling and top-k filtering provides fine-grained control over the exploration-exploitation trade-off during sequence generation.

D. Surrogate Model Ensemble

1) *Dynamic Model Selection*: A key innovation in our DyNA-PPO framework is the adaptive selection of surrogate models based on the amount of available training data. Different model classes exhibit varying sample efficiency, generalization capabilities, and computational costs. To maximize performance across all training stages, we dynamically construct an ensemble that matches the current data regime.

The model selection strategy is partitioned into three data-size regimes, each with carefully chosen model configurations:

Small Data Regime ($|\mathcal{D}| < 100$):

When training data is scarce, we prioritize simple, highly regularized models that are robust to overfitting and can provide reasonable predictions with minimal samples. The ensemble for this regime includes:

- **Random Forest (RF-XS, RF-S)**: Tree ensemble methods with aggressive regularization:

$$\text{RF-XS: } n_{\text{trees}} = 30, d_{\text{max}} = 3, n_{\text{min}}^{\text{split}} = 10, n_{\text{min}}^{\text{leaf}} = 5 \quad (43)$$

$$\text{RF-S: } n_{\text{trees}} = 50, d_{\text{max}} = 4, n_{\text{min}}^{\text{split}} = 8, n_{\text{min}}^{\text{leaf}} = 4 \quad (44)$$

where d_{max} is maximum tree depth, $n_{\text{min}}^{\text{split}}$ is minimum samples for splitting, and $n_{\text{min}}^{\text{leaf}}$ is minimum samples per leaf. These constraints prevent deep, overfitted trees.

- **K-Nearest Neighbors (KNN-XS, KNN-S)**: Non-parametric models with adaptive neighborhood size:

$$\text{KNN-XS: } k = \min(3, |\mathcal{D}|/10) \quad (45)$$

$$\text{KNN-S: } k = \min(5, |\mathcal{D}|/5) \quad (46)$$

The number of neighbors scales with data size to ensure stable predictions while maintaining local sensitivity.

- **Bayesian Ridge Regression**: Linear model with automatic relevance determination, providing uncertainty estimates and feature selection. The model assumes Gaussian priors on weights, making it particularly stable with limited data.
- **Gradient Boosting (GB-XS)**: Shallow boosted trees with conservative learning:

$$n_{\text{trees}} = 30, d_{\text{max}} = 3, \alpha = 0.1, \text{subsample} = 0.8 \quad (47)$$

where α is the learning rate and subsample ratio reduces overfitting through row sampling.

- **Gaussian Process (GP)**: Non-parametric Bayesian model with RBF kernel:

$$k(s, s') = \sigma^2 \exp\left(-\frac{\|f(s) - f(s')\|^2}{2\ell^2}\right) \quad (48)$$

where $\sigma^2 = 1.0$ is the signal variance, $\ell = 1.0$ is the length scale, and $f(\cdot)$ denotes the feature representation. GP models excel with small datasets due to their ability to quantify uncertainty.

- **Support Vector Regression (SVR-RBF-S)**: Kernel-based regression with RBF kernel:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{|\mathcal{D}|} \max(0, |y_i - f(s_i)| - \epsilon) \quad (49)$$

with $C = 1.0$ (regularization), $\epsilon = 0.1$ (ϵ -insensitive loss), and γ = scale (kernel coefficient).

The small-data ensemble contains 8 models, emphasizing diversity and robustness over individual model capacity.

Medium Data Regime ($100 \leq |\mathcal{D}| < 300$):

With moderate data availability, we balance model complexity and generalization. This regime introduces more sophisticated models while maintaining regularization:

- **Random Forest (RF-M, RF-L)**: Increased ensemble size and depth:

$$\text{RF-M: } n_{\text{trees}} = 100, d_{\text{max}} = 5, n_{\text{min}}^{\text{split}} = 5, n_{\text{min}}^{\text{leaf}} = 2 \quad (50)$$

$$\text{RF-L: } n_{\text{trees}} = 150, d_{\text{max}} = 7, n_{\text{min}}^{\text{split}} = 4 \quad (51)$$

- **Gradient Boosting (GB-M, GB-L)**: Deeper boosted ensembles:

$$\text{GB-M: } n_{\text{trees}} = 50, d_{\text{max}} = 4, \alpha = 0.05 \quad (52)$$

$$\text{GB-L: } n_{\text{trees}} = 75, d_{\text{max}} = 5, \alpha = 0.03 \quad (53)$$

Lower learning rates enable finer-grained optimization.

- **XGBoost (XGB-M)**: Advanced gradient boosting with additional regularization:

$$n_{\text{trees}} = 100, d_{\text{max}} = 5, \alpha = 0.05, \rho_{\text{col}} = 0.8, \rho_{\text{row}} = 0.8 \quad (54)$$

where ρ_{col} and ρ_{row} are column and row subsampling ratios.

- **K-Nearest Neighbors (KNN-M, KNN-Tuned)**: Fixed and adaptive neighborhood sizes:

$$\text{KNN-M: } k = 5 \quad (55)$$

$$\text{KNN-Tuned: } k = \lfloor \sqrt{|\mathcal{D}|} \rfloor \quad (56)$$

- **Multi-Layer Perceptron (MLP-M)**: Neural network with early stopping:

$$\text{Architecture: } 50 \rightarrow 25 \rightarrow 1, \quad \text{max_iter} = 500, \quad \text{validation} = 0.2 \quad (57)$$

A two-hidden-layer network with 50 and 25 neurons, trained with 20% validation split for early stopping.

- **Support Vector Regression (SVR-RBF, SVR-Poly)**: Linear and polynomial kernels:

$$\text{SVR-RBF: } k(s, s') = \exp(-\gamma \|f(s) - f(s')\|^2), \quad C = 1.0 \quad (58)$$

$$\text{SVR-Poly: } k(s, s') = (\gamma \langle f(s), f(s') \rangle + r)^d, \quad d = 2, \quad C = 1.0 \quad (59)$$

- **Bayesian Ridge Regression:** Maintained as a robust linear baseline.

The medium-data ensemble contains 11 models, providing a richer hypothesis space while avoiding extreme complexity.

