Reward Copilot for RL-driven Systems Optimization

Karan Tandon
Microsoft Research, India
karantandon@microsoft.com

Manav Mishra
IISER Bhopal, India
mishra20@iiserb.ac.in

Gagan Somashekar
Microsoft365 Research, USA
gsomashekar@microsoft.com

Mayukh Das
Microsoft365 Research, India
mayukhdas@microsoft.com

Nagarajan Natarajan
Microsoft Research, India
nagarajn@microsoft.com

Abstract

Systems optimization problems such as workload auto-scaling, kernel parameter tuning, and cluster management arising in large-scale enterprise infrastructure are becoming increasingly RL-driven. While effective, it is difficult to set up the RL framework for such real-world problems – designing correct and useful reward functions or state spaces is highly challenging and needs a lot of domain expertise. Our proposed novel REWARD COPILOT solution can help design suitable and interpretable reward functions guided by client-provided specifications for any RL framework. Using experiments on standard benchmarks as well as systems-specific optimization problems, we show that our solution can return reward functions with a certain (informal) feasibility certificate, in addition to pareto-optimality.

1 Introduction

Systems optimization problems such as workload auto-scaling [Luo et al., 2022, Yang et al., 2013, 2014, Rzadca et al., 2020], kernel parameter tuning [Akgun et al., 2020], and cluster resource and configuration management [Aron et al., 2000, Radiskhlebova et al., 2019, Maurer et al., 2013] arising in large-scale enterprise infrastructure are becoming increasingly RL-driven [Somashekar et al., 2024, Karthikeyan et al., 2023]. The success of RL-driven systems optimization crucially hinges on designing a single reward function that involves competing objectives (e.g., throughput and latency) and feedback from multiple measurements [Grzes, 2017, Hu et al., 2020, Goyal et al., 2019], which often requires domain expertise.

Large Language Models (LLMs) are capable of generating complex code, and have recently been applied in the context of generating reward functions [Yu et al., 2023, Ma et al., 2023, Xie et al., 2024]. Text2Reward [Xie et al., 2024] uses LLMs to effectively create dense, interpretable reward programs by transforming task descriptions into reward structures. However it relies on few-shot learning and human-guided refinement limiting its applicability in more complex systems tasks. Another approach, Eureka [Ma et al., 2023] takes an evolutionary optimization approach combined with LLM-driven generation, enabling zero-shot reward design. While effective on many RL benchmarks, deploying Eureka in real-world systems is difficult due to the need for domain-specific fitness functions, which is as challenging as reward design itself. In real deployments, it is more practical and consistent for clients to provide requirements or problem specifications (specs) that qualify the nuances and constraints of the environment and target metrics.

We propose REWARD COPILOT, a framework that directly incorporates client-provided specifications (specs) to guide generation and reflective refinement of interpretable reward functions that lead to superior policies that are both Pareto-optimal and have feasibility certificates that we demonstrate, via extensive evaluations on standard RL benchmarks and real systems optimization tasks.

Machine Learning for Systems Workshop at (NeurIPS 2024).

2 Methodology

Problem Setting. The reward design problem (RDP), defined by [Barto et al., 2009], aims to return a shaped reward function for a ground-truth reward function that may be challenging to design or optimize directly. Formally, RDP as a tuple $P = \langle M, \Re, \pi_M \rangle$, where $M = (S, A, T)$ represents the world model, S and A the state and action spaces resp., and T the transition function. The space of reward functions $\Re$ is mapped to policies Π via a learning algorithm $\Psi_M() : \Re \rightarrow \Pi$, which induces a policy $\pi \in \Pi$ that optimizes reward $r \in \Re$ in the associated Markov Decision Process (MDP) (M, r). Assuming $\exists \mathcal{G}$ a global quality metric, the RDP's objective is to design a reward function $r \in \Re$ that produces a policy $\pi^*_r \in \Pi$ which maximizes the global quality metric $\mathcal{G}$; $\pi^*_r \underset{\mathcal{G}}{\succeq} \pi^*_{\Re \setminus r}$.

