Balancing Act: Prioritization Strategies for LLM-Designed Restless Bandit Rewards

Shresth Verma
Harvard University

Niclas Boehmer
Harvard University

Lingkai Kong
Harvard University

Milind Tambe
Harvard University

Abstract

LLMs are increasingly used to design reward functions based on human preferences in Reinforcement Learning (RL). We focus on LLM-designed rewards for Restless Multi-Armed Bandits, a framework for allocating limited resources among agents. In applications such as public health, this approach empowers grassroots health workers to tailor automated allocation decisions to community needs. In the presence of multiple agents, altering the reward function based on human preferences can impact subpopulations very differently, leading to complex tradeoffs and a multi-objective resource allocation problem. We are the first to present a principled method termed *Social Choice Language Model* for dealing with these tradeoffs for LLM-designed rewards for multiagent planners in general and restless bandits in particular. The novel part of our model is a transparent and configurable selection component, called an *adjudicator*, external to the LLM that controls complex tradeoffs via a user-selected social welfare function. Our experiments on a mobile health program dataset demonstrate that we reliably select more effective, aligned, and balanced reward functions compared to purely LLM-based approaches.

1 Introduction

Reward functions play a fundamental role in the generation of optimal policies for sequential decision-making via reinforcement learning. Previous work has shown that LLMs are an effective tool for designing reward functions that can be guided and customized via human language prompts [12, 7, 11, 27, 13, 28, 9]. We study the problem of designing high-quality reward functions aligned with human preference prompts in the context of sequential resource allocation problem in mobile health. We present a transparent framework around LLMs that constructs effective, aligned, and balanced reward functions for complex human prompts.

Specifically, our work studies the reward design problem for restless multi-armed bandits (RMABs), a popular model in multiagent systems for sequentially allocating a limited number of resources to a set of agents [26, 17]. In RMABs, each agent is represented by an individual Markov Decision Process including a reward function. By choosing these reward functions, one can control which agents are more or less likely to receive a resource. RMABs have been used in various domains such as machine maintenance [1], anti-poaching [19], and healthcare [5, 24]. In many of them, system organizers have evolving allocation priorities based on agents' features that need to be incorporated into the resource allocation process [8, 25]. For instance, in a healthcare program, a healthcare worker might want to change the allocation policy to prioritize low-income beneficiaries who are at higher risk or older beneficiaries who have transportation barriers for healthcare access [16, 23] via the preference prompt: *Prioritize low-income beneficiaries and older beneficiaries*.

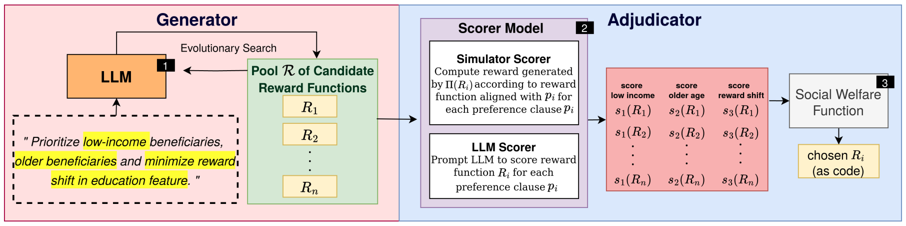
Generator
Evolutionary Search
LLM
1
Pool $\mathcal{R}$ of Candidate Reward Functions
R_1
R_2
R_n
" Prioritize low-income beneficiaries, older beneficiaries and minimize reward shift in education feature. "
Adjudicator
Scorer Model
2
Simulator Scorer
Compute reward generated by $\Pi(R_i)$ according to reward function aligned with p_i for each preference clause p_i
LLM Scorer
Prompt LLM to score reward function R_i for each preference clause p_i
score low income
score older age
score reward shift
$s_1(R_1)$ $s_2(R_1)$ $s_3(R_1)$
$s_1(R_2)$ $s_2(R_2)$ $s_3(R_2)$
$s_1(R_n)$ $s_2(R_n)$ $s_3(R_n)$
Social Welfare Function
3
chosen R_i (as code)

Figure 1: Overview of SCLM. Step 1: preference prompt is passed to the generator, which performs evolutionary search to create a pool $\mathcal{R}$ of candidate reward functions. Step 2: these functions are passed to the adjudicator where a scorer model (simulator/LLM scorer) computes alignment scores. Step 3: a user-defined social welfare function selects a reward function based on the alignment scores.

Translating such human language prompts to effective and aligned reward functions is a general, non-trivial challenge in RL [9, 7, 12], e.g., as it is unclear how changes in the reward influence the used policy downstream. However, the multiagent nature of the RMAB problem adds a new twist to the problem of reward design in RL: It becomes fundamentally multi-objective. Consider the above example prompt asking for the prioritization of two subpopulations. As these subpopulations may contain different agents, selecting a reward function will most likely involve trading off the interests of the low-income vs. older beneficiaries, making this a multi-objective problem. If this multi-objective nature is ignored, a selected reward function might heavily favor one of the two groups (e.g., leading to the allocation of many resources to low-income beneficiaries, and no resources to older ones).

We present a Social Choice Language Model (SCLM) that designs reward functions aligned with complex, multi-objective human language preferences in a mobile health program (see Figure 1). Our pipeline separates generation of candidate reward functions in the *generator* from the selection of one function in the *adjudicator*. *SCLM combines the generative power of LLMs to design reward functions with capabilities of social choice to handle multi-objective decision-making scenarios.*

2 Preliminaries

An instance of Restless Multi-Armed Bandits (RMAB) is defined by a set of N arms (or agents), a time horizon T, and a budget K. Each arm $i \in [N]$ is an independently evolving MDP with state space $\mathcal{S}_i$, actions $\mathcal{A}_i = \{0, 1\}$, transition function $P_i : \mathcal{S}_i \times \mathcal{A}_i \times \mathcal{S}_i \rightarrow \mathbb{R}_{\geq 0}$, and reward function $R_i : \mathcal{S}_i \rightarrow \mathbb{R}$. We refer to 1 as the *active* action corresponding to pulling the arm (i.e., allocating a resource) and 0 as the *passive* action corresponding to not pulling the arm. We focus on MDP with two states, i.e., $\mathcal{S}_i = \{0, 1\}$ for all $i \in [N]$ where 0 is the *bad* and 1 is the *good* state. For each step in which an agent is in the good state, they derive a *utility* of 1, while they derive a utility of 0 in the bad state. Accordingly, agents' *default reward function* R^* is $R^*(s) = s$. We assume that there is a set of categorical features. Each arm is associated with a value of each feature. A *global reward function* is a reward function defined over features, which induces a reward function for each arm by plugging in its feature values.