Large Data Regime ($|\mathcal{D}| \geq 300$):

With abundant data, we deploy high-capacity models capable of learning complex non-linear patterns:

- **Random Forest (RF-Tuned-L, RF-Tuned-XL):** Data-adaptive configurations:

$$\text{RF-Tuned-L: } n_{\text{trees}} = \min(200, |\mathcal{D}|/2), d_{\text{max}} = \min(10, \frac{|\mathcal{D}|}{30}) \quad (60)$$

$$\text{RF-Tuned-XL: } n_{\text{trees}} = \min(300, |\mathcal{D}|), d_{\text{max}} = \min(15, \frac{|\mathcal{D}|}{20}) \quad (61)$$

Model complexity scales with dataset size, bounded to prevent excessive computation.

- **Gradient Boosting (GB-Tuned-L, GB-Tuned-XL):** Aggressive boosting:

$$\text{GB-Tuned-L: } n_{\text{trees}} = \min(100, |\mathcal{D}|/3), d_{\text{max}} = 5, \alpha = 0.05 \quad (62)$$

$$\text{GB-Tuned-XL: } n_{\text{trees}} = \min(150, |\mathcal{D}|/2), d_{\text{max}} = 6, \alpha = 0.03 \quad (63)$$

- **XGBoost (XGB-L, XGB-XL):** Advanced regularization:

$$\text{XGB-L: } n_{\text{trees}} = 150, d_{\text{max}} = 6, \alpha = 0.1, \rho_{\text{row}} = 0.7 \quad (64)$$

$$\text{XGB-XL: } n_{\text{trees}} = 200, d_{\text{max}} = 8, \alpha = 0.05, \lambda_{\text{L1}} = 0.1, \lambda_{\text{L2}} = 0.001 \quad (65)$$

L1 and L2 regularization terms prevent overfitting in the XL variant.

- **Multi-Layer Perceptron (MLP-Adaptive-XL, MLP-L):** Deep architectures:

$$\text{MLP-Adaptive-XL: } 128 \rightarrow 64 \rightarrow 32 \rightarrow 1, \text{max_iter} = 1000, \alpha_{\text{adaptive}} \quad (66)$$

$$\text{MLP-L: } 100 \rightarrow 50 \rightarrow 1, \text{max_iter} = 800, \text{activation} = \tanh \quad (67)$$

The XL variant uses adaptive learning rate scheduling for better convergence.

- **Support Vector Regression (SVR-RBF-XL, SVR-Poly-L):** High-capacity kernels:

$$\text{SVR-RBF-XL: } C = 10.0, \gamma = \text{auto}, \epsilon = 0.01 \quad (68)$$

$$\text{SVR-Poly-L: } d = 3, C = 1.0, \epsilon = 0.1 \quad (69)$$

Increased regularization parameter C allows for more complex decision boundaries.

- **K-Nearest Neighbors (KNN-Tuned-Sqrt, KNN-Tuned-L):**

$$\text{KNN-Tuned-Sqrt: } k = \lfloor \sqrt{|\mathcal{D}|} \rfloor \quad (70)$$

$$\text{KNN-Tuned-L: } k = \min(20, |\mathcal{D}|/20) \quad (71)$$

- **Bayesian Ridge Regression:** Retained as a linear baseline for ensemble diversity.

- **Gaussian Process (GP):** Included if $|\mathcal{D}| < 500$ due to $O(|\mathcal{D}|^3)$ computational complexity. Uses Matérn kernel:

$$k(s, s') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\|f(s) - f(s')\|}{\ell} \right)^\nu K_\nu \left(\sqrt{2\nu} \frac{\|f(s) - f(s')\|}{\ell} \right) \quad (72)$$

with smoothness parameter $\nu = 1.5$, balancing smoothness and flexibility.

The large-data ensemble contains up to 13 models, leveraging data abundance to fit expressive hypotheses.

Model Training Strategy: Surrogate models are retrained from scratch at each experiment round n using the cumulative dataset \mathcal{D}_n of all oracle-evaluated sequences. This retraining strategy ensures models always reflect the most current understanding of the oracle landscape.

Data Preprocessing:

Before training, we apply outlier removal to improve model robustness:

- 1) Compute interquartile range: $\text{IQR} = Q_3 - Q_1$ where Q_1 and Q_3 are the 25th and 75th percentiles of rewards.
- 2) Define outlier bounds:

$$r_{\text{lower}} = Q_1 - 1.5 \cdot \text{IQR} \quad (73)$$

$$r_{\text{upper}} = Q_3 + 1.5 \cdot \text{IQR} \quad (74)$$

- 3) Filter dataset: $\mathcal{D}_{\text{clean}} = \{(s, r) \in \mathcal{D}_n : r_{\text{lower}} \leq r \leq r_{\text{upper}}\}$

If more than 20 samples remain after outlier removal, we use $\mathcal{D}_{\text{clean}}$; otherwise, we retain all data to avoid excessive sample loss.

Cross-Validation for Model Selection:

Each candidate model f_k is evaluated using k -fold cross-validation to obtain a robust estimate of generalization performance:

$$\bar{R}_k^2 = \frac{1}{K} \sum_{j=1}^K R_k^{2,(j)} \quad (75)$$

where $K = \min(5, |\mathcal{D}_{\text{clean}}|/10)$ is the number of folds, adapted to dataset size. The fold-specific R^2 score is:

$$R_k^{2,(j)} = 1 - \frac{\sum_{i \in \mathcal{V}^{(j)}} (r_i - \hat{f}_k(s_i))^2}{\sum_{i \in \mathcal{V}^{(j)}} (r_i - \bar{r})^2} \quad (76)$$

where $\mathcal{V}^{(j)}$ is the validation set for fold j , $\hat{f}_k^{(-j)}$ is the model trained without fold j , and \bar{r} is the mean reward.

Adaptive Model Acceptance Threshold:

Models are included in the ensemble if their cross-validated R^2 score exceeds a dynamic threshold $\tau(n)$:

$$\mathcal{S}_n = \{k : R_k^2 > \tau(n)\} \quad (77)$$

We investigate two threshold strategies:

Fixed Threshold: $\tau(n) = \tau_{\text{fixed}} = 0.2$ for all rounds. This maintains consistent quality standards throughout training.