2.1 The Reward Copilot Framework

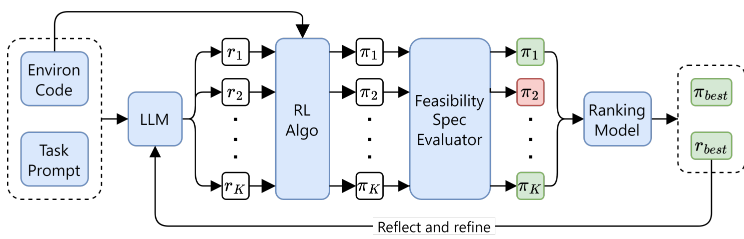


Figure 1: Reward Copilot pipeline operates in three stages – Reward candidate generation: produces candidate reward function code and trains respective policies, Specs Evaluation: filters candidates for spec compliance and ranks them, & Prompt Reflection: refines the generation process.

Our Reward Copilot framework (Alg 1) aims to intelligently design and recommend interpretable and (near) optimal reward functions, through an iterative three-step process: Reward candidate generation, specs evaluation, and prompt reflection (Figure 1). These steps work together to iteratively refine the reward code. Though [Ma et al., 2023] is conceptually similar, yet instead of relying on a predefined fitness function F to evaluate reward function quality, we employ the feasibility and optimality specs checker. This component approximates a surrogate fitness function, eliminating the need for users to manually define it, which can be challenging in system-level optimization. The LLM starts by ingesting the task description τ, a Pythonic environment code $\mathfrak{E}$ to set the syntax and context, and an initial prompt $\mathcal{P}$ to initiate the code generation. Note that, environment source code often reveals the structure of the process flow and variable, which may help with reward generation. For cases where a self-contained code is not available, the APIs can be leveraged to estimate the structure of variables.

Algorithm 1: Reward Copilot algorithm

Input: Task description τ, environment code $\mathfrak{E}$, initial task prompt $\mathcal{P}$, Feasibility & Optimality Specs $\mathfrak{S}_f$, $\mathfrak{S}_o$
Output: Optimal reward function $\mathcal{R} \in \Re$
Initialize LLM $\mathcal{L} \leftarrow \tau, \mathfrak{E}$.
repeat{// Reward candidate generation}
  Generate k reward function candidates $\{r_1, \ldots, r_k\} \leftarrow \mathcal{L}_{\tau,\mathfrak{E}}(\mathcal{P})$.
  Evaluate candidates using specs. $\mathcal{R}, \mathbf{R}^C \leftarrow$Ranker$(\{r_1, \ldots, r_k\}, \mathfrak{S}_f, \mathfrak{S}_o)$ {Specs Evaluation (Alg 2); $\mathbf{R}^C$ = fine-grained traces of individual reward components}
  Reflect on results and refine LLM prompt. $\mathcal{P} \leftarrow Refine(\mathcal{P}, \mathcal{R}, \mathbf{R}^C)$ {Prompt reflection}
until an optimal reward function is found or convergence is achieved

Reward candidate generation. The LLM $[\mathcal{L}_{\tau,\mathfrak{E}}(\mathcal{P})]$ then generates k independently sampled candidate reward functions $\{r_i\}_1^k \in \Re$. Independently sampling make having syntactic errors in all of them highly unlikely and enhances the diversity among the candidates as well, thereby increasing the likelihood of finding *valid* reward functions. We then train k policies for the subsequent modules.

Specs Evaluation. Post-training, the learned policies $\{\pi^*_1, \ldots, \pi^*_k\}$ are evaluated on metrics M induced from user-provided specs $(\mathfrak{S}_f, \mathfrak{S}_o)$. The metrics are tracked throughout the policy learning process to avoid overhead from real-world system deployments. This is analogous to the approach in [Yang et al., 2019], which involves a learning and adaptation phase to optimize policy performance.

Feasibility spec evaluator. This phase involves a filtration of those policies that satisfy a minimum feasibility with regard to the system specs provided by the user. This is done with the intention to filter out those candidates that satisfy some minimum spec eligibility. In the feasibility spec checker, each policy is evaluated using a score function $S : M \to [0, 1]$, where a score of 1 is assigned if no specification violations occur and 0 otherwise. Policies with an average score exceeding a predefined *feasibility threshold* m are retained for further evaluation. For this work, a feasibility threshold of 0.70 is employed, ensuring that only k' policies meeting the minimum constraint satisfaction criteria advance to the optimality specifications phase.