In each step within time horizon T, the planner observes the state of all arms and decides to pull a subset of at most K arm. As solving the RMAB problem optimally is computationally intractable [18], we make use of the very popular state-dependent Whittle index [26, 17], which tries to quantify for each state of each arm the reward gain achieved from applying the active action to the arm in this state. In the Whittle index policy Π, in each step, we compute Whittle index for each arm (based on its current state) and pull arms with the K highest Whittle indices. For a global reward function R, we write $\Pi(R)$ to denote the Whittle index policy for R, i.e., the Whittle index policy for the instance where each agent uses the function R after plugging in their feature values as their reward. We refer to $\Pi(R^*)$ as the *default policy*. To assess the quality of a global reward function R, we often consider the *utility feature distribution* for some feature X. This distribution shows for each value of the feature, the expected utility generated by arms with this feature value under the policy $\Pi(R)$.

3 Social Choice Language Model (SCLM)

We propose a Social Choice Language Model to generate rewards from human language composite preference prompts (see Figure 1). Separating the generation and selection of reward functions, the model consists of two sequential components. The LLM-powered *generator* generates a pool of candidate reward functions. Subsequently, taking a social-choice-inspired viewpoint, the *adjudicator* selects a reward function from the pool in two steps: i) a scorer model computes an alignment score for each reward function with each prioritization clause (i.e., we judge each reward function from perspective of all relevant "objectives") and ii) a user-defined social welfare function aggregates these scores into a "winning" candidate reward function.

Generator Given a prompt, generator creates a set of candidate reward functions (as Python code) via a variant of evolutionary search [6]: We proceed in multiple steps. First, inputting the problem description, feature descriptions and the preference prompt, we ask an LLM to generate code for a reward function. We repeat this query n_p times to obtain a set $\mathcal{R}$ of n_p candidate reward functions. Afterwards, for each function $R \in \mathcal{R}$ we compute the utility feature distributions of the policy $\Pi(R)$ induced by the reward function R on the given RMAB instance (via repeatedly simulating the policy on the instance). Then, the prompt and the set of candidate reward functions together with the associated utility feature distributions are passed to an LLM, which is asked to select the reward function R' from $\mathcal{R}$ best aligned with the prompt [12, 22]. Now, we repeat the whole process, this time including the selected policy R' as a seed in the reward function generation prompts. Once we have executed the process n_r times, we add all generated $n_p \cdot n_r$ candidate reward functions R to the pool $\mathcal{R}$ (see Section A.2 for details).

Adjudicator The adjudicator selects a reward function from pool of candidate reward functions returned by generator. To handle complex tradeoffs arising within composite prompts and resulting multi-objective optimization problem, the adjudicator follows a social choice approach. Social choice is a discipline at the intersection of economics, philosophy, and mathematics and concerned with aggregating potentially contradicting preferences of a set of voters into a fair compromise alternative from a given candidate set [4, 15]. It thus provides a theoretically grounded methodology for balancing competing interests. In our problem, we interpret reward functions as candidates and preference clauses in the prompt as voters with their preferences over the candidates reflecting the reward function's alignment with the clause. This view gives rise to the following strategy: Given a prompt $P = \{p_1, p_2, \dots, p_\ell\}$, we evaluate each reward function $R \in \mathcal{R}$ from the perspective of each preference clause p_i by computing an (alignment) score $s_i(R)$. $s_i(R)$ measures the alignment of $\Pi(R)$ with preference clause p_i, i.e., how much the voter representing p_i "likes" the candidate R.

Social Welfare Functions Social welfare functions select an alternative based on input preferences of voters. The pros and cons of individual social welfare functions have been extensively researched and debated in social choice [4, 21]. We make use of cardinal social welfare functions [10] which take as input our alignment scores $(s_i(R))_{i \in [\ell], R \in \mathcal{R}}$ and output the winning reward function. We consider the two arguably most popular social welfare functions:

Utilitarian Return the reward function maximizing the sum of its scores, i.e., $\arg\max_{R \in \mathcal{R}} \sum_{i \in [\ell]} s_i(R)$.

Egalitarian Return the reward function maximizing its minimum score, i.e., $\arg\max_{R \in \mathcal{R}} \min_{i \in [\ell]} s_i(R)$.

Social welfare functions also allow for assigning a different importance to clauses: The user could submit an importance score w_i for each clause p_i, which can be easily incorporated in the social welfare function, e.g., the Utilitarian welfare function becomes $\arg\max_{R \in \mathcal{R}} \sum_{i \in [\ell]} w_i \cdot s_i(R)$. Selecting the social welfare function gives us control over the tradeoffs between objectives: By picking the egalitarian function, we ensure that one clause will not get prioritized over another. In contrast, the Utilitarian function prioritizes the summed alignment, allowing for mismatches between clauses. Further, the adjudicator makes the selection process more transparent, as the different objectives, the selection criterion, and the performance of the candidate reward functions regarding the objectives become explicit. To compute alignment scores $s_i(R)$, we present two general methods.

Simulator Scorer Model (SCLM-SIM) First, for each preference clause $p_i \in P$, we compute a reward function R_i aligned with p_i by casting it as a singular prompt to the DLM pipeline. Next, for each $R \in \mathcal{R}$, we compute $s_i(R)$ as the expected reward according to reward function R_i produced by policy $\Pi(R)$. $s_i(R)$ quantifies the quality of the policy induced by the candidate reward function R from the perspective of p_i (as captured by R_i). As the scale of the reward functions can vary significantly among preference clauses, we normalize the scores by the performance of the default policy, i.e., we compute $\frac{s_i(R)-s_i(R^*)}{s_i(R^*)}$.

LLM Scorer Model (SCLM-LLM) The Simulator Scorer Model assumes access to reward functions capturing individual preference clauses well. If no well-aligned reward functions can be obtained, the performance of SCLM-SIM can deteriorate because it can become noisy. Another disadvantage of SCLM-SIM is that the scores in SCLM-SIM are all computed via simulation, which can be computationally expensive. Motivated by this, we propose a faster and more flexible LLM-based approach, where we prompt an LLM to rate the alignment of a candidate reward function with a preference clause. In particular, for each $R \in \mathcal{R}$ and $p_i \in P$, we use a prompt that includes R, p_i, and the utility feature distributions produced by policy $\Pi(R)$. We ask the LLM to rank how well R aligns with the preference clause p_i on a scale from 1 to 5 (see Section C for prompt texts).