Dynamic Threshold: The threshold increases linearly to enforce progressively stricter quality requirements:

$$\tau(n) = \begin{cases} \tau_{\text{start}} = -0.3 & \text{if } n < \lfloor N/10 \rfloor \\ \min(\tau_{\text{final}}, \tau_{\text{start}} + (n - \lfloor N/10 \rfloor) \cdot \delta) & \text{otherwise} \end{cases} \quad (78)$$

where τ_{final} is the target final threshold (e.g., 0.2), $\delta = 0.01$ is the increment per round, and $\tau_{\text{start}} = -0.3$ allows even poorly-performing models initially when data is scarce. This adaptive strategy balances the need for surrogate guidance in early rounds with quality assurance in later rounds.

Fallback Strategy:

If no models exceed the threshold ($\mathcal{S}_n = \emptyset$), we employ a fallback mechanism to avoid halting model-based training:

$$\mathcal{S}_n = \{\arg \max_k R_k^2\} \quad \text{if } \max_k R_k^2 > -0.5 \quad (79)$$

This ensures at least one model is used unless all models are catastrophically poor ($R^2 < -0.5$, worse than predicting the mean).

After acceptance, models are retrained on the full cleaned dataset $\mathcal{D}_{\text{clean}}$ (without cross-validation splits) to maximize their predictive capability:

$$\hat{f}_k = \arg \min_{f \in \mathcal{H}_k} \sum_{(s,r) \in \mathcal{D}_{\text{clean}}} \mathcal{L}(f(s), r) \quad (80)$$

where \mathcal{H}_k is the hypothesis class for model type k and \mathcal{L} is the loss function (typically mean squared error).

3) *Feature Engineering*: Effective surrogate modeling requires transforming discrete sequences into continuous feature vectors that capture relevant structural information. We employ a context window encoding scheme combined with positional information.

Context Window Encoding:

For a sequence $s = (s_1, s_2, \dots, s_T)$, we extract features based on the final W tokens (where $W = 8$):

$$s_{\text{context}} = (s_{T-W+1}, s_{T-W+2}, \dots, s_T) \quad (81)$$

If $T < W$, we prepend zero-padding:

$$s_{\text{context}} = (\underbrace{0, \dots, 0}_{W-T}, s_1, \dots, s_T) \quad (82)$$

One-Hot Encoding:

Each token in the context window is one-hot encoded. For vocabulary $\mathcal{V} = \{0, 1, 2, 3\}$, token s_i is mapped to:

$$\mathbf{e}_{s_i} = \begin{bmatrix} \mathbb{1}_{[s_i=0]} \\ \mathbb{1}_{[s_i=1]} \\ \mathbb{1}_{[s_i=2]} \\ \mathbb{1}_{[s_i=3]} \end{bmatrix} \in \{0, 1\}^{|\mathcal{V}|} \quad (83)$$

The context window is flattened into a single vector:

$$\mathbf{c}_{\text{onehot}} = [\mathbf{e}_{s_{T-W+1}}; \mathbf{e}_{s_{T-W+2}}; \dots; \mathbf{e}_{s_T}] \in \{0, 1\}^{W \cdot |\mathcal{V}|} \quad (84)$$

For our configuration, this produces a 32-dimensional binary vector ($8 \times 4 = 32$).

Sinusoidal Positional Encoding:

To capture sequence length and positional information, we append a sinusoidal encoding of the sequence length T :

$$\text{PE}(T, 2i) = \sin\left(\frac{T}{10000^{2i/d_{\text{pos}}}}\right) \quad (85)$$

$$\text{PE}(T, 2i+1) = \cos\left(\frac{T}{10000^{2i/d_{\text{pos}}}}\right) \quad (86)$$

for $i \in \{0, 1, \dots, \lfloor d_{\text{pos}}/2 \rfloor - 1\}$, where $d_{\text{pos}} = 16$ is the positional encoding dimension.

Final Feature Representation:

The complete feature vector is the concatenation:

$$\phi(s) = [\mathbf{c}_{\text{onehot}}; \text{PE}(T)] \in \mathbb{R}^{W \cdot |\mathcal{V}| + d_{\text{pos}}} \quad (87)$$

For our setup, $\phi(s) \in \mathbb{R}^{48}$ (32 for context + 16 for position). This fixed-size representation enables standard regression models to process variable-length sequences.

Feature Scaling:

Each surrogate model applies standardization to its input features:

$$\tilde{\phi}(s) = \frac{\phi(s) - \mu}{\sigma} \quad (88)$$

where μ and σ are the mean and standard deviation computed on the training set. This normalization improves convergence for gradient-based models (MLP, SVR) and distances for KNN.

The context window approach captures local sequence patterns (e.g., motifs, base composition) while positional encoding provides global length information. Together, they enable surrogate models to approximate the complex, multi-component oracle function with high fidelity, achieving R^2 scores typically exceeding 0.5 after 50-100 training samples.

E. Ensemble Weighting Methods

A critical component of our surrogate modeling approach is determining how to combine predictions from multiple models. Rather than treating all models equally, we investigate three weighting strategies that adapt to model performance and training progress. The ensemble prediction for a sequence s is computed as:

$$\hat{R}_{\text{ensemble}}(s) = \sum_{k \in \mathcal{S}_n} w_k \hat{f}_k(\phi(s)) \quad (89)$$

where \mathcal{S}_n is the set of accepted models at round n , w_k is the weight assigned to model k , and \hat{f}_k is the surrogate model's prediction function. The weights satisfy $w_k \geq 0$ and $\sum_{k \in \mathcal{S}_n} w_k = 1$.

1) *Average Ensemble*: The simplest weighting scheme assigns equal weight to all accepted models:

$$w_k = \frac{1}{|\mathcal{S}_n|} \quad \forall k \in \mathcal{S}_n \quad (90)$$

This uniform weighting strategy offers several advantages:

- **Simplicity**: No additional computation or validation data required.
- **Robustness**: Reduces the impact of any single model's errors through democratic averaging.
- **Diversity preservation**: All models contribute equally, maintaining ensemble diversity.

The average ensemble prediction becomes:

$$\hat{R}_{\text{avg}}(s) = \frac{1}{|\mathcal{S}_n|} \sum_{k \in \mathcal{S}_n} \hat{f}_k(\phi(s)) \quad (91)$$

While this approach lacks adaptivity to varying model quality, it serves as a strong baseline due to the variance reduction properties of averaging. The prediction variance decreases as:

$$\text{Var}[\hat{R}_{\text{avg}}(s)] = \frac{1}{|\mathcal{S}_n|^2} \sum_{k \in \mathcal{S}_n} \text{Var}[\hat{f}_k(\phi(s))] + \frac{2}{|\mathcal{S}_n|^2} \sum_{k < k'} \text{Cov}[\hat{f}_k(\phi(s)), \hat{f}_{k'}(\phi(s))] \quad (92)$$

When models are diverse (low covariance), averaging significantly reduces variance compared to individual model predictions.

