COMET: Composite Objective Modeling for Enhanced Task-solving

Aryan Singh*[1], Hee Su Chung*[1], Arjun Prakash[1], Nora Ayanian[1]
Brown University, United States

Abstract

Large Language Models (LLMs) such as GPT-4 have extended their ability beyond text generation to complex tasks such as the design of reward functions for agent-based task execution. While human engineers are capable of designing these functions to train agents to complete tasks, LLMs offer a scalable alternative for generating efficient algorithms. However, adapting these pre-trained agents to navigate new environments or undertake more complex missions remains a challenge. To address this, we introduce COMET: Composite Objective Modeling for Enhanced Task-solving, a method leveraging the in-context learning capabilities of LLMs to enhance agent adaptability. This paper demonstrates how COMET allows a point robot to follow a certain trajectory to reach a goal, using reward functions autonomously generated by LLMs. Our approach not only showcases the adaptability of agents in varied environments but also underscores the potential of LLMs to handle increasingly complex decision-making scenarios. Through COMET, we aim to set a new standard in agent flexibility and performance in dynamic settings.

Introduction

Autonomous agents have shown proficiency in diverse areas such as text generation, computation, and task manipulation. However, training these agents with specific datasets and writing reward functions demands significant effort, time, and financial resources. A major challenge in deploying autonomous agents is their limited ability to adapt to new environments using previously trained parameters. To address this issue, we used Large Language Models (LLMs) to design versatile reward functions that accommodate various objectives, thus optimizing outcomes. To overcome this bottleneck, we utilized GPT-4-Turbo to generate executable reward functions in a zero-shot manner, saving the network parameters and using them as checkpoints for retraining with different desired goals. The aim of this project was to facilitate a seamless transition from general task descriptions to executable code, enhancing the agent's ability to perform

*Equal contribution

tasks that require subtly different behaviors based on complex objectives.

Furthermore, recent developments in leveraging Large Language Models (LLMs) have revolutionized the field, offering a scalable and efficient approach to reward function design ((Ma et al., 2023), (Hazra, Dos Martires, & De Raedt, 2024)), and how they can be used to shape reward functions for better agent training on different environments. However, we realized that there is less research done on composite reward modeling. Most papers use the LLMs to fulfill goals that have been predetermined by the environment (Song et al., 2023) or have been used for low-level robotic tasks and planning (Yu et al., 2023). Through this paper, we attempt to go beyond predetermined goals by first making the agent learn a reward function that conflicts with the initial environment design and try to make it achieve multiple goals through one reward function.

Hence, we propose COMET (Composite Objective Modeling for Enhanced Task-solving) as a way to use LLMs as a tool for modeling composite reward functions. Through COMET, we are able to:

1. Harness LLMs to design a generalizable reward function for the original environment that can be easily modified for further use
2. Use the environment source code as state context in order to design a novel reward function that goes against the initial reward function
3. Train the model on this new reward function and save the network parameters for the next training loop
4. Have the LLM generate a composite reward function combining the two previous goals, and use the previous network parameters to train on this new reward function. While this approach encountered some challenges, this method improved upon it by integrating both objectives.

Markov Decision Processes

A Markov Decision Process (MDP) is a stochastic decision-making process that serves as a representation of the different interactions between an agent and an environment over a series of different time steps. An agent is defined as the learner and decision-maker which attempts to maximize its reward throughout its training, and the environment is defined as everything that the agent interacts with.

We can use this framework to model the decision-making capabilities of the agent. More formally, the MDP can be defined as a tuple $\langle S, A, P, r, \gamma \rangle$(Sutton & Barto, 2018).