4 Experiments and Results

Mobile Health Domain ARMMAN [3] is a non-profit in India that operates large-scale Maternal and Child Care Mobile Health programs for underserved communities. One of their programs disseminates critical health information via weekly automated voice messages. The goal of the NGO is to maximize beneficiaries' engagement, i.e., the number of messages they listen to. A limited number of beneficiaries are called by health workers every week to boost engagement. The problem of planning which beneficiaries to call has been modeled and solved as an RMAB, where the good/bad state corresponds to a high/low weekly engagement of the beneficiary. We use anonymized data from a quality improvement study conducted in January 2022 [24]. For each beneficiary, we have access to their income, education, and age level, which we use as our three features. Beneficiaries' historic listenership values are used to estimate their transition probabilities under the passive action [14]. One problem in estimating transition probabilities under the active action is that due to the limited number of service calls made, such transitions are rare. Thus, active transition probability estimates are noisy. To alleviate this issue, we use the features and passive transition probabilities from ARMMAN data together while we synthetically generate active transitions (see Section A.3 for details). Finally, we create datasets for three different synthetic generation hyperparameters. Each dataset consists of five sampled RMAB instances with $N = 2100$ arms, a budget of $B = 210$ and a time horizon of $T = 12$.

An instance of our problem consist of two parts: A preference prompt and an RMAB instance. We initially focus on prioritization prompts. Specifically, for each feature X, we consider two different prioritization clauses "Prioritize agents with low/high value of feature X". This gives rise to 6 singular prompts consisting of one prioritization clause, two for each feature. For composite prompts, we take all combinations of two features and the two prioritization clauses for each feature (e.g. "Prioritize agents with high value of feature A and also prioritize agents with low value of feature B"). This results in $3 \cdot 4 = 12$ composite prompts. In our experiments, we run each prompt on the 15 RMAB instances from the three datasets.

4.1 Models & Baselines

We analyze four different variants of SCLM differing in the used social welfare function (Utilitarian or Egalitarian) and scorer model (Simulator or LLM), e.g., we denote as *SCLM-SIM-Egalitarian* SCLM with the Simulator Scorer Model and the Egalitarian social welfare function. In addition, we consider several LLM-focused baselines (see Section C for detailed descriptions):

- **LLM-Zeroshot**: This baseline only queries the LLM once. It asks for a reward function aligned with given preference prompt after providing context of problem and feature description
- **DLM**: This baseline implements the Decision-Language Model by Behari et al. [6].
- **DLM-PromptEngg**: This is a modified version of DLM where within the reflection prompt, we include examples for singular queries of how the LLM should reason over the different reward function choices.

Following the work of Behari et al. [6], which constitutes our most important baseline, we use Gemini Pro [2] as the LLM in our experiments.

Evaluation Metrics As our goal is to fulfill the preferences specified by the user (in contrast to the classic goal of maximizing total utility), we need to quantify the alignment of the returned reward function with the given prompt P to evaluate our models. Due to the composite, multi-objective nature of our prompts, we start by measuring the alignment of the returned reward function R with each prioritization clause $p_i \in P$ in a separate evaluation score $e_i(R)$. We compute $e_i(R)$ as the summed utility generated by the agents with highest/lowest value of feature X under the policy $\Pi(R)$ normalized by the utility generated by these agents under the default policy $\Pi(R^*)$. Reflecting the multi-objective nature of our problem, we consider two metrics for measuring the alignment of a reward R with a full composite prompt: the sum and minimum of the $\%$ change of the utility generated by the two prioritized groups under policy $\Pi(R)$ compared to the default policy.

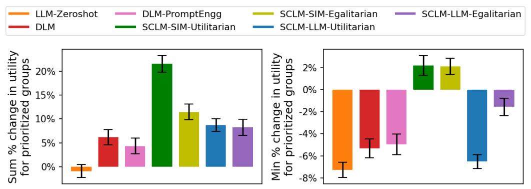

Figure 2: Sum $\%$ change (left) and minimum of $\%$ change in utility for the two groups prioritized. Results comparing the quality of reward design methods for composite prioritization prompts.

Results In Figure 2, we depict average summed and minimum alignment with the two clauses of the composite prompt, i.e., the minimum/summed change in the utility generated by the prioritized group of agents (see section B.1 for additional results). We start by focusing on SCLM with Simulator Scorer *SCLM-SIM* (green-shaded bars), our strongest method which significantly outperforms all baselines for both minimum and summed $\%$ change independent of whether Utilitarian or Egalitarian social welfare function is chosen. *SCLM-SIM-Utilitarian* outperforming the baselines for minimum change and *SCLM-SIM-Egalitarian* outperforming them for summed change highlights the advantages of SCLM, as these objectives are not explicitly optimized by the respective models, e.g., *SCLM-SIM-Utilitarian* aims at maximizing the summed change and not the minimum one, but still performs well regarding the minimum change. This indicates that SCLM independent of chosen welfare function does a better job at picking effective and aligned reward functions. Comparing *SCLM-SIM-Utilitarian* and *SCLM-SIM-Egalitarian*, the two methods exhibit a big difference under the summed change criterion, while the difference regarding minimum change is much smaller.

If we replace Simulation Scorer with LLM Scorer, the performance of the SCLM decreases, but is still better than all of our three baselines. The difference between the LLM and Simulation Scorer highlights the advantage of the additional information acquired through more complex and computationally expensive simulation method. Regarding the performance of the baselines, our DLM baseline with prompt engineering *DLM-PromptEngg* performs similarly to DLM. This suggests that prompt engineering itself is not sufficient to adequately deal with the multi-objective nature of composite prompts; an external component (like our adjudicator) is needed. Finally, LLM-zeroshot performs the worst, which highlights the non-triviality of the reward design problem and the need for extensive guided search within the reward function search space.

5 Conclusion and Social Impact

We present a customizable Social Choice Language Model to handle multi-objective nature of preference prompts in reward design for RMABs in a mobile health setting. We showcase how methods from social choice can be used to improve quality and transparency of decision-making of LLM-based frameworks, as we present adjudicator component that makes final decision from options generated by the LLM. SCLM significantly improves the quality of the chosen reward functions.