2) *Weighted Ensemble*: To account for varying model quality, we introduce performance-based weighting that assigns higher weights to models with better cross-validation scores. The weights are computed using a temperature-scaled softmax function:

$$w_k = \frac{\exp(R_k^2/T_w)}{\sum_{j \in \mathcal{S}_n} \exp(R_j^2/T_w)} \quad (93)$$

where R_k^2 is the cross-validated R^2 score of model k and T_w is a temperature parameter controlling weight concentration.

Temperature Control:

The temperature parameter T_w governs the distribution of weights:

- **Low temperature** ($T_w \rightarrow 0$): Weights become one-hot, selecting only the best model (maximum exploitation).
- **High temperature** ($T_w \rightarrow \infty$): Weights approach uniform, equivalent to average ensemble (maximum diversity).
- **Moderate temperature** ($T_w = 0.1$): Our default setting, providing sharp differentiation between models while maintaining some diversity.

The softmax formulation ensures several desirable properties:

- 1) **Automatic normalization**: $\sum_{k \in \mathcal{S}_n} w_k = 1$ by construction.
- 2) **Non-negativity**: $w_k > 0$ for all k , ensuring all models contribute.
- 3) **Monotonicity**: Higher R^2 scores yield higher weights.
- 4) **Differentiability**: Smooth weight transitions as scores change.

Mathematical Justification:

The softmax weighting can be derived from a probabilistic perspective. If we model each model's prediction as a Gaussian with precision proportional to its R^2 score:

$$p(\hat{f}_k(\phi(s)) | R(s)) \propto \exp(-\beta(1 - R_k^2)(\hat{f}_k(\phi(s)) - R(s))^2) \quad (94)$$

then the maximum likelihood ensemble weight is proportional to $\exp(\beta R_k^2)$, which normalizes to the softmax form with $T_w = 1/\beta$.

This method requires no additional validation data beyond the cross-validation already performed for model selection, making it computationally efficient while still adapting to model quality.

3) *Dynamic Ensemble*: The most sophisticated weighting strategy employs different optimization methods as training progresses, adapting to data availability and model maturity. The dynamic ensemble uses a three-phase schedule:

Phase 1: Early Rounds ($n \leq N/3$)

In the initial training phase, limited data prevents reliable weight optimization. We use performance-based weighting:

$$w_k = \frac{\exp(R_k^2/T_w)}{\sum_{j \in \mathcal{S}_n} \exp(R_j^2/T_w)} \quad \text{for } n \leq N/3 \quad (95)$$

This provides a reasonable weighting scheme without requiring additional validation splits that would further reduce the already-scarce training data.

Phase 2: Middle Rounds ($N/3 < n \leq 2N/3$)

With moderate data accumulation, we transition to ridge regression-based weight learning. Given a validation set $\mathcal{D}_{\text{val}} = \{(s_i, R(s_i))\}_{i=1}^{|\mathcal{D}_{\text{val}}|}$, we collect predictions from all models:

$$\mathbf{P} = \begin{bmatrix} \hat{f}_1(\phi(s_1)) & \hat{f}_2(\phi(s_1)) & \cdots & \hat{f}_{|\mathcal{S}_n|}(\phi(s_1)) \\ \hat{f}_1(\phi(s_2)) & \hat{f}_2(\phi(s_2)) & \cdots & \hat{f}_{|\mathcal{S}_n|}(\phi(s_2)) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{f}_1(\phi(s_{|\mathcal{D}_{\text{val}}|})) & \hat{f}_2(\phi(s_{|\mathcal{D}_{\text{val}}|})) & \cdots & \hat{f}_{|\mathcal{S}_n|}(\phi(s_{|\mathcal{D}_{\text{val}}|})) \end{bmatrix} \in \mathbb{R}^{|\mathcal{D}_{\text{val}}| \times |\mathcal{S}_n|} \quad (96)$$

The weights are learned by solving a ridge regression problem:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \|\mathbf{P}\mathbf{w} - \mathbf{r}\|^2 + \lambda_{\text{ridge}} \|\mathbf{w}\|^2 \quad (97)$$

where $\mathbf{r} = [R(s_1), R(s_2), \dots, R(s_{|\mathcal{D}_{\text{val}}|})]^T$ is the vector of true rewards and $\lambda_{\text{ridge}} = 1.0$ is the regularization parameter.

The closed-form solution is:

$$\mathbf{w}^* = (\mathbf{P}^T \mathbf{P} + \lambda_{\text{ridge}} \mathbf{I})^{-1} \mathbf{P}^T \mathbf{r} \quad (98)$$

Post-processing: Ridge regression can produce negative weights, which are undesirable for ensemble averaging. We apply rectification and normalization:

$$\tilde{w}_k = \max(0, w_k^*) \quad (99)$$

$$w_k = \frac{\tilde{w}_k}{\sum_{j \in \mathcal{S}_n} \tilde{w}_j + \epsilon} \quad (100)$$

where $\epsilon = 10^{-10}$ prevents division by zero.

Advantages of Ridge Regression:

- **Regularization**: The L2 penalty prevents overfitting to validation noise.
- **Multicollinearity handling**: When models are correlated, ridge regression can assign zero weight to redundant models, improving ensemble efficiency.
- **Stability**: The regularization term ensures numerical stability even with ill-conditioned $\mathbf{P}^T \mathbf{P}$.

