Project Proposal

Reinforcement Learning–Based Dynamic Ensembling of Small LLMs

Name	**Student ID**
Gadamsetty Lakshmi Monish	2023A7PS0602P
Samarth Agrawal	2023A8PS0716P
Amruth Srivatsan Suryanarayanan	2023A7PS0026P
Rushil Dosi	2023A7PS0023P

Overview

Large Language Models (LLMs) have different strengths based on their design, size, and training data. As a result, a single model often does not perform well across multiple tasks. Current ensemble methods either require running many large models for each query, which is costly, or rely on fixed rules that do not exploit the combined strengths of multiple models. This project modifies the Dynamic Ensemble Reasoning (DER) concept for small, student-friendly LLMs. We aim to create a reinforcement learning (RL) agent that redirects queries among a small group of models to produce high-quality results at a low inference cost.

Large Language Models(LLMs) all have different strengths depending on their design, size, and training data. Due to this, a single LLM model doesn't perform uniformly well across multiple tasks. The current ensemble models we have either require many LLMs for each query, which is just too costly, or rely on a set of rules that fail to take advantage of the combined strengths of using many LLMs. This project is a Design Ensemble Reasoning(DER) concept for a small, student-friendly LLMS. We will be aiming to create a Reinforcement learning (RL) agent that redirects queries among a small group of models to produce a higher-quality result with low interference cost.

Proposed Solution

We propose a Dynamic Ensemble Reasoning framework based on reinforcement learning that treats ensembling as a sequential decision process formulated as a Markov Decision Process (MDP). At each step, the RL agent selects one small LLM to query or decides to STOP. It then adds that model's output to the context. A learned policy improves a reward that balances answer quality with model-call cost. This encourages short but effective reasoning paths that leverage the strengths of different models.

We propose a DER (Dynamic Ensemble Reasoning) framework based on RL (Reinforcement Learning) that treats the ensembling process as a sequential decision-making process, formulated as an MDP (Markov Decision Process). At each step, the RL agent selects an LLM to query or STOPS. After that, it adds the model's output to the context. A learned policy is implemented that improves the reward for enhancing answer quality while also balancing this with model-call cost. This encourages short and effective reasoning paths to take advantage of the different strengths of different models.

Methodology

Ensemble Setup

We will select a set of small language models such as GPT-2 (117M), GPT-2 Large (774M), and an open LLaMA-2 variant (7B or smaller) as experts. We will fine-tune or prompt these models for the chosen tasks. During inference, the ensemble agent may query any selected model as needed. The ensemble will contain 3–5 models and permit up to 2 sequential hops to keep computation manageable.

We select a set of small LLMs such as GPT-2 (117M), GPT-2 Large (774M), and an open LLaMA-2 variant (7B or smaller) as the experts. We then fine-tune or prompt these models for the tasks that we chose. During inference, the ensemble agent may query any selected model needed. The ensemble will contain 3–5 models and permit up to 2 sequential hops to keep the computation manageable.

MDP Formulation

We model dynamic ensembling as a Markov Decision Process (MDP), where the agent sequentially decides which language model expert to invoke.

State. At time step t, the state s_t consists of the original input query and the most recent answer generated so far:

$$s_t = [Q, A_{t-1}]$$

This representation captures both the task context and the current quality of the response, allowing the agent to reason about how much improvement is still needed.

This representation captures both the task context and the current quality of the response, which allows the agent to reason how much improvement is needed.

Actions. An action a_t corresponds to selecting one expert from the available pool of LLMs or choosing a special `STOP` action to terminate the process. Each expert differs in capability and cost, making action selection a non-trivial trade-off.

An action a_t corresponds to selecting one expert from the available pool of LLMs or choosing a special `STOP` action to terminate the process. Each expert is different in capability and cost, making action selection a critical trade-off in this process.

Transitions. Upon selecting an expert, the chosen model generates a new answer using the Knowledge Transfer Prompt (KTP). This output is appended to the state, producing the next state s_{t+1}. The episode ends when `STOP` is selected or a maximum number of steps is reached.

After selecting an expert, the chosen model generates a new answer using the Knowledge Transfer Prompt (KTP). This output gets appended to the state, producing the next state s_{t+1}. The episode ends when `STOP` is selected or the maximum number of steps is reached.

Reward. The reward function evaluates the quality of the generated answer while penalizing excessive model usage. This encourages the agent to achieve strong performance with minimal expert calls.

The reward function evaluates the quality of the generated answer while penalizing any excessive model usage. This will encourage the agent to achieve strong performance along with minimal expert calls.