Our framework is especially useful in allowing experts at the grassroots - community volunteers and health workers to participate in decision-making by incorporating their insights into policy through natural language. This helps decentralize planning and build community tailored solutions. In addition, the framework also allows for rapid adaptation to changing government regulations. SCLM also adds a layer of transparency and reliability, which is crucial for widespread adoption. For future work, we believe SCLM can be extended to handle multiple preference prompts specified by multiple health workers or end users, which is often a challenging task because of a lack of consensus.

Acknowledgements

This work was supported by the Harvard Data Science Initiative.

References

[1] Abderrahmane Abbou and Viliam Makis. Group maintenance: A restless bandits approach. *INFORMS J. Comput.*, 31(4):719–731, 2019. doi: 10.1287/IJOC.2018.0863. URL https://doi.org/10.1287/ijoc.2018.0863.

[2] Rohan Anil et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/ARXIV.2312.11805. URL https://doi.org/10.48550/arXiv.2312.11805.

[3] ARMMAN. ARMMAN: Advancing Reduction in Mortality and Morbidity of Mothers, Children, and Neonates, 2024. https://armman.org/.

[4] Kenneth J Arrow, Amartya Sen, and Kotaro Suzumura. *Handbook of social choice and welfare*. Elsevier, 2010.

[5] Turgay Ayer, Can Zhang, Anthony Bonifonte, Anne C Spaulding, and Jagpreet Chhatwal. Prioritizing hepatitis c treatment in us prisons. *Operations Research*, 67(3):853–873, 2019.

[6] Nikhil Behari, Edwin Zhang, Yunfan Zhao, Aparna Taneja, Dheeraj Nagaraj, and Milind Tambe. A decision-language model (DLM) for dynamic restless multi-armed bandit tasks in public health. *CoRR*, abs/2402.14807, 2024. doi: 10.48550/ARXIV.2402.14807. URL https://doi.org/10.48550/arXiv.2402.14807.

[7] Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Guolong Liu, Gaoqi Liang, Junhua Zhao, and Yun Li. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *CoRR*, abs/2404.00282, 2024. doi: 10.48550/ARXIV.2404.00282. URL https://doi.org/10.48550/arXiv.2404.00282.

[8] Katrina V Deardorff, Arianna Rubin Means, Kristjana H Ásbjörnsdóttir, and Judd Walson. Strategies to improve treatment coverage in community-based public health programs: a systematic review of the literature. *PLoS neglected tropical diseases*, 12(2):e0006211, 2018.

[9] Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve: Reward evolution with large language models for autonomous driving. *CoRR*, abs/2406.01309, 2024. doi: 10.48550/ARXIV.2406.01309. URL https://doi.org/10.48550/arXiv.2406.01309.

[10] Ralph L Keeney and Craig W Kirkwood. Group decision making using cardinal social welfare functions. *Management Science*, 22(4):430–437, 1975.

[11] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=10uNUgI5Kl.

[12] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *CoRR*, abs/2310.12931, 2023. doi: 10.48550/ARXIV.2310.12931. URL https://doi.org/10.48550/arXiv.2310.12931.

[13] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. *CoRR*, abs/2406.01967, 2024. doi: 10.48550/ARXIV.2406.01967. URL https://doi.org/10.48550/arXiv.2406.01967.

[14] Aditya Mate, Lovish Madaan, Aparna Taneja, Neha Madhiwalla, Shresth Verma, Gargi Singh, Aparna Hegde, Pradeep Varakantham, and Milind Tambe. Field study in deploying restless multi-armed bandits: Assisting non-profits in improving maternal and child health. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 12017–12025, 2022.

[15] Hervé Moulin. *Fair division and collective welfare*. MIT press, 2004.

[16] Lyndsay A Nelson, Shelagh A Mulvaney, Tebeb Gebretsadik, Yun-Xian Ho, Kevin B Johnson, and Chandra Y Osborn. Disparities in the use of a mhealth medication adherence promotion intervention for low-income adults with type 2 diabetes. *Journal of the American Medical Informatics Association*, 23(1):12–18, 2016.

[17] José Niño-Mora. Markovian restless bandits and index policies: A review. *Mathematics*, 11(7): 1639, 2023.

[18] Christos H Papadimitriou and John N Tsitsiklis. The complexity of optimal queueing network control. In *Proceedings of IEEE 9th annual conference on structure in complexity Theory*, pages 318–322. IEEE, 1994.

[19] Yundi Qian, Chao Zhang, Bhaskar Krishnamachari, and Milind Tambe. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 123–131. ACM, 2016. URL http://dl.acm.org/citation.cfm?id=2936946.

[20] Yundi Qian, Chao Zhang, Bhaskar Krishnamachari, and Milind Tambe. Restless poachers: Handling exploration-exploitation tradeoffs in security domains. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 123–131, 2016.

[21] John Rawls. A theory of justice. In *Applied ethics*, pages 21–29. Routledge, 2017.

[22] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Annual Conference on Neural Information Processing Systems 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[23] Samina T Syed, Ben S Gerber, and Lisa K Sharp. Traveling towards disease: transportation barriers to health care access. *Journal of community health*, 38:976–993, 2013.

[24] Shresth Verma, Gargi Singh, Aditya Mate, Paritosh Verma, Sruthi Gorantla, Neha Madhiwalla, Aparna Hegde, Divy Thakkar, Manish Jain, Milind Tambe, and Aparna Taneja. Expanding impact of mobile health programs: SAHELI for maternal and child care. *AI Mag.*, 44(4): 363–376, 2023. doi: 10.1002/AAAI.12126. URL https://doi.org/10.1002/aaai.12126.

[25] Shresth Verma, Yunfan Zhao, Niclas Boehmer Sanket Shah, Aparna Taneja, and Milind Tambe. Group fairness in predict-then-optimize settings for restless bandits. openreview.net/pdf?id=GJlZbpLWX3, 2024.

[26] Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A):287–298, 1988.

[27] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=tUM39YTRxH.

[28] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 374–404. PMLR, 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

Appendix

A Implementation Details

A.1 Whittle Index Policy

To formulate the Whittle Index Policy, first we define the value function for an arm $i \in [N]$ under the subsidy λ as

$$V_i^{\lambda}(s) = \max_{a \in \{0,1\}} Q_i(s, a_i, \lambda). \tag{1}$$

Here, $Q_i(s, a_i, \lambda)$ measures the expected discounted cumulative future reward where a subsidy λ is added to the reward when the passive action is taken. The Whittle index associated to the state s_i is then defined as:

$$W_i(s_i) := \inf_m \left\{ Q_i(s_i, a_i = 0, m) = Q_i(s_i, a_i = 1, m) \right\}.$$

The whittle index is thus the value of subsidy such that the passive ($a = 0$) and active ($a = 1$) actions have the same value. Intuitively, this captures the value of taking active action on an arm.