RANKER *model.* The filtered candidate reward functions are evaluated to identify the *relatively* optimal reward configuration using a proposed rank-based preference algorithm. Initially, users assign a preference rank to each metric they wish the system to prioritize. Once feasibility constraints are established, the algorithm selects the optimal candidate among the feasible options. This approach operates by comparing each specification metric in a pairwise fashion based on user-provided rankings and formulates the problem as a two-player game (Alg. 2 in Appendix A). The WIN-SELECTOR function evaluates and compares candidates based on their ranked metrics with an $n\%$ margin. For more detailed steps see Alg 3 in Appendix A. The winner candidate reward function from RANKER, is said to be a surrogate for the fitness function, without the reliability on the user to provide a heuristic fitness function. This best candidate reward function is then used to generate a feedback prompt for further LLM refinement through the prompt reflection step.

Prompt reflection. In order to ground the reward function to the LLM, the best reward candidate reward function must be able to put into words the quality of the generated rewards. The selected reward function undergoes further refinement through prompt reflection. The specs evaluation step serves as a measure of a *relative* ground truth in order to assess and determine the best reward function in that evolution/iteration step. Textual feedback on policy performance and reward component dynamics is automatically generated, providing insights into how well the reward function aligns with system objectives. This is done by leveraging intermediate policy checkpoints to track and evaluate the individual components of the reward function during the policy training, leading to adjustments that improve reward synthesis in subsequent iterations. This feedback reflection process is important for the reward optimization process, as it offers the LLM module insights into how well the RL algorithm optimizes individual reward components. The Spec checker, ranker and the prompt reflection steps together implicitly nudge us closer to the pareto-front. This enables Reward Copilot to produce more precise reward edits and develop reward functions aligned with the application.

3 Evaluation Setup

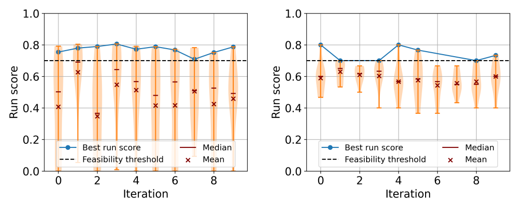
We evaluate REWARD COPILOT on two test environments—a Gymnasium [Towers et al., 2024] environment and a systems benchmark—against the state-of-the-art Eureka baseline and hand-crafted reward functions. We use Stable Baselines Jax (sbx) Raffin et al. [2021] for policy training.

Redis with Yahoo Cloud Serving Benchmark (YCSB). Redis [Redis, 2024] with YCSB [Cooper et al., 2010] evaluates the system's efficiency in a real-world cloud-serving environment. Configuration tuning of Redis is challenging due to its different use cases, parameter inter-dependencies and their sensitivity to workload, or multiple (conflicting) objectives like throughput, latency, memory efficiency etc. Manually crafting a suitable reward function (or even a fitness function) is extremely challenging. We tune four integer parameters of Redis, namely `maxmemory`, `maxmemory-samples`, `zset-max-ziplist-entries` and `hz` to minimize the time taken to process the workload generated by YCSB (see Appendix B.1 for more details where Fig. 5 shows the spec chart for Redis.)

Classic CartPole problem. The CartPole problem is a standard benchmark in reinforcement learning where the goal is to balance a straight pole on a moving cart. It comprises a continuous state space, represented by 4 variables: cart position, cart velocity, pole angle, and pole angular velocity. The action space is discrete, consisting of two actions: applying a force in either direction, left or right. The objective is to maximize the time the pole remains upright by keeping the pole angle within a specified range. We consider three metrics for spec evaluation, namely `duration`, `cart displacement`, `pole displacement` (see Appendix B.2 for more details). Figure 6 shows the spec chart being used for CartPole.