Phase 3: Late Rounds ($n > 2N/3$)

In the final training phase, we employ validation-based optimization to find the globally optimal weights. This method

directly minimizes the mean squared error on the validation set:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^{|\mathcal{D}_{\text{val}}|} \left(R(s_i) - \sum_{k \in \mathcal{S}_n} w_k \hat{f}_k(\phi(s_i)) \right)^2 \quad (101)$$

subject to constraints:

$$\sum_{k \in \mathcal{S}_n} w_k = 1 \quad (102)$$

$$w_k \geq 0 \quad \forall k \in \mathcal{S}_n \quad (103)$$

We solve this constrained optimization problem using Sequential Least Squares Programming (SLSQP):

Algorithm 1 Validation-Based Weight Optimization

- 1: **Input:** Models $\{\hat{f}_k\}_{k \in \mathcal{S}_n}$, validation set \mathcal{D}_{val}
 - 2: **Initialize:** $\mathbf{w}_0 = \frac{1}{|\mathcal{S}_n|} \mathbf{1}$ (uniform weights)
 - 3: Compute prediction matrix \mathbf{P} as in equation (51)
 - 4: Define objective: $L(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{val}}|} \|\mathbf{P}\mathbf{w} - \mathbf{r}\|^2$
 - 5: Define constraints: $g_1(\mathbf{w}) = \sum_k w_k - 1 = 0$, $g_2(\mathbf{w}) = \mathbf{w} \geq \mathbf{0}$
 - 6: **Solve:** $\mathbf{w}^* = \text{SLSQP}(L, \mathbf{w}_0, \text{constraints} = \{g_1, g_2\})$
 - 7: **Return:** \mathbf{w}^*
-

This approach finds the mathematically optimal weights for the validation set, potentially discovering non-intuitive weight distributions that exploit complementary strengths of different models.

Validation Set Construction:

At each round, we construct the validation set from recent data to ensure relevance:

$$\mathcal{D}_{\text{val}} = \{(s_i, R(s_i)) : i \in \{|\mathcal{D}_n| - \min(30, |\mathcal{D}_n|/3) + 1, \dots, |\mathcal{D}_n|\}\} \quad (104)$$

This uses the last 30 sequences (or 1/3 of all data if fewer than 90 sequences exist), ensuring the validation set reflects the current exploration region.

Weight History Tracking:

Throughout training, we maintain a history of learned weights:

$$\mathcal{H}_w = \{(n, \mathbf{w}_n)\}_{n=1}^N \quad (105)$$

This enables post-hoc analysis of weight evolution, revealing which model types gain or lose importance as training progresses and the oracle landscape is better understood.

4) *Comparison and Trade-offs:* The three weighting methods span a spectrum of complexity and adaptivity:

| Method | Computation | Data Required | Adaptivity |
|----------|---|----------------|------------|
| Average | $O(\mathcal{S}_n)$ | None | None |
| Weighted | $O(\mathcal{S}_n)$ | CV scores only | Moderate |
| Dynamic | $O(\mathcal{S}_n ^3 \mathcal{D}_{\text{val}})$ | Validation set | High |

TABLE I

COMPUTATIONAL COMPLEXITY AND REQUIREMENTS OF ENSEMBLE WEIGHTING METHODS.

Theoretical Insights:

From a bias-variance perspective:

- **Average ensemble** minimizes variance through maximum diversity but may have higher bias if poor models are included.
- **Weighted ensemble** reduces bias by down-weighting poor models while maintaining reasonable variance through softmax smoothing.
- **Dynamic ensemble** can achieve lowest bias by optimally combining models, but risks overfitting to validation noise in early rounds (hence the phased approach).

The dynamic ensemble’s three-phase schedule balances these considerations: starting simple when data is scarce, introducing regularized learning with moderate data, and employing full optimization once sufficient data enables robust weight estimation. This adaptive strategy aims to achieve the best of all approaches across the full training trajectory.

F. Training Procedure

The DyNA-PPO training procedure orchestrates the interplay between oracle evaluations, policy optimization, and surrogate model learning. The algorithm progresses through N experiment rounds, each consisting of multiple training phases designed to maximize sample efficiency while maintaining exploration.

1) *Warm-up Phase:* Before the main training loop begins, we optionally execute a warm-up phase to bootstrap the learning process with diverse initial data. Cold-starting with a randomly initialized policy often yields poor-quality sequences, providing surrogate models with uninformative training signals. The warm-up phase addresses this by generating a diverse initial dataset through strategic sampling.

Warm-up Dataset Generation:

We generate $N_{\text{warmup}} = 50$ sequences using four distinct strategies, each representing 25% of the warm-up set:

- 1) **Uniform Random:** Complete random exploration of the sequence space:

$$s_i \sim \text{Uniform}(\mathcal{V}), \quad i = 1, 2, \dots, T \quad (106)$$

This provides unbiased coverage of the sequence space.

- 2) **High GC-Content:** Biased sampling toward G and C bases:

$$s_i \sim \begin{cases} \text{Uniform}(\{G, C\}) & \text{with probability 0.7} \\ \text{Uniform}(\{A, T\}) & \text{with probability 0.3} \end{cases} \quad (107)$$

For DNA sequences, this explores the high-stability region of the landscape.

- 3) **High AT-Content:** Complementary bias toward A and T bases:

$$s_i \sim \begin{cases} \text{Uniform}(\{A, T\}) & \text{with probability 0.7} \\ \text{Uniform}(\{G, C\}) & \text{with probability 0.3} \end{cases} \quad (108)$$

This provides contrast to the GC-biased sequences.

- 4) **Repetitive with Noise:** Structured patterns with stochastic perturbations:

$$s_i = \begin{cases} p_{(i \bmod 2)} & \text{with probability 0.7} \\ \text{Uniform}(\mathcal{V}) & \text{with probability 0.3} \end{cases} \quad (109)$$

where $p = (p_0, p_1)$ is a randomly chosen 2-token pattern (e.g., "AG", "TC"). This creates structured sequences with controlled variation.

Each generated sequence s is evaluated by the oracle to obtain its reward $R(s)$. The warm-up dataset $\mathcal{D}_0 = \{(s_i, R(s_i))\}_{i=1}^{N_{\text{warmup}}}$ is added to the algorithm's cumulative data storage.

Pre-training Surrogate Models:

After collecting warm-up data, we optionally pre-train surrogate models on \mathcal{D}_0 . This provides initial predictive capability, though model quality is typically poor ($R^2 < 0$) due to limited data. The pre-training serves primarily to verify model implementation and establish baseline performance.

Benefits of Warm-up:

The warm-up phase provides several advantages:

- **Diverse initialization:** Multiple sampling strategies ensure broad coverage of the sequence space.
- **Surrogate bootstrapping:** Models receive meaningful initial training data rather than starting from zero.
- **Policy initialization:** The policy network can be pre-trained on warm-up data (though we do not implement this in the current version).
- **Oracle characterization:** Early understanding of the reward distribution's range and variance.