$$s \in S : \text{States.}$$

$$a \in A : \text{Actions.}$$

$$r : S \times A \to \mathbb{R} : \text{Rewards.}$$

$$\gamma : \text{Discount factor.}$$

$$P : S \times A \times S \to [0, 1] : \text{Transition Probability.}$$

Here S is the set of all possible states in the environment, where a state represents the configuration of an environment at a particular time step t. A is defined as the set of all actions that can be taken by the agent. P is the transition probability, which refers to the probability of going from one state to another. It can be defined as $P(s', r|s, a)$ where s' is the new state, and s, a represents the current action and state. R is the reward function, which is defined as the immediate reward you get if the agent executes the current action in the current state. It is formally defined as a function $R(s, a)$ where s is the current state and a is the current action. One important factor to consider for a reward function is the discount rate, denoted by γ. This rate controls the present value of future rewards and discounts their values in order to prevent infinite rewards. Hence, the total reward we can get can be represented by:

$$\Sigma_{k=0}^{\infty} \gamma^k R_{k+t+1}$$

where $0 \leq \gamma \leq 1$. We use R_{k+t+1} so that we appropriately apply the discount factor to the correct future reward at the specific time step e.g., at time step $t + 3$, the discount factor applied should be γ^2.

The goal of MDP is to find the optimal policy for the decision maker, in this case, a function that will reach a desired state for the agent. A policy can be formally defined as $\pi(a \mid s)$ at time t, stating what the probability of the current action is to be a if the current state of the environment at time t is s. Using this, we would want to find the best policy such that starting from our current state, we are able to maximize the total rewards from our environment. Mathematically, this can be expressed as:

$$v_\pi = \mathbb{E}_\pi[\Sigma_k^{\infty} \gamma^k R_{k+t+1} \mid S_t = s], \text{for all s} \in \text{S}$$

This is formally known as the state-value function, and it tells us what the expected reward is if we follow policy π if we are currently in state S_t. We can define a similar mathematical function for the action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[\Sigma_k^{\infty} \gamma^k R_{k+t+1} \mid S_t = s, A_t = a] \text{for all s} \in \text{S and for all a } \in A(s)$$

which tells us the expected reward return if we follow policy π, are currently in state S_t, and take action A_t. The end goal of an MDP is to find an optimal policy such that will allow us to maximize our total rewards in the long run. More formally, we want to find the following:

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

so that we are able to maximize our total rewards from the interactions between the agent and environment in the long run.

Environment

The environment we used was the Gymnax Point Robot environment (Dorfman, Shenfeld, & Tamar, 2021). The state space consists of 2 continuous Euclidean spaces: $S = \mathbb{R}^2 \times \mathbb{R}^2$, where the first component represents the position of the point robot itself (p_{robot}) in a 2-dimensional Euclidean space, and the second component represents the position of the goal we want to travel to (p_{goal}). The action space of the environment is represented via $A = \{a \in \mathbb{R}^2 | \|a\| \leq F_{max}\}$ where the magnitude of each action taken by the point robot is clipped by a F_{max} parameter we set in order to ensure that the movements of the robot are within the boundary imposed by the environment.

The reward function, $R : S \times A \to \mathbb{R}$, for this environment is able to take two forms: a dense reward system and a sparse reward system. The dense reward system is where the reward function takes the form $R(s, a) = -\|p_{robot} - p_{goal}\|$ and is proportionally related to the distance between the robot and the goal. The sparse reward system is where the reward function takes the form:

$$R(s, a) = \begin{cases} 1 & \text{if } \|p_{robot} - p_{goal}\| \leq r \\ 0 & \text{otherwise} \end{cases}$$

where r represents the goal radius. The dense reward allows for more fine-grained approaches to the reward tuning as we can calculate the reward at each step of the agent, whereas the sparse setting allows for a more simplified view of the environment.

Proximal Policy Optimization (PPO)

The algorithm we used for the reinforcement learning aspect of this experiment is the Proximal Policy Optimization algorithm, which is a method for training an agent's policy network (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). The policy gradient method focuses on modeling and optimizing the policy directly, and decisions are made from sampling a probability distribution generated by the policy network. Given any state, the policy outputs a probability distribution over actions, and an action is randomly sampled from this distribution. A critical reason why we decided to use a policy gradient method is that since Point Robot is a continuous environment, it makes value-based approaches inefficient and computationally expensive.