To implement the Whittle Index computation, we use the method in [20] based on binary search. Additionally, for all the experiments, we cache computed whittle index for a given set of transition probabilities and reward functions to reduce computation time.

A.2 DLM Pipeline

In our work, we use a modified version of the DLM pipeline [6], which employs the Whittle Index policy as the planner for RMAB. Our approach differs from Behari et al. [6] in that we use the Whittle Index policy specifically for simulating RMAB, whereas Behari et al. [6] use the PreFeRMAB policy (Zhao and Behari et al. 2023). This modification allows for faster and more stable simulations and effectively decouples the learning problem from the planning problem.

Specifically, the DLM consists of three components. First, a user provides a natural language preference P. We then create a prompt for LLM which includes the context of the RMAB problem, the preference P, a description of features available, and the index of those features in the feature array. Finally, the LLM is asked to generate the Python code of the reward function in text format. We describe the prompt used in Figures 4 and 5.

The LLM is queried n_p times to obtain n_p reward functions. For each reward function, we also compute the reward distribution over all the features. Next, given all the generated reward functions and their corresponding reward distributions, we query the LLM to select the best reward function. We describe the prompt used in Figures 6 and 7. This is called the reflection step. The best reward function is then used inside the prompt for next step of generating n_p reward functions. This process is repeated n_r times to obtain $n_p \cdot n_r$ reward functions. In all our reward function generation experiments, we query the LLM $n_p = 4$ times and run the reflection loop $n_r = 5$ times resulting in 20 candidate reward functions.

As an LLM, we use the Gemini Pro model by Google. We query the LLM using python based API from the *generative-ai-python* library.

A.3 Synthetic Dataset Generation

An RMAB problem is defined by the transition probabilities of Markov Decision Process governing every arm and the reward functions. We consider a 2-state, 2-action Markov decision process in all our experiments. This results in 8 transition probability parameters $P_i(s, a, s') \; \forall s \in \{0,1\}, a \in \{0,1\}, s' \in \{0,1\}$ for every arm i. Out of these 8 parameters, we only need to define 4 parameters $P_i(s, a, s' = 1) \; \forall s \in \{0,1\}, a \in \{0,1\}$ independently, and the rest 4 parameters can be calculated as the compliment values $P_i(s, a, s' = 0) = 1 - P_i(s, a, s' = 1) \; \forall s \in \{0,1\}, a \in \{0,1\}$.

To simulate the effects of conflicting preferences and trade-offs in a controlled setup, we configure the four transition probability parameters to depend on the features describing each arm. We consider each arm to be characterized by a vector of continuous features f, with three values ranging between

Table 1: Weight vector parameters for different synthetic datasets.

	Feature A	**Feature B**	**Feature C**	σ
Dataset 1	0.8	-1.5	1	0.1
Dataset 2	10	-1.5	1	0.1
Dataset 3	1	-1.5	10	0.1

0 and 1 ($f \in [0,1]^3$). We use the following setup to stochastically generate the 4 transition probability parameters.

1. For every arm i, uniformly sample the two passive probability parameters $P_i(s, a = 0, s' = 1) \sim \text{Uniform}(0,1),\ \forall s \in \{0,1\}$.
2. For every arm i, uniformly sample the three features $f \sim Uniform([0,1]^3)$
3. Define a three-dimensional weight vector $W \in [0,1]^3$ and a standard deviation parameter σ.
4. Sample the effect of intervention from the normal distribution as $\delta \sim \mathcal{N}(\Delta, \sigma)$, where $\Delta = W \cdot f^T$.
5. Calculate active transition probabilities as $P(s, a = 1, s') = P(s, a = 0, s') + \delta$
6. Calculate the complimentary probabilities as $P_i(s, a, s' = 0) = 1 - P_i(s, a, s' = 1),\ \forall s \in \{0,1\}, a \in \{0,1\}$

The magnitude of the weight value determines the extent to which a feature influences the effect of the intervention, while the direction indicates whether a low or high feature value amplifies this effect. The standard deviation parameter, σ, controls the spread of sampled probabilities around the mean effect of the intervention. Table 1 shows the Weight vector used for the three synthetic datasets generated in our experiments. We consider weight values that describe the following scenarios : i) all weights are roughly equal (dataset 1) ii) one of the features has much higher weight than the other two (dataset 2) and we switch which feature has maximum weight (dataset 3).

Synthetic Domain We create three synthetic datasets, each with three features (A, B, and C). For some agent and feature, we randomly sample the agent's feature value between 0 and 1. Arm's passive transition probabilities ($P(s, a = 0, s')$ for $s, s' \in [0,1]$) are sampled uniformly between 0 and 1. Our three datasets differ in how active transition probabilities are sampled. For each dataset, we set a three-dimensional weight vector $W \in [0,1]^3$ specifying how much each feature impacts the effect of applying an active action (see Section A.3 for details). For each agent, let f denote their feature values; we sample their active transition probabilities for $s, s' \in [0,1]$ as $P(s, a = 1, s') = P(s, a = 0, s') + \delta,\ \delta \sim \mathcal{N}(\Delta, \sigma)$, where $\Delta = W \cdot f^T$ and σ is a hyperparameter. Subsequently, we discretize the values of all features into 5 different equal-sized buckets to be consistent with the real-world domain. For each dataset, we sample 5 instances according to the above procedure with $N = 2100$ arms, a budget of $B = 210$, and a time horizon of $T = 12$ (to replicate the setup of the real-world domain described below where roughly $3\% - 10\%$ of beneficiaries receive a resource every week for up to three months).

A.4 Real World Dataset

As described in Section 6.1, one challenge in estimating transition probabilities under active actions is the rarity of such transitions due to the limited number of service calls. Consequently, the estimates for active transition probabilities tend to be noisy. To mitigate this issue, we follow a procedure similar to that in the synthetic domain to construct the datasets, but with some modifications. Specifically, instead of uniformly sampling passive probabilities in step 1, we use the transition probabilities estimated from real-world data [24]. Next, in step 2, we use the attributes describing the real-world beneficiaries to define features. Thus, instead of features A, B and C, we use features Age, Income and Education. Finally, we use the same weight vector values as in Table 1 to generate three real world datasets. This allows us to generate multiple datasets with varying levels of effect of intervention based on features while still using the realistic passive transition probabilities and feature distributions.