2) *Main Training Loop:* Following the warm-up phase, the algorithm executes N experiment rounds. Each round $n \in \{1, 2, \dots, N\}$ consists of seven coordinated phases:

Phase 1: Adaptive Parameter Configuration

At the start of each round, we update hyperparameters based on training progress:

- **Learning Rate Schedule:** Cosine annealing with warm restarts:

$$\alpha(n) = \alpha_{\min} + \frac{\alpha_{\max} - \alpha_{\min}}{2} \left(1 + \cos \left(\frac{\pi \cdot (n \bmod P)}{P} \right) \right) \quad (110)$$

where $\alpha_{\max} = 3 \times 10^{-4}$, $\alpha_{\min} = 1 \times 10^{-5}$, and $P = 5$ is the restart period. This schedule periodically increases the learning rate to escape local minima.

- **Exploration Rate:** Decaying schedule balancing exploration and exploitation:

$$\epsilon_{\text{explore}}(n) = \begin{cases} 0.3 & \text{if } n \leq 3 \\ \max(0.15, 0.4 - 0.03n) & \text{if } 3 < n \leq 7 \\ \max(0.05, 0.2 - 0.015n) & \text{if } n > 7 \end{cases} \quad (111)$$

High initial exploration transitions to refined exploitation in later rounds.

- **Entropy Coefficient:** Regularization strength for policy entropy:

$$\beta_H(n) = \begin{cases} 0.02 & \text{if } n \leq 3 \\ 0.01 & \text{if } 3 < n \leq 7 \\ 0.005 & \text{if } n > 7 \end{cases} \quad (112)$$

Decreasing entropy bonus shifts from exploration to deterministic policies.

- **PPO Clip Ratio:** Adaptive clipping based on reward variance:

$$\epsilon_{\text{clip}}(n, \sigma_R) = \begin{cases} 0.3 & \text{if } n \leq 3 \\ 0.1 & \text{if } \sigma_R > 2.0 \\ 0.2 & \text{otherwise} \end{cases} \quad (113)$$

where σ_R is the standard deviation of rewards in the current batch.

Phase 2: Guided Sequence Generation

The policy network generates a batch of B sequences using a three-part strategy that balances exploitation, guided exploration, and random exploration:

Early Rounds ($n \leq 2$): Emphasize broad exploration:

$$s \sim \begin{cases} \text{Uniform}(\mathcal{V}^T) & \text{with probability 0.5} \\ \pi_{\theta}^{T=1.5} & \text{with probability 0.5} \end{cases} \quad (114)$$

where $\pi_{\theta}^{T=1.5}$ denotes temperature-scaled sampling with high temperature for diversity.

Later Rounds ($n > 2$): Use guided generation with three sub-strategies:

- **Exploitation (First 1/3 of batch):** Generate sequences near the optimal GC content identified from top-performing historical sequences:

$$\text{target_gc} = \mathbb{E}[\text{gc}(s) : s \in \mathcal{T}_{10\%}] \quad (115)$$

$$\text{desired_gc} \sim \mathcal{N}(\text{target_gc}, \sigma_{\text{gc}}/2) \quad (116)$$

where $\mathcal{T}_{10\%}$ is the top 10% of sequences by reward and σ_{gc} is their GC content standard deviation. Sequences are constructed with the desired GC content, then shuffled to randomize base positions.

- **Guided Exploration (Second 1/3 of batch):** Sample from the policy network with adaptive temperature:

$$s \sim \pi_{\theta}^{T(n)}, \quad T(n) = \begin{cases} 1.2 & \text{if } n \leq 5 \\ 1.0 & \text{if } n > 5 \end{cases} \quad (117)$$

- **Random Exploration (Final 1/3 of batch):** Uniform random sequences to maintain diversity and prevent premature convergence:

$$s \sim \text{Uniform}(\mathcal{V}^T) \quad (118)$$

This three-part strategy ensures each batch contains sequences that exploit known good patterns, explore variations guided by the policy, and maintain random exploration to discover entirely new regions.

Diversity Metrics:

After generation, we compute batch diversity:

$$D_{\text{batch}} = \frac{|\{s : s \in \mathcal{B}_n\}|}{|\mathcal{B}_n|} \quad (119)$$

where \mathcal{B}_n is the set of generated sequences in round n . This metric ranges from $1/B$ (all identical) to 1 (all unique).

Phase 3: Compute Old Log Probabilities

Before oracle evaluation, we compute the log probability of each generated sequence under the current policy. For sequence $s = (s_1, \dots, s_T)$:

$$\log \pi_{\theta_{\text{old}}}(s) = \sum_{t=1}^T \log \pi_{\theta_{\text{old}}}(s_t | s_{<t}) \quad (120)$$

This computation is performed in inference mode (no gradient tracking) and stored for later PPO updates. The "old" policy $\pi_{\theta_{\text{old}}}$ refers to the parameters before the current round's updates.

Phase 4: Oracle Evaluation and Intrinsic Rewards

Each generated sequence is evaluated by the expensive oracle function $R(s)$. To encourage exploration, we augment oracle rewards with intrinsic rewards based on novelty:

$$R_{\text{total}}(s) = R(s) + R_{\text{intrinsic}}(s, n) \quad (121)$$

The intrinsic reward quantifies how different s is from previously generated sequences:

$$R_{\text{intrinsic}}(s, n) = \underbrace{\min \left(1, \frac{\bar{d}_k(s)}{T} \right)}_{\text{novelty}} \cdot \underbrace{0.5 \cdot \max \left(0, 1 - \frac{n}{10} \right)}_{\text{exploration weight}} \quad (122)$$

where:

- $\bar{d}_k(s) = \frac{1}{k} \sum_{i=1}^k d_i$ is the average edit distance to the $k = 5$ nearest neighbors in the sequence history
- T is the sequence length (maximum possible edit distance)
- The exploration weight decays linearly, reaching zero at round 10

The intrinsic reward is highest for novel sequences in early rounds and gradually diminishes as training progresses, shifting from exploration to exploitation.

Data Storage:

All evaluated sequences are stored in the cumulative dataset:

$$\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{(s_i, R(s_i))\}_{i=1}^B \quad (123)$$

Note that we store the original oracle rewards $R(s)$, not the augmented rewards $R_{\text{total}}(s)$, ensuring surrogate models learn to predict the true oracle.