The two main components in policy gradient are the policy model and the value function. For PPO, the Actor-Critic method uses the value function in addition to the policy to update the policy model, which reduces gradient variance. The Critic updates the value function parameters ω, while the Actor updates the policy parameters accordingly. The Actor-Critic network incorporates additional terms for value estimation error and entropy. The value function $V(s)$

computes the advantage as the difference between the expected return from taking an action a_t and state s_t. (Weng, 2018)

$$\delta = A_t = R_t + \gamma V(s_{t+1}) - V(s_t)$$

Then we perform Generalized Advantage Estimation (GAE), which is a popular method for estimating advantages and provides a balance between variance and bias by averaging over multiple estimates of advantages using weighted importance factors.

$$A_t = \sum_{t=0}^{\infty} (\gamma\lambda)^t \delta_{k+t}$$

PPO mitigates potential instability in policy updates by clipping the probability ratio to a predetermined hyper-parameter of ϵ around 1, which disincentive large updates that destabilize learning. The probability ratio between the new and the old policies is:

$$r(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

where a_t represents the action chosen by the policy, and s_t is the state of the environment at the current timestep t.

Empirical methods

Taking inspiration from the Eureka paper from NVIDIA (Ma et al., 2023), we utilized Large Language Models (LLMs) as high-level planners for sequential decision-making tasks. In particular, we wanted to focus on combining different reward functions to allow the agent to combine two separate goals to create a more complex goal. In order to achieve this, we utilized LLMs to write human-level reward design algorithms and saved the parameters that were deemed successful in retraining the agent with a new goal. For this particular task, we employed GPT-4-Turbo for in-context code generation and optimization over reward code in Jax. Utilizing this, we trained the point robot agent to traverse to the goal from its origin. Then, with the resulting rewards and network parameters, we trained the agent to complete a certain path of traversal before reaching its goal, such as a rectangle.

In our experiment, the LLM acted as a function from string to string and returned to use a subset of strings containing the reward functions for our environment. In order for the LLM to gain information on the context of the environment as a whole, we decided to feed the entire environment code so that it could have access to all of the environment parameters and state variables, allowing it to fully understand the behavior of the program. After feeding the information into the LLM, we passed in a string l that describes the composite reward we want it to complete, and the output of the LLM is a reward function $R(s, a)$ that maximizes the cumulative reward over time $\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$.

To maintain the previously successful behavior of the agent reaching the goal, we saved its parameters as a checkpoint by pickling the network parameters. These parameters were then loaded for use in subsequent training loops, allowing us to generate composite behavior from the agent.

To evaluate the code given by the LLM, we first evaluate whether the actual code was executable or not. Similar to what was done in the Eureka paper, we utilized Chain-of-Thought Prompting and saw that if we queried the LLM multiple times for one problem, the chance of the output being executable increases. (Wei et al., 2023) We would then use this executable code as our basis for the next queries in order to build in-context learning for the LLM. In order to update the LLM on whether its reward function worked as expected or not, we used 2 different approaches to query it. The first one was to give a visual description of what the agent was doing and how it was interacting with the environment and then based on that, the LLM would modify the reward function to make the agent move in the shape that we want it to. The other method was to record the rewards we were receiving from the environment at specific time steps, which were predetermined through our hyper-parameters, and supply it to the LLM in order to track the reward updates, and modify the reward functions to maximize the cumulative rewards. We used the second method when the agent was able to move in the shape we desired it to, whereas the first method was used for the initial steps in modifying the reward function.

Results

The initial behavior of the point robot environment had a reward function that made the agent traverse to the goal. As we can see from the learning plot below, the agent as able to converge at around 20000 updates and had a relatively consistent reward return rate. Visually, we were able to see the point robot agent move to the goal consistently using efficient routes.

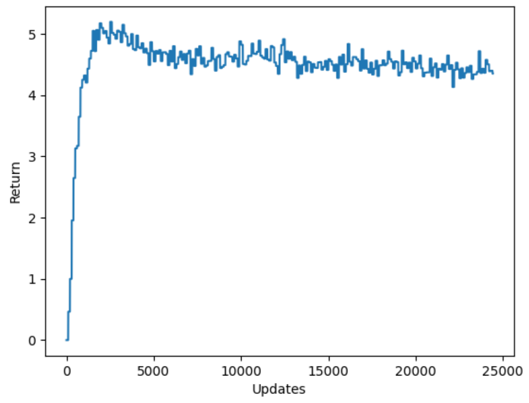

Figure 1: Initial Point Robot Environment Learning Curve

Afterward, we were able to generate a reward function through the LLM that allowed the agent to form a rectangle, despite not reaching the goal. We can see the learning plot for the reward function below, and can see that compared to traditional learning plots, it seems to spike up and remain stagnant for a large number of updates. Visually, we were again able to see the agent move in a rectangular path in quick intervals using pre-defined coordinates we set ourselves.