Policy. The agent follows a stochastic policy parameterized by θ, implemented as a neural network. Given the current state s_t, the policy outputs a probability distribution over all possible actions (i.e., expert selections). The probability of choosing action i at time step t is defined using a softmax function:

The agent follows a stochastic policy parameterized by θ, implemented as is in a neural network. Given the current state s_t, the policy outputs a probability distribution over all the possible actions (i.e., expert selections). The probability of choosing an action i at the time step t is defined using a softmax function:

$$\pi(a_t = i \mid s_t; \theta) = \frac{\exp\left(f_\theta(s_t)_i\right)}{\sum_{j=1}^{N} \exp\left(f_\theta(s_t)_j\right)}$$

where $f_\theta(s_t)$ denotes the unnormalized action scores the policy network produces, and N is the number of available actions. This method enables the agent to balance exploration and exploitation while redirecting the queries to the most suitable expert.

Reward Function

The reward function plays a central role in guiding the ensemble agent, for balancing three competing objectives: the overall quality of the generated answer, the improvement from invoking an additional expert, and the computational cost from doing so.

Let $P(\cdot)$ denote an answer-quality metric such as BERTScore, which correlates well with human judgment. At time step t, the selected expert M_{a_t} produces an output $\hat{y}_t$. We define the immediate reward as:

$$R_t = \{\ P\ (\hat{y}_t) - \alpha C(M_{a_t}), t = 0 P(\hat{y}_t) + \beta \Delta P(\hat{y}) - \alpha C(M_{a_t}), t > 0$$

where $C(M_{a_t})$ denotes the computational cost of invoking expert M_{a_t}, and

$$\Delta P(\hat{y}) = P(\hat{y}_t) - P(\hat{y}_{t-1})$$

captures the incremental improvement in answer quality between the successive steps. The coefficients α and β control the trade-off between the computation cost and improvement in quality.

To encourage efficient termination, we introduce a threshold-based adjustment to the reward. Let p_0 be a predefined quality threshold and $T_{\max}$ the maximum allowed number of steps. The final reward is defined as:

$$\mathcal{R}(s_t, a_t) = \{ \, R_t + \gamma, t \leq T_{\max} and P(\hat{y}_t) \geq p_0 R_t - \gamma, t = T_{\max} and P(\hat{y}_t) < p_0$$

where γ provides an additional bonus or penalty depending on whether the agent terminates with a sufficient quality answer.

Intuitively, this reward structure will encourage the agent to continue querying the experts only when they realistically and meaningfully improve the answer, while discouraging unnecessary model calls. As a result of this, the agent learns to produce strong answers using short and cost-effective expert sequences.

Knowledge Transfer Prompt (KTP)

The Knowledge Transfer Prompt (KTP) enables successive language model experts to improve upon the previous answers rather than generating responses from scratch. The prior output is framed in the LLM as an answer written by another student, encouraging the current model to treat it as warm guidance instead of strict instructions. This design mitigates the need to override bias and promotes critical refinement while explicitly preventing references to other students. As a result, KTP facilitates hidden knowledge transfer across experts while producing a single, coherent final response.

RL Training

We train the ensemble routing agent using reinforcement learning to dynamically select an effective sequence of LLM experts while minimizing the computational cost. Following prior work on Dynamic Ensemble Reasoning, the expert selection problem is treated as a sequential decision process and optimized using Proximal Policy Optimization (PPO), chosen for its training stability and sample efficiency.

Policy Network. The policy is implemented as a lightweight neural network that operates on the fixed embeddings in the current state, which consists of the input query and the most recent answer. To keep training efficient and decoupled from expert models, the policy network is significantly smaller than the LLMs themselves and only outputs action probabilities over the available experts.

Training Algorithm. We adopt PPO to update the policy parameters based on collected trajectories. At each step, the agent selects an expert, observes the reward derived from answer quality and cost, and updates the policy using clipped policy gradients. A value function is trained alongside the policy to estimate the expected returns and stabilize the learning.

Efficiency Considerations. All expert models remain frozen during training. Embeddings and rewards are precomputed wherever it is possible, avoiding backpropagation through the LLMs and significantly reducing the training overhead cost. This design allows the routing agent to get trained efficiently while using strong but computationally expensive experts at inference time.

Models and Datasets

To implement the proposed framework, we instantiate the system with a small set of instruction-tuned language models and benchmark datasets. These choices are from the dual goals of computational efficiency and meaningful evaluation of the routing agent's behavior.

Language Models. We select three compact LLMs with complementary strengths:

- **LLaMA 3.2 (3B, quantized):** Chosen for its strong reasoning ability under low-resource constraints, making it suitable for efficient inference on limited hardware.
- **Qwen 2.5 (3B, quantized):** Selected for its strengths in mathematical reasoning, coding, and structured technical tasks, providing capabilities the are complementary in the expert tool
- **Mistral Instruct (7B):** Included as a higher-capacity model offering improved generation quality when additional memory and compute are available.