Oracle Call Tracking:

We maintain a cumulative count of oracle evaluations:

$$C_{\text{oracle}}(n) = C_{\text{oracle}}(n-1) + B \quad (124)$$

This metric is critical for assessing sample efficiency, as minimizing oracle calls is a primary objective.

Phase 5: Policy Network Update

Using the oracle-evaluated sequences and their augmented rewards, we update the policy network via PPO. The number of optimization epochs is adaptive:

$$E_{\text{policy}}(n) = \begin{cases} 8 & \text{if } n \leq 3 \\ 4 & \text{if } 3 < n \leq 7 \\ 2 & \text{if } n > 7 \end{cases} \quad (125)$$

For each epoch $e \in \{1, \dots, E_{\text{policy}}(n)\}$:

- 1) **Recompute Current Probabilities:** For each sequence s in the batch, compute $\log \pi_{\theta}(s)$ with the current (updated) policy parameters.
- 2) **Compute Value Estimates:** For each sequence position, obtain value predictions:

$$V_{\phi}(s_{<t}) = \text{ValueNet}([\text{context}(s_{<t}); \text{PE}(t)]) \quad (126)$$

Average over all positions to get a sequence-level value estimate.

- 3) **Compute Advantages:** Use the temporal difference method:

$$A(s) = R_{\text{total}}(s) - V_{\phi}(s) \quad (127)$$

Normalize advantages across the batch:

$$\hat{A}(s) = \frac{A(s) - \mu_A}{\sigma_A + \epsilon} \quad (128)$$

where μ_A and σ_A are the mean and standard deviation of advantages, and $\epsilon = 10^{-8}$ prevents division by zero.

- 4) **Compute PPO Objective:** Calculate the probability ratio:

$$r(\theta) = \frac{\pi_{\theta}(s)}{\pi_{\theta_{\text{old}}}(s)} = \exp(\log \pi_{\theta}(s) - \log \pi_{\theta_{\text{old}}}(s)) \quad (129)$$

The clipped surrogate objective is:

$$L^{\text{CLIP}}(\theta) = \mathbb{E} \left[\min \left(r(\theta) \hat{A}(s), \text{clip}(r(\theta), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}) \hat{A}(s) \right) \right] \quad (130)$$

- 5) **Compute Entropy Bonus:** Encourage exploration through policy entropy:

$$H(\pi_{\theta}) = -\mathbb{E}_{s,t} \left[\sum_{a \in \mathcal{V}} \pi_{\theta}(a | s_{<t}) \log \pi_{\theta}(a | s_{<t}) \right] \quad (131)$$

- 6) **Compute Value Loss:** Mean squared error between predicted and target values:

$$L^V(\phi) = \mathbb{E} [(V_{\phi}(s) - R_{\text{total}}(s))^2] \quad (132)$$

- 7) **Total Loss:** Combine objectives with adaptive weighting:

$$L_{\text{total}} = -L^{\text{CLIP}}(\theta) + 0.5 L^V(\phi) - \beta_H(n) H(\pi_{\theta}) + \lambda_{\text{L2}} \|\theta\|^2 \quad (133)$$

where $\lambda_{\text{L2}} = 10^{-4}$ is the L2 regularization coefficient.

- 8) **Gradient Update:** Compute gradients and apply clipped gradient descent:

$$\mathbf{g} = \nabla_{\theta} L_{\text{total}} \quad (134)$$

$$\mathbf{g}_{\text{clip}} = \text{clip}(\mathbf{g}, -g_{\text{max}}(n), g_{\text{max}}(n)) \quad (135)$$

$$\theta \leftarrow \theta - \alpha(n) \mathbf{g}_{\text{clip}} \quad (136)$$

where the gradient clipping threshold is:

$$g_{\text{max}}(n) = \begin{cases} 1.0 & \text{if } n \leq 3 \\ 0.5 & \text{if } n > 3 \end{cases} \quad (137)$$

- 9) **KL Divergence Monitoring:** Compute KL divergence between old and new policies:

$$D_{\text{KL}}(\pi_{\theta_{\text{old}}} \| \pi_{\theta}) = \mathbb{E} [\log \pi_{\theta_{\text{old}}}(s) - \log \pi_{\theta}(s)] \quad (138)$$

If $D_{\text{KL}} > 0.05$ and $e > 0$, trigger early stopping to prevent excessive policy changes that could destabilize learning.

Phase 6: Surrogate Model Training

If the cumulative dataset contains at least 30 samples ($|\mathcal{D}_n| \geq 30$), we retrain surrogate models:

- 1) **Outlier Removal:** Apply IQR-based filtering as described in Section 3.4.2.
- 2) **Model Selection:** Instantiate data-size-appropriate models as described in Section 3.4.1.
- 3) **Cross-Validation:** Evaluate each model using k -fold CV to obtain R_k^2 scores.
- 4) **Model Acceptance:** Include models with $R_k^2 > \tau(n)$ in the ensemble \mathcal{S}_n .
- 5) **Retraining:** Fit accepted models on the full cleaned dataset $\mathcal{D}_{\text{clean}}$.

Phase 7: Model-Based Virtual Training

If reliable surrogate models exist ($\mathcal{S}_n \neq \emptyset$), we perform M_{actual} virtual training rounds, where M_{actual} is determined by model quality:

$$M_{\text{actual}} = \begin{cases} \min(M, 5) & \text{if } \max_{k \in \mathcal{S}_n} R_k^2 > 0.3 \\ \min(M, 3) & \text{if } \max_{k \in \mathcal{S}_n} R_k^2 > 0 \\ 2 & \text{if } \max_{k \in \mathcal{S}_n} R_k^2 > -0.3 \\ 1 & \text{otherwise} \end{cases} \quad (139)$$

For each virtual round $m \in \{1, \dots, M_{\text{actual}}\}$:

- 1) **Generate Virtual Sequences:** Use the same guided generation strategy as Phase 2, with possible temperature adjustment:

$$T_{\text{virtual}}(m) = 1.3 - 0.1m \quad (140)$$

decreasing temperature across virtual rounds to focus on promising regions.