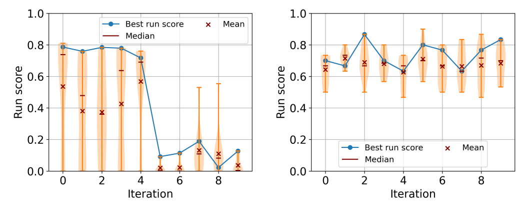
4 Results

To evaluate the effectiveness of REWARD COPILOT in skill and reward learning, we benchmark its performance against a human-designed reward and Eureka's approach. For the CartPole environment, three normalized specifications were considered: episode duration, x-displacement of the cart, and pole angle displacement over the entire episode. These metrics were ranked based on preferences, as illustrated in Figure 2(a). For the Redis-YCSB benchmark, the primary metric used was the mean throughput per episode, as shown in Figure 2(b). REWARD COPILOT's performance evolution across both benchmarks is demonstrated by tracking the iteration progress of the reward functions. At each iteration step, $k = 10$ candidate reward functions are sampled and subjected to a feasibility check, requiring a minimum specification score of $m = 0.7$ (i.e., a 70% threshold) during training.



(a) CartPole results (b) Redis results

Figure 2: REWARD COPILOT results



(a) CartPole results (b) Redis results

Figure 3: Eureka results

Figure 2 illustrates the distribution of candidate rewards for Cartpole and Redis using a violin plot for the reward copilot framework, with the black dotted line marking the feasibility threshold. Reward candidates exceeding this threshold, denoted by k', are selected for ranking. The line plot depicts the highest-performing reward function at each iteration, which informs the final performance prompt. Figure 3 highlights Eureka's performance, which is evaluated for specification satisfaction. Unlike REWARD COPILOT, Eureka's results do not feature a feasibility threshold, as it is not used in their framework. The analysis shows a decline in Eureka's reward quality when specification satisfaction is considered during training, attributed to the Eureka's lack of explicit handling of specification violations. Despite not having direct access to the fitness function, REWARD COPILOT achieves comparable reward quality (see Appendix C for the generated reward function codes).

We compare REWARD COPILOT's performance against 2 baselines - (1) Eureka's reward function and (2) human-designed reward across 100 evaluation runs on the generated reward functions. For CartPole, we consider the environment's standard reward as the human-designed one, while for Redis, we use a linear combination of latency and throughput. Figure 4 compares REWARD COPILOT and Eureka, using human-designed rewards as the basis. Top plot (1) displays the human-normalized score on the fitness function for both CartPole and Redis. It highlights REWARD COPILOT's ability to generate effective reward functions without explicit dependence on a fitness function. Bottom Plot (2) compares the mean *specs* satisfaction rates between then REWARD COPILOT significantly outperforms Eureka in this case, underlining the limitation of Eureka in adhering to client specs. LLM-generated code plays an indispensable role in smartly identifying complex assimilation of attributes and bounds into the reward function compared to beyond what traditional human design can do, as evident in tasks like Redis, where REWARD COPILOT can lead to higher quality rewards and better spec satisfaction.

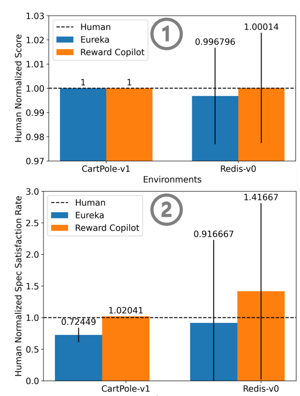


Figure 4: Comparison among Reward Copilot, Eureka and Human-designed rewards of human-normalized (1) scores evaluated on common fitness fn. (2) specs satisfaction rate.

Conclusion. We present REWARD COPILOT to automatically design interpretable reward functions for RL-driven system optimization problem specifications. REWARD COPILOT can seamlessly design reward functions, which helps learn policies that are both Pareto-optimal and have a feasibility certificate without the limiting assumptions of existing approaches. Additionally, fine-tuning the LLM on a range of systems problems can further enhance its ability to generate contextually relevant reward functions, potentially leading to improved optimization outcomes. Preliminary evaluations demonstrate that REWARD COPILOT is comparable or better than all baselines. Assessing across a variety of Gymnasium and MuJoCo environments, and expanding to other system benchmarks are immediate next steps.

References

I. U. Akgun, A. S. Aydin, and E. Zadok. KMLib: Towards machine learning for operating systems. In *Proceedings of the 2020 On-Device Intelligence Workshop, co-located with the MLSys Conference*, February 2020. Refereed abstract+poster.