Together, these models form a diverse expert pool that allows the routing agent to learn important trade-offs between the answer quality and the computational cost.

Datasets. We evaluate the system on two datasets with increasing reasoning complexity:

- **BoolQ (SuperGLUE):** A binary question-answering benchmark used as an initial validation dataset. Its restricted output space ("yes" or "no") and exact-match accuracy metric give us a controlled environment for checking the stability and correctness of the RL routing mechanism.
- **HotpotQA:** A multi-hop question-answering dataset with reasoning across multiple supporting passages and produces free-text answers. The dataset tests the agent's ability to detect incomplete responses and trigger sequential refinement through the KTP (Knowledge Transfer Prompt).

Together, these datasets allow systematic evaluation of both simple decision-making behavior and complex, multi-step reasoning behavior.

Ensembling Methods evaluated

We evaluate three different ensembling strategies to understand when and why dynamic routing helps, compared to simpler alternatives.

Static Round-Robin (Baseline). This is the simplest approach. The system cycles through the available experts in fixed order, such as LLaMA $\rightarrow$ Qwen $\rightarrow$ Mistral, for every question. It does not look at the question itself or at the previous answers. We use this as a baseline because if a learned agent cannot beat this basic strategy, it indicates that the routing policy isn't learning anything useful.

Self-Correction (Single-Expert Refinement). Here, instead of switching between different experts, the system continuously asks the same model to improve its previous answer. The idea is that a reasonably strong model may notice and fix its own mistakes when given another chance. This method helps in checking whether spending more effort on a single expert is as effective as asking multiple experts.

Dynamic Ensemble Reasoning (DER). In this approach, expert selection is done by a reinforcement learning agent. At each step, the agent looks at the original question and the current answer and decides which expert it wants to call next. The agent is trained with a reward that discourages unnecessary model calls, so it learns to use more than necessary experts only when they are likely to improve the answer.

Results

We report the preliminary results comparing the three ensembling strategies across the selected datasets. For each method, we measure task accuracy, the average number of expert calls per query, and an estimated average computational cost. These metrics give us the ability to evaluate both answer quality and efficiency.

Dataset	**Ensembling Method**	**Accuracy**	**Avg Calls**	**Avg Cost**
3*BoolQ (Yes/No)	Round-Robin	89.6%	1.46	9.90
	Self-Correction	88.3%	1.25	1.19
	DER (Agent)	85.6%	1.44	11.3
3*HotpotQA	Round-Robin			
	Self-Correction			
	DER (Agent)			

Table 1: Comparison of ensembling methods across datasets.

On BoolQ, all three methods achieve competitive accuracy, with the static round-robin baseline performing strongly due to the task being very simple. Self-correction achieves a comparable performance while requiring fewer model calls and significantly lower cost, meaning that iterative refinement by a single expert is effective for binary question answering. The DER agent costs higher in this setting and doesn't outperform simpler baselines, showing us that dynamic routing provides limited benefits on tasks with restricted output spaces.

Objectives and Deliverables

Primary Objectives

1. Reproduce DER-like gains using small, accessible models.

2. Achieve improvements in accuracy, BERTScore, ROUGE, and inference efficiency (fewer calls, reduced FLOPs) vs. single-model and static ensemble baselines.

Deliverables

- Codebase implementing the RL-based router and evaluation pipeline.
- Trained policy checkpoints and logs.
- Final report with experiments, ablations, and cost measurements.
- Presentation notebook and demo script.

Evaluation Tasks and Datasets

We will test across multiple NLP tasks:

- **Reasoning / Math:** GSM8K
- **Question Answering:** SQuAD or MC-QA subsets
- **Summarization:** XSum or CNN/DM subsets
- **Classification:** SST-2 sentiment dataset

Related Works

- Hu, J. et al. (2024). *Dynamic ensemble reasoning for LLM experts.* arXiv:2412.07448v2.
- Lu, K. et al. (2024). *Routing to the expert: Efficient reward-guided ensemble of large language models.* NAACL 2024. https://doi.org/10.18653/v1/2024.naacl-long.109.
- Liu, Z. et al. (2024). *A dynamic LLM-powered agent network for task-oriented agent collaboration.* COLM 2024. arXiv:2310.02170.
- Jiang, D. et al. (2023). *LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion.* ACL 2023. https://doi.org/10.18653/v1/2023.acl-long.792.
- Fu, Y. et al. (2025). *RLAE: Reinforcement learning-assisted ensemble for LLMs.* EMNLP 2025. aclanthology.org/2025.emnlp-main.680.pdf.
- Zeng, A., Shen, T., Lin, Z. (2024). *Mixture-of-Agents: Collaborative inference among large language models.* ICLR 2024. arXiv:2406.04692.