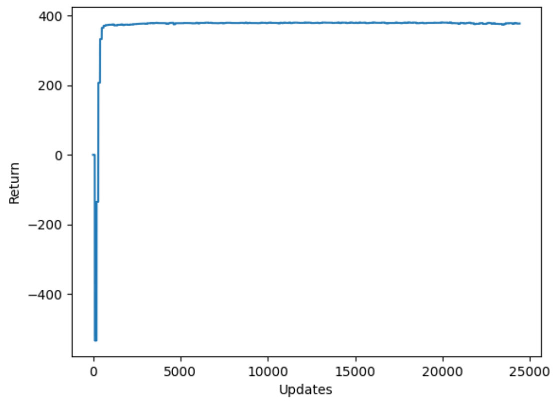

Figure 2: Rectangular Motion Point Robot Environment Learning Curve

From this, we can see that the LLM was successful in generating a novel reward function that goes against the original goal of the environment. This shows us its capabilities of understanding the environment as an entire context, and updating it to make the agent perform novel actions.

Therefore, given these two separate reward functions that generate successful behavior, we prompted the LLM to combine the two behaviors based on the fact the network parameters are pre-trained. From here, we encountered numerous different unexpected behaviors from the agents based on executable reward functions rewritten by the LLMs. The additional goal of making the agent traverse a rectangular trajectory before reaching the goal had a negative impact on the original goal since the trajectory of the rectangular path involved the agent moving against the pre-trained parameters. Thus, there were undesirable behaviors, such as the agent continuously moving away from the goal after attempting to fly in a rectangle, generating negative rewards. However, once we re-queried the LLMs and described the visual behavior and the rewards returned, we were able to receive new reward functions that improved the behavior of the agent, suggesting there are potential ways for the LLM to modify and generate successful reward functions that will achieve complex behaviors for the agent. We can see a learning curve for our most successful experiment below.

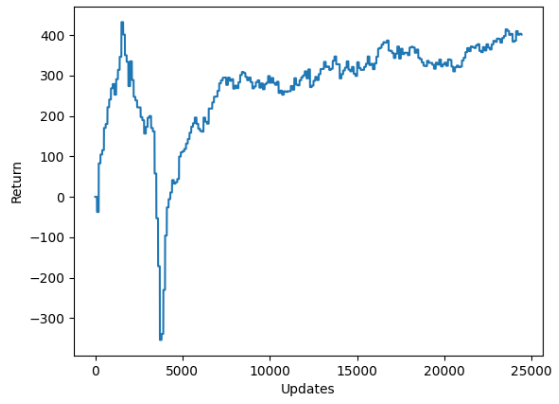

Figure 3: Composite Behavior Point Robot Environment Learning Curve

Conclusion

We proposed COMET as a way to have single agents learn composite reward functions through LLM reward function shaping. Our experiments showcased the adaptability of agents in varied environments and underscored the potential of LLMs to handle increasingly complex decision-making scenarios. Through our empirical methods, we observed that prompting LLMs to generate reward functions that deviate from the original task goals was feasible, resulting in agents exhibiting novel behaviors as seen by the learning curves and visual movements of the agent. While building composite reward functions challenges such as unintended behaviors and negative rewards, our results suggest avenues for further exploration in refining LLM-generated reward functions and optimizing agent behaviors. One possible path for future work is transferring these learning behaviors to robotic agents to see the effect of knowledge transfer from a simulation to the real world, similar to (Ma et al., 2024). Furthermore, another possible path to explore is to transfer this methodology to multi-agent systems to address more complex behavior and reward functions needing the collaboration of many agents towards one goal. Additionally, in order to achieve the combined reward function, we could expand our reward function shaping by adding weights to the different reward functions. For instance, the reward function generated by the LLM can be of the form:

$$R(s,a) = \alpha * R_{rectangle}(s,a) + \beta * R_{goal}(s,a)$$

where α and β are the weights associated with the specific reward functions we are trying to combine (R_{goal} is the reward function of going to the goal and $R_{rectangle}$ is the reward function of going in a rectangular path). Through this, we can achieve the desired effect by optimizing the weights of the individual reward functions in the composite reward.