M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 90–101, 2000.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5): 834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

A. G. Barto, R. L. Lewis, and S. Singh. Where do rewards come from. 2009. URL `https://api.semanticscholar.org/CorpusID:14951500`.

B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300360. doi: 10.1145/1807128.1807152. URL `https://doi.org/10.1145/1807128.1807152`.

P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

M. Grzes. Reward shaping in episodic reinforcement learning. 2017.

Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

A. Karthikeyan, N. Natarajan, G. Somashekar, L. Zhao, R. Bhagwan, R. Fonseca, T. Racheva, and Y. Bansal. {SelfTune}: Tuning cluster managers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1097–1114, 2023.

J. Liu, Y. Ma, C. Chen, and Z. Yu. Atr: Auto-tuning configurations of redis via ensemble learning. In *2020 6th International Conference on Big Data and Information Analytics (BigDIA)*, pages 104–112, 2020. doi: 10.1109/BigDIA51454.2020.00025.

S. Luo, H. Xu, K. Ye, G. Xu, L. Zhang, G. Yang, and C. Xu. The power of prediction: microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 355–369, 2022.

Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

A. Mahgoub, P. Wood, A. Medoff, S. Mitra, F. Meyer, S. Chaterji, and S. Bagchi. SOPHIA: Online reconfiguration of clustered NoSQL databases for Time-Varying workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 223–240, Renton, WA, July 2019. USENIX Association. ISBN 978-1-939133-03-8. URL `https://www.usenix.org/conference/atc19/presentation/mahgoub`.

A. Mahgoub, A. M. Medoff, R. Kumar, S. Mitra, A. Klimovic, S. Chaterji, and S. Bagchi. OPTIMUSCLOUD: Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 189–203. USENIX Association, July 2020. ISBN 978-1-939133-14-4. URL `https://www.usenix.org/conference/atc20/presentation/mahgoub`.

M. Maurer, I. Brandic, and R. Sakellariou. Adaptive resource configuration for cloud infrastructure management. *Future Generation Computer Systems*, 29(2):472–487, 2013.

A. A. Radiskhlebova, A. B. Vavrenyuk, A. S. Rusakova, and V. V. Makarov. Study of the possibilities of using virtualization tools to optimize the cluster resources management. In *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 310–314. IEEE, 2019.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Redis. Redis: A persistent memory database, 2024. URL `https://redis.io`. Accessed: 2024-10-28.

K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, et al. Autopilot: workload autoscaling at google. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

G. Somashekar, A. Suresh, S. Tyagi, V. Dhyani, K. Donkada, A. Pradhan, and A. Gandhi. Reducing the tail latency of microservices applications via optimal configuration tuning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 111–120, 2022. doi: 10.1109/ACSOS55765.2022.00029.

G. Somashekar, K. Tandon, A. Kini, C.-C. Chang, P. Husak, R. Bhagwan, M. Das, A. Gandhi, and N. Natarajan. OPPerTune: Post-Deployment configuration tuning of services made easy. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1101–1120, Santa Clara, CA, Apr. 2024. USENIX Association. ISBN 978-1-939133-39-7. URL `https://www.usenix.org/conference/nsdi24/presentation/somashekar`.

M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=tUM39YTRxH`.

J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen. Workload predicting-based automatic scaling in service clouds. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 810–815. IEEE, 2013.

J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16:7–18, 2014.

R. Yang, X. Sun, and K. Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/4a46fbfca3f1465a27b210f4bdfe6ab3-Paper.pdf`.

W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 415–432, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450356435. doi: 10.1145/3299869.3300085. URL `https://doi.org/10.1145/3299869.3300085`.

A Additional Method Details

The RANKER algorithm ranks multiple reward functions based on user-defined specifications. The procedure for the RANKER algorithm is detailed in Algorithm 2. Using the WIN-SELECTOR module, it executes a pairwise round-robin tournament to determine the best reward function.