Table 2: Results summary comparing different reward function choice strategies aggregated across the three synthetic Datasets. Cells in bold indicate the top 2 best values (higher is better). DLM: Decision-Language Model, SCLM: Social Choice Language Model, SCLM-SIM: Simulation based Scorer Model, SCLM-LLM: LLM based Scorer Model

	Social Welfare Function	**Minimum of % change in utility**			**Minimum of % change in utility**		
Method		**top/bottom 1 bucket**	**top/bottom 2 buckets**	top/bottom 3 buckets	**top/bottom 1 bucket**	**top/bottom 2 buckets**	**top/bottom 3 buckets**
LLM-zeroshot		-0.266%±1.51%	-0.228%±1.3%	-3.269%±1.14%	-15.16%±0.81%	-13.249%±0.71%	-12.485%±0.61%
DLM		3.879%±1.42%	5.379%±1.27%	1.89%±0.99%	-7.844%±0.74%	-6.249%±0.62%	-6.761%±0.49%
DLM-PromptEngg		7.607%±1.53%	8.665%±1.44%	4.087%±1.24%	-9.301%±0.82%	-7.17%±0.81%	-7.767%±0.71%
SCLM-SIM	Utilitarian	**28.944**%±**2.12**%	**21.936**%±**1.55**%	**13.654**%±**1.07**%	**-2.278**%±**1.02**%	**-3.579**%±**0.83**%	**-3.559**%±**0.55**%
SCLM-LLM		14.348%±1.66%	10.304%±1.22%	6.333%±0.99%	-3.973%±0.76%	-4.428%±0.6%	-4.817%±0.54%
SCLM-SIM	Egalitarian	16.448%±1.95%	11.425%±1.34%	8.6%±0.94%	**-1.176**%±**1.01**%	**-2.028**%±**0.71**%	**-1.833**%±**0.49**%
SCLM-LLM		11.421%±1.62%	7.845%±1.21%	4.141%±0.92%	-4.877%±0.73%	-4.373%±0.59%	-4.68%±0.5%
SCLM-SIM	Nash	**28.262**%±**2.42**%	**20.416**%±**1.73**%	**11.102**%±**1.19**%	-4.053%±1.18%	-5.408%±0.89%	-5.261%±0.62%
SCLM-LLM		9.478%±1.68%	6.608%±1.23%	2.957%±1.02%	-5.973%±0.77%	-5.782%±0.6%	-6.117%±0.54%

Table 3: Results summary comparing different reward function choice strategies aggregated across the three Real World Datasets. Cells in bold indicate the top 2 best values (higher is better). DLM: Decision-Language Model, SCLM: Social Choice Language Model, SCLM-SIM: Simulation based Scorer Model, SCLM-LLM: LLM based Scorer Model

	Social Welfare Function	**Minimum of % change in utility**			**Minimum of % change in utility**		
Method		**top/bottom 1 bucket**	**top/bottom 2 buckets**	top/bottom 3 buckets	**top/bottom 1 bucket**	**top/bottom 2 buckets**	**top/bottom 3 buckets**
LLM-zeroshot		-0.893%±1.38%	-5.541%±0.79%	-10.436%±0.46%	-7.285%±0.69%	-7.003%±0.45%	-6.971%±0.26%
DLM		6.219%±1.59%	-0.485%±0.95%	-8.733%±0.43%	-5.317%±0.84%	-6.454%±0.39%	-7.674%±0.19%
DLM-PromptEngg		4.341%±1.65%	-3.57%±0.91%	-9.417%±0.57%	-4.957%±0.94%	-7.366%±0.5%	-7.586%±0.37%
SCLM-SIM	Utilitarian	**21.502**%±**1.76**%	**10.643**%±**1.06**%	**-3.434**%±**0.53**%	**2.206**%±**0.89**%	**-1.203**%±**0.45**%	-4.423%±0.29%
SCLM-LLM		8.711%±1.3%	-0.574%±1.08%	-8.285%±0.82%	-6.512%±0.63%	-6.457%±0.55%	-6.756%±0.43%
SCLM-SIM	Egalitarian	11.481%±1.61%	7.92%±1.01%	**-0.416**%±**0.37**%	**2.109**%±**0.75**%	**0.287**%±**0.4**%	**-1.974**%±**0.22**%
SCLM-LLM		8.239%±1.67%	-1.919%±0.98%	-9.488%±0.48%	-1.561%±0.8%	-4.67%±0.39%	-7.059%±0.2%
SCLM-SIM	Nash	**20.579**%±**1.8**%	**10.195**%±**1.1**%	-4.376%±0.54%	2.09%±0.9%	-1.568%±0.47%	-4.886%±0.29%
SCLM-LLM		-0.732%±1.25%	-4.854%±0.95%	-10.455%±0.68%	-9.877%±0.65%	-8.373%±0.53%	-8.107%±0.36%

A.5 Hyperparameters

We run all experiments with $N = 2100$ arms, $B = 210$ as budget, $T = 12$ weeks and $\gamma = 0.9$ as discount factor. For every dataset, 5 RMAB instances are generated based on the different weight vectors as described in the section above. Additionally, we run each RMAB simulation with 10 different seeds. We estimate the cumulative rewards and feature-level utility by calculating the mean of the discounted sum of rewards over T timesteps across all 10 seeds.

A.6 Computing Resources

All experiments are run on a machine with 16GB RAM and M1 chip CPU. For every RMAB instance, it took roughly 3 hours to generate 360 candidate reward functions (20 reward functions for each of the 18 prompts). The primary bottleneck in speed was the API call limits for using LLM (Gemini Pro).

B Additional Results

B.1 Composite Prioritization Prompts

In Tables 2 and 3, we show results aggregated for the synthetic and real world datasets. Specifically, we show the Sum of % change in utility and Minimum of % change in utility not just for the highest/lowest value of a feature but for top/bottom two and three buckets. We also include results from the scorer model that optimizes for Nash Social Welfare function.

Overall, we observe that the Simulator-based scorer (SCLM-SIM) performs best in all scenarios, both when optimizing for the Utilitarian objective or the Egalitarian objective. It is also worth noting that when optimizing for one of the objectives (for instance, Utilitarian objective maximizes the sum of % change in utility), SCLM outperforms baselines in the objective it is not explicitly optimizing for (for instance, Minimum of % change in utility). Lastly, we observe that optimizing for the Nash objective yields very similar performance as optimizing for the Utilitarian objective.