Appendix

Queries

Query:

Context: The agent's network parameters are already trained to reach the goal. Please allow the goal of my model to be preserved while it moves in a rectangular motion by modifying my reward function: (Environment and current reward function provided)

Context: The agent does not move in a rectangular motion and does not reach the goal. Please modify the reward function so that it traverses in a rectangular trajectory once before reaching the goal. (Environment and previously queried reward function provided)

Context: The agent, while moving in a rectangular motion, traverses off the screen and accumulates negative reward. Ensure that the agent remains within the bounds of the window and reach the goal. (Environment and previously queried reward function provided)

Context: The agent is able to reach its goal again. Please provide an additional reward if the agent completes one full rectangle rather than multiple incomplete rectangles while reaching the goal. The code must be executable through Jax. (Environment and previously queried reward function provided)

Environment layout

The following figure shows the layout of the point robot environment and agent with no modifications from the Gymnax library.

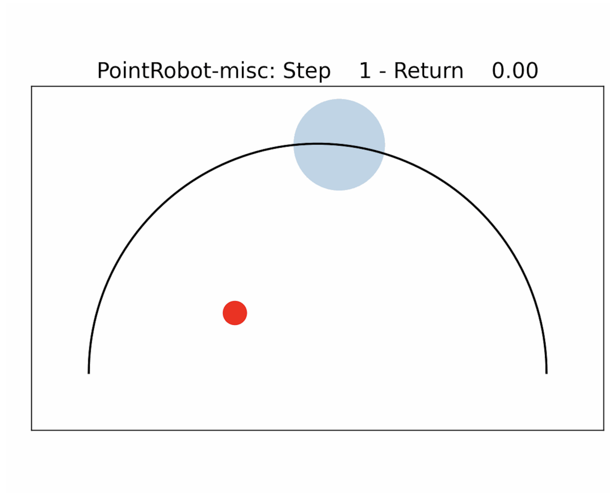

Figure 4: Point Robot Environment

Model Hyperparameters

Table 1: Model Configuration

Hyperparameter	Value
Learning rate	3e-4
Number of environments	2048
Number of steps	10
Total Timesteps	5e7
Update Epochs	4
Number of Minibatches	32
Discount factor	0.99
Gae Lambda	0.95
Clip epsilon	0.2
Entropy coefficient	0.0
Value function coefficient	0.5
Max gradient norm	0.5
Activation function	tanh
Environment name	PointRobot-misc
Anneal Learning Rate	False
Normalize environment	True
Debug mode	True

References

Dorfman, R., Shenfeld, I., & Tamar, A. (2021). Offline meta reinforcement learning – identifiability challenges and effective data collection strategies. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (Vol. 34, pp. 4607–4618). Curran Associates, Inc.

Hazra, R., Dos Martires, P. Z., & De Raedt, L. (2024). Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 38, pp. 20123–20133).

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., … Anandkumar, A. (2023). *Eureka: Human-level reward design via coding large language models.*

Ma, Y. J., Liang, W., Wang, H., Wang, S., Zhu, Y., Fan, L., … Jayaraman, D. (2024). Dreureka: Language model guided sim-to-real transfer.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.*

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., & Su, Y. (2023, October). Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the ieee/cvf international conference on computer vision (iccv).*

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* Cambridege, Massachusetts: MIT Press.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., … Zhou, D. (2023). *Chain-of-thought prompting elicits reasoning in large language models.*

Weng, L. (2018). *Policy gradient algorithms.* Retrieved from `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`

Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Gonzalez Arenas, M., … Xia, F. (2023). Language to rewards for robotic skill synthesis. *Arxiv preprint arXiv:2306.08647*.