Algorithm 2: RANKER - Rank Preference-Based Reward Selection

Input: $\{r_1, \ldots, r_k\}$ candidates, Specs $\mathfrak{S}_f, \mathfrak{S}_o$
Output: Optimal reward function $\mathcal{R}$
$\{\pi_1, \ldots, \pi_k\}, \{\mathbf{R}_1^C, \ldots, \mathbf{R}_k^C\} \leftarrow$ Train policies for $\{r_1, \ldots, r_k\}$ $\{\mathbf{R}_i^C$ = Reward component traces$\}$
// Each candidate competes against all others in a round-robin fashion.
for each pair of candidates (r_i, r_j) **do**
  Use WIN-SELECTOR$(r_i, r_j, \pi_i, \pi_j, \mathfrak{S}_f, \mathfrak{S}_o)$ to determine the winner between them. {(Alg. 3)}
end for
$\mathcal{R} \leftarrow$ candidate with the highest score (maximum wins) is selected as the optimal reward.
return $\mathcal{R}, \mathbf{R}_{\mathcal{R}}^C$

The WIN-SELECTOR module receives a list of metrics ranked by importance, as specified by the user. For each metric, it assesses dominance using a threshold margin $n\%$: if one reward function surpasses another by at least $n\%$, it is deemed dominant, and that reward function is marked as the winner. If no dominance is established, the comparison proceeds to the next metric in the hierarchy, continuing until a winner is determined.

Algorithm 3: WIN-SELECTOR

Input: Candidates r_a and r_b, Specs $\mathfrak{S}_f, \mathfrak{S}_o$, Policies π_a, π_b, Quality margin n
Output: Winner between (r_a, r_b)
Compute ranked metrics for r_x; $M_x = [m_1^{(x)}, m_2^{(x)}, m_3^{(x)}, \ldots] \leftarrow$ SATISFYSPEC$(\pi_x, \mathfrak{S}_f, \mathfrak{S}_o)$
for each metric dimension m_i in M **do**
  if $m_i^{(a)}$ is better than $m_i^{(b)}$ by at least $n\%$ **then**
    return r_a as the winner
  else if $m_i^{(b)}$ is better than $m_i^{(a)}$ by at least $n\%$ **then**
    return r_b as the winner
  else
    Proceed to evaluate the next metric m_{i+1}
  end if
end for

B Test Domains and Environment Designs

B.1 Redis

Redis is an open-source, in-memory data structure store that can be used as a database, cache, and message broker. Configuration tuning of Redis is challenging due to its different use cases, parameters' sensitivity to workload, and parameter interdependencies. Many prior works tackle this challenge by employing machine learning and optimization algorithms [Mahgoub et al., 2019, Liu et al., 2020, Mahgoub et al., 2020, Somashekar et al., 2022]. However, Redis' falls into the category of systems whose performance involves multiple, often conflicting objectives like throughput, latency, memory efficiency, data persistence and consistency. Hence, framing a reward function that captures all these objectives is challenging. Moreover, some configuration changes may provide immediate performance benefits but lead to long-term issues (e.g., increased memory fragmentation). Prior works tackle this by handcrafted reward functions which only address some of these objectives [Zhang et al., 2019, Mahgoub et al., 2020]. YCSB [Cooper et al., 2010] is a widely used database benchmarking suite that provides various workloads that simulate different real-world scenarios, making it suitable for evaluating Redis in different configurations.

```yaml
env_id: Redis-v0
description: to reduce execution time of the database workload
items:
  - name: mean_throughput
    desc: The average rate at which Redis processes the workload (in
    ↪  requests/second) during the episode
    min: 9500
    max: null
    aim: maximize
    rank: 0
```

Figure 5: Spec for Redis

```yaml
env_id: CartPole-v1
description: to balance a pole on a cart so that the pole stays upright
items:
  - name: normalized_episode_length
    desc: The number of steps for which the pole stays upright in an
    ↪  episode, normalized by the maximum episode length
    min: 0.85
    max: 1.0
    aim: maximize
    rank: 0
  - name: normalized_abs_delta_theta
    desc: Sum of all theta deviations of the pole in an episode,
    ↪  normalized by the maximum episode length
    min: 0.0
    max: 0.03
    aim: minimize
    rank: 1
  - name: normalized_abs_displacement
    desc: Sum of all x-displacements of the cart in an episode, normalized
    ↪  by the maximum episode length
    min: 0.0
    max: 0.05
    aim: minimize
    rank: 2
```