Table 4: Results comparing different reward function choice selection strategies for deciding tradeoff between utility maximization and preference alignment, aggregated across the three synthetic datasets (left) and three real world datasets (right). Social Choice Language Model has the highest percent change in reward for the desired attribute while also ensuring the low drop in utility as compared to default.

	Synthetic			
	% Change in Desired Attribute		Utility Metrics	
Method	**top/bottom 1 bucket**	**top/bottom 2 buckets**	**Normalized Utility Score**	**% drop in Utility**
DLM-PrioritizationOnly	-0.812±0.85	0.072±0.83	0.507±0.01	-4.867±0.21
DLM-ExtendedPrompt	-2.508±1.2	-1.089±1.26	0.548±0.02	-3.825±0.29
SCLM	12.389±1.01	**10.276±0.8**	**0.71±0.01**	**-1.141±0.16**

Real World			
% Change in Desired Attribute		Utility Metrics	
top/bottom 1 bucket	**top/bottom 2 buckets**	**Normalized Utility Score**	**% drop in Utility**
5.678±0.67	1.177±0.43	0.327±0.01	-6.051±0.14
3.416±1.04	0.461±0.64	0.347±0.02	-5.689±0.25
9.058±0.61	**4.868±0.4**	**0.532±0.01**	**-3.662±0.16**

B.2 Effect of Social Choice Function

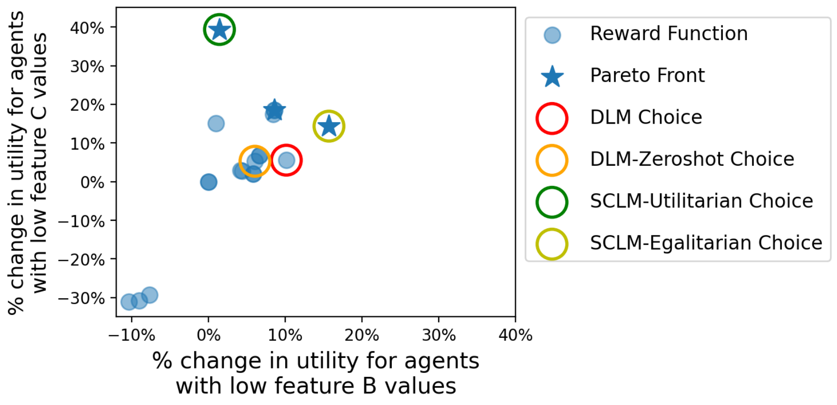

Figure 3: Comparison of different methods using the prompt "Prioritize agents with a low value for feature B and agents with a low value for feature C".

While both the Utilitarian and Egalitarian social welfare function leads to good overall results, the choice between them significantly influences which reward function is selected at the instance level. Figure 3 shows one example instance for a prompt with two prioritization clauses. Each point corresponds to a candidate reward function with the axes measuring alignment with the two respective clauses. (as described in the beginning of the section 6.3). The SCLM-Utilitarian model chooses a reward function from the Pareto frontier that shows a much stronger effect for the second clause (i.e., the utility increase for agents with a low value of feature C is more pronounced). In contrast, the reward function selected by SCLM-Egalitarian is much more balanced. DLM-based baselines fail to pick a reward function from the Pareto frontier.

C Prompt Texts

In Figures 4-13, we show the prompts passed to LLM for various experiments described in the paper. Specifically, prompt in Figures 4,5 show how to generate a reward function in code form based on the problem description, the indices of features, and preference goals for synthetic and real-world problem domains, respectively. Figures 6, 7 show how the LLM is prompted to select the best reward function from those generated in the previous step for synthetic and real-world domains, respectively. Together, these prompts establish the DLM baseline in the paper. LLM-Zeroshot baseline has the exact same prompt as Figures 4, 5 for synthetic and real world domains. The only difference is that there is no reflection step, and the first reward generated by LLM is chosen as the best reward.

In Figure 8 and 9, we show how the prompt is enhanced with additional information to assist in selecting a reward function. The additional information is highlighted in bold. Section 6.4 explains how SCLM is used to specify additional objectives, such as minimizing utility shifts in features not included in the preference prompt and maximizing overall utility. In Figures 10, 11 we show the prompts for the DLM-EP baselines, which explicitly include instructions to minimize utility shifts in unintended features. In Figures 12, 13, we show the prompts that explicitly include instructions for maximizing overall utility.

Generator: Synthetic Domain

Prompt

Create a Python reward function for RL in resource allocation problem for agents, with the objective of prioritizing higher states and: Focus on the agents with low value of feature A.. The function should use 'state' (value is either 0,1) and features 'agent_feats' (length 15 array) to direct the RL agent. Here is a description of the features you may use along with the index in 'agent_feats' array:
Index Name DataType
0. Feature A bucket 1 - Binary
1. Feature A bucket 2 - Binary
2. Feature A bucket 3 - Binary
3. Feature A bucket 4 - Binary
4. Feature A bucket 5 - Binary
5. Feature B bucket 1 - Binary
6. Feature B bucket 2 - Binary
7. Feature B bucket 3 - Binary
8. Feature B bucket 4 - Binary
9. Feature B bucket 5 - Binary
10. Feature C bucket 1 - Binary
11. Feature C bucket 2 - Binary
12. Feature C bucket 3 - Binary
13. Feature C bucket 4 - Binary
14. Feature C bucket 5 - Binary
All buckets are in increasing order of the feature values. For example, 'Feature A bucket 1' would consist of bucket of lowest values of feature A while 'Feature A bucket 5' would consist of highest value of feature A. This is true for Feature A, B and C.

Your task: 1. Write a simple, single-line Python reward function. Exclude the word 'return' and exclude non-standard libraries. Format your code with triple $ signs: $$$[YOUR FUNCTION]$$$. Note that HIGHER states are always preferred, so ensure reward increases as state increases. Make sure reward is always positive and increasing with state.