- 2) **Ensemble Prediction:** Compute predicted rewards using the selected ensemble weighting method (average, weighted, or dynamic):

$$\hat{R}(s) = \sum_{k \in \mathcal{S}_n} w_k \hat{f}_k(\phi(s)) \quad (141)$$

- 3) **Prediction Clipping:** Constrain predictions to observed reward ranges:

$$r_{\min} = 5\text{th percentile of } \{R(s') : (s', R(s')) \in \mathcal{D}_n\} \quad (142)$$

$$r_{\max} = 95\text{th percentile of } \{R(s') : (s', R(s')) \in \mathcal{D}_n\} \quad (143)$$

$$\hat{R}_{\text{clip}}(s) = \text{clip}(\hat{R}(s), r_{\min}, r_{\max}) \quad (144)$$

This prevents surrogate models from making unrealistic extrapolations.

- 4) **Diversity Penalty:** If model quality is sufficient ($\max_k R_k^2 > 0$) and diversity penalties are enabled, apply penalties for sequences too similar to history:

$$R_{\text{final}}(s) = \hat{R}_{\text{clip}}(s) - \lambda_{\text{div}}(m) \sum_{s' \in \mathcal{H}} \max\left(0, 1 - \frac{d(s, s')}{\epsilon_d}\right) \quad (145)$$

where:

$$\lambda_{\text{div}}(m) = \lambda \cdot \max(0.2, 1 - m/M_{\text{actual}}) \quad (146)$$

$$d(s, s') = \text{edit_distance}(s, s') \quad (147)$$

$$\mathcal{H} = \text{sequence history} \quad (148)$$

The diversity weight decreases across virtual rounds to allow exploitation.

- 5) **Policy Update:** Update the policy using predicted rewards with reduced epochs:

$$E_{\text{virtual}} = \begin{cases} 2 & \text{if } \max_k R_k^2 > 0 \\ 1 & \text{otherwise} \end{cases} \quad (149)$$

Use half the entropy coefficient:

$$\beta_{H, \text{virtual}} = 0.5\beta_H(n) \quad (150)$$

This conservative approach limits over-reliance on potentially inaccurate surrogate predictions.

- 6) **History Update:** Add virtual sequences to the history (but not to \mathcal{D}_n , as they were not oracle-evaluated):

$$\mathcal{H} \leftarrow \mathcal{H} \cup \{s_1, \dots, s_B\} \quad (151)$$

Virtual Training Safeguards:

Several mechanisms prevent degradation from poor surrogate predictions:

- **Adaptive virtual rounds:** Fewer iterations when models are poor.
- **Reduced update epochs:** Limits policy change from virtual data.
- **Prediction clipping:** Prevents unrealistic reward estimates.
- **Lower entropy bonus:** Encourages exploitation of predicted high-reward regions rather than broad exploration.

The virtual training phase enables extensive policy improvement with zero additional oracle calls, amplifying the learning signal from each expensive oracle evaluation by a factor of up to $M \times B/B = M$.

3) *Convergence and Termination:* The main training loop continues for all N rounds unless early termination criteria are met. After completing round n , we compute summary statistics:

$$\mu_R(n) = \frac{1}{B} \sum_{i=1}^B R(s_i^{(n)}) \quad (\text{mean oracle reward}) \quad (152)$$

$$R_{\max}(n) = \max_{i \in \{1, \dots, B\}} R(s_i^{(n)}) \quad (\text{best reward this round}) \quad (153)$$

$$R_{\max}^{\text{all}}(n) = \max_{(s, r) \in \mathcal{D}_n} r \quad (\text{best reward overall}) \quad (154)$$

Optional early stopping can be triggered if performance plateaus:

$$\text{Var}([\mu_R(n-2), \mu_R(n-1), \mu_R(n)]) < 10^{-3} \text{ and } \mu_R(n) > 0 \quad (155)$$

However, in our experiments we typically run all N rounds to fully characterize learning dynamics.

Upon completion, the algorithm returns the trained policy network π_{θ^*} , the final surrogate ensemble $\{\hat{f}_k\}_{k \in \mathcal{S}_N}$, and the complete evaluation history \mathcal{D}_N containing all oracle-evaluated sequences.

G. Policy Optimization

1) PPO Loss Function:

- Clipped surrogate objective
- Value function loss
- Entropy regularization for exploration

2) Adaptive Mechanisms:

- Dynamic clip ratio based on training progress
- KL divergence-based early stopping
- Gradient clipping

3) Reward Engineering:

- Intrinsic rewards for exploration
- Diversity penalties
- Reward normalization strategies

H. Adaptive Learning Mechanisms

1) Learning Rate Scheduling:

- Cosine annealing with warm restarts
- Performance-based adjustment

2) Exploration Scheduling:

- Decaying exploration rate over rounds
- Adaptive entropy coefficients

3) Threshold Management:

- Fixed vs. dynamic threshold modes
- Linear interpolation for dynamic thresholds

I. Diversity Control

1) Sequence Diversity Metrics:

- Edit distance computation
- K-nearest neighbor diversity assessment

2) Diversity Penalties:

- Novelty-based intrinsic rewards
- Exploration weight decay

J. Experimental Setup

1) Implementation Details:

- Framework: PyTorch for policy network, scikit-learn for surrogate models
- Hardware specifications

2) Evaluation Metrics:

- Oracle reward statistics (mean, max, std)
- Model R^2 scores
- Sequence diversity
- Oracle call efficiency

3) Configuration Variants:

- Ensemble methods comparison (average, weighted, dynamic)
- Threshold types (fixed vs. dynamic)
- Impact of diversity penalties

V. EXPERIMENTS AND RESULTS

A. Base Comparison

We compare the following approaches:

- Standard with average ensemble
- R^2 -based weighted ensemble
- Dynamic optimal ensemble
- Pure PPO

Metrics to track:

- Final best reward achieved
- Convergence speed (rounds to reach 90% of best)
- Reward Variance

B. Ablation Study

We test the importance of each component by removing them individually. Configurations:

- no_warmup
- no_diversity_penalty
- fixed_threshold
- uniform_weights_only
- no_context_encoding

C. Model Contribution Analysis

We analyze the contribution of each model over time. For each round, we log:

- Individual model R^2 scores
- Assigned weights
- Prediction accuracy

Visualization: stacked area charts will show the weight distribution over time.

VI. CONCLUSION

Conclusion goes here.