Figure 6: Spec for CartPole

B.2 CartPole

The CartPole environment is a classical benchmark in reinforcement learning (RL), originally introduced in [Barto et al., 1983]. It simulates a cart moving along a track with a pole attached to it. The goal of the agent is to balance the pole by applying forces to the cart to keep the pole upright for as long as possible. The environment is simple yet effective for testing RL algorithms due to its continuous action space and deterministic dynamics. In this environment, the agent receives a reward of +1 for every time step the pole remains upright, and the episode terminates when the pole falls past a certain angle threshold or the cart moves beyond the bounds of the track. This makes the task episodic, where the agent seeks to maximize the cumulative reward by maintaining balance. The challenge in CartPole lies in learning the optimal control policy to maintain balance, making it a foundational environment for testing policy-based, value-based, and hybrid RL algorithms. Although it represents a single-objective task, it forms the basis for more complex multi-objective environments where trade-offs between competing objectives, such as stability (by measuring the angular displacement of the pole) and energy efficiency (by measuring the linear displacement of the cart), are introduced.

C Examples of generated reward functions

This section provides examples of generated reward functions for Redis and CartPole.

```python
def compute_reward(self) -> Tuple[float, Dict[str, float]]:
    """
    Compute reward for reducing execution time of the database workload.

    Reward is calculated based on the instantaneous ops per second (throughput)
    and the total error replies. A higher throughput and lower error replies result
    in a higher reward.
    """

    # Define temperature variables for transformation functions
    throughput_temperature = 100.0

    # Calculate individual reward components
    throughput_reward = self.redis_server_metrics["instantaneous_ops_per_sec"] / throughput_temperature
    error_replies_reward = -self.redis_server_metrics["total_error_replies"]

    # Combine individual rewards into a total reward
    total_reward = throughput_reward + error_replies_reward

    return total_reward, {
        "throughput_reward": throughput_reward,
        "error_replies_reward": error_replies_reward,
    }
```

Figure 7: Redis - Reward function

```python
def compute_reward(self) -> Tuple[float, Dict[str, float]]:
    """
    Compute reward for CartPoleEnv.

    The goal is to balance a pole on a cart so that the pole stays upright.
    We use a combination of rewards to encourage the agent to keep the pole upright and the cart
    ↪ centered.

    :return: total_reward, reward_components
    """

    # Reward for keeping the pole upright
    pole_angle_temperature = 10.0
    pole_angle_reward = np.exp(-self.state[2]**2 / pole_angle_temperature)  # Re-write reward component
    ↪ to provide more nuanced feedback

    # Reward for keeping the cart centered
    cart_position_scale = 0.1
    cart_position_reward = -cart_position_scale * abs(self.state[0])  # Re-scale value to a proper range

    # Reward for survival
    survival_reward = 0.10 if self.state[2] < 0.2095 else 0.00  # Keep the reward component as it is
    ↪ written

    # Total reward
    total_reward = pole_angle_reward + cart_position_reward + survival_reward

    return total_reward, {
        'pole_angle_reward': pole_angle_reward,
        'cart_position_reward': cart_position_reward,
        'survival_reward': survival_reward
    }
```

Figure 8: CartPole - Reward function

D Details of Implementation Assets

Following are the details regarding the assets that we have used in our implementation:

Asset	**Version**	**License**
Gymnasium	0.29.1	MIT
Stable Baselines Jax (sbx)	0.17.0	MIT
YCSB	0.17.0	Apache-2.0
Redis	7.4.0	Redis Source Available License 2.0 (RSALv2)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper claims and motivates the scope and contribution of our work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [No]

   Justification: The paper presents preliminary results and is still a work in progress.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: This work is application-focused and does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The environments, specs, feasibility threshold, workloads and the solution pipeline have been described in section 3 of the main paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: This is still a work in progress. We are working on making the evaluation even more robust and hope to release the code post that.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details are covered in section 3 of the main paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The results are presented either as violinplots or bar plots with standard deviation bars.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: This work-in-progress paper presents preliminary results, with full resource details to be included in the final conference submission.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This is an early-stage, work-in-progress research study.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the assets used in the primary text in Appendix D.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This work does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.