Example Prompt: Prioritize agents that have low feature A and high feature C

Example Response: Python Code: '$$$ state+state * ((agent_feats[0] or agent_feats[1]) and (agent_feats[17] or agent_feats[18] or agent_feats[19])) $$$' or '$$$ state * (agent_feats[0] or 3*agent_feats[19]) $$$' or '$$$ state + 2*state * ((5*agent_feats[0]+agent_feats[1]) and agent_feats[19]) $$$' In these example, agent_feats[0] and agent_feats[1] represent agents with low values for feature A and agent_feats[17], agent_feats[18], agent_feats[19] represent agents with high values for feature C
It is upto you to decide which features will represent a preference. For example low values could be the lowest feature bucket, or lower three feature buckets or so on. Come up with a unique new reward for the specified goal: Focus on the agents with low value of feature A..
Goal: Focus on the agents with low value of feature A.

Output

$$$ 2*state + state * (1*agent_feats[0]+ 0.5*agent_feats[1]) $$$

Figure 4: Prompt passed to the LLM to generate reward function based on the problem scenarios in the Synthetic Domain.

D Ethics Statement

ARMMAN domain

Data Usage, Collection and Consent The data used for the realworld domain contains fully-anonymyzed datasets. Our experiments constitute secondary analysis of the data and are approved by ARMMAN's ethics board. The paper does not involve any realworld deployment of the proposed algorithms for ARMMAN. For data collection consent is taken from each beneficiary in ARMMAN's automated voice call program. Data exchange and use was regulated through clearly defined exchange

Generator: Real World Domain

Prompt

Create a Python reward function for RL in phone call resource allocation to mothers in India, with the objective of prioritizing higher states and: Focus on the young mothers by age. The function should use 'state' (value is either 0,1) and features agent feats (length 43 array) to direct the RL agent. Here is a description of the features you may use along with the index in agent feats array:

Index Name DataType
0. Ages 10-20 - Binary
1. Ages 21-30 - Binary
2. Ages 31-40 - Binary
3. Ages 41-50 - Binary
4. Ages 51-60 - Binary
5. Education level 1/7 – illiterate - Binary
6. Education level 2/7 – 1-5th Grade Completed - Binary
7. Education level 3/7 – 6-9th Grade Completed - Binary
8. Education level 4/7 – 10th Grade Passed - Binary
9. Education level 5/7 – 12th Grade Passed - Binary
10. Education level 6/7 – Graduate - Binary
11. Education level 7/7 – Post graduate - Binary
12. Income bracket 1 (e.g., 0-5000) - Binary
13. Income bracket 2 (e.g., 5001-10000) - Binary
14. Income bracket 3 (e.g., 10001-15000) - Binary
15. Income bracket 4 (e.g., 15001-20000) - Binary
16. Income bracket 5 (e.g., 20001-25000) - Binary
17. Income bracket 6 (e.g., 25001-30000) - Binary
18. Income bracket 7 (e.g., 30000-999999) - Binary

Your task:
Write a simple, single-line Python reward function. Exclude the word 'return' and exclude non-standard libraries. Format your code with triple $ signs: $$$[YOUR FUNCTION]$$$. Note that HIGHER states are always preferred, so ensure reward increases as state increases. Make sure reward is always positive and increasing with state. Example Prompt: Prioritize agents that are older and rich
Example Response:
Python Code: '$$$ state * (agent_feats[4] and agent_feats[18]) $$$'
Come up with a unique new reward for the specified goal: Focus on the young mothers by age.
Goal: Focus on the young mothers by age

Output

$$$ state * agent_feats[7] $$$

Figure 5: Prompt passed to the LLM to generate reward function based on the problem scenarios in the Real World Domain.

protocols including anonymization, read-access only to researchers, restricted use of the data for research purposes only, and approval by ARMMAN's ethics review committee.

Accessibility of Health Information In our simulation experiments, we change the reward functions for every agent in mobile health program. But this only improves the quality of service calls and no health information is withheld from any agent. Participants in the program can still request service calls through free missed call service.

DLM Choice: Synthetic Domain

Prompt

My goal was to create a Python reward function for RL in resource allocation, with the objective of: Focus on the agents with low value of feature A. I tried several reward functions for this task.
Below are the reward functions I used and their corresponding reward distributions:
Function Number 0: Reward Function: 2*state + state * (1*agent_feats[0]+ 0.5*agent_feats[1])
Reflection: '
Category: A Feature A bucket 1: 113.20
Feature A bucket 2: 137.07
Feature A bucket 3: 56.51
Feature A bucket 4: 56.82
Feature A bucket 5: 54.60

Category: B Feature B bucket 1: 82.89
Feature B bucket 2: 65.36
Feature B bucket 3: 60.33
Feature B bucket 4: 50.13
Feature B bucket 5: 46.00

Category: C Feature C Bucket 1: 46.78
Feature C Bucket 2: 49.96
Feature C Bucket 3: 46.64
Feature C Bucket 4: 66.58
Feature C Bucket 5: 62.24

Function Number 1: Reward Function: state * (agent_feats[0] and not (agent_feats[1] or agent_feats[2]))
Reflection: ' Category: A Feature A bucket 1: 177.84
Feature A bucket 2: 54.76
Feature A bucket 3: 55.93
Feature A bucket 4: 57.00
Feature A bucket 5: 55.29

Category: B Feature B bucket 1: 64.47
Feature B bucket 2: 58.76
Feature B bucket 3: 56.96
Feature B bucket 4: 51.11
Feature B bucket 5: 50.13

Category: C Feature C Bucket 1: 49.80
Feature C Bucket 2: 51.69
Feature C Bucket 3: 48.40
Feature C Bucket 4: 65.89
Feature C Bucket 5: 60.00

Based on the above reward distributions and the given goal: Focus on those with high education., please identify the FUNCTION NUMBER of the most effective reward function. Provide your answer EXACTLY IN the following format: 'The best reward function is at number: [FUNCTION NUMBER]'.

Output:

The best reward function is at number: 1

Figure 6: Prompt passed to the LLM to choose a reward function based on the context of problem scenario in the Synthetic Domain, the generated reward functions and the reward distribution corresponding to every reward function.

DLM Choice: Real World Domain

Prompt

My goal was to create a Python reward function for RL in resource allocation, with the objective of: Focus on those with high education. I tried several reward functions for this task. Below, I have the given reward function, and the corresponding distribution of reward achieved across 20 agent features.
Below are the reward functions I used and their corresponding reward distributions:
Function Number 0: Reward Function: -agent_feats[5] -agent_feats[6]-agent_feats[7]-agent_feats[8]-agent_feats[9]-agent_feats[10]-agent_feats[11]
Reflection: '
Category: Age Ages 10-20: 121.73
Ages 21-30: 421.04
Ages 31-40: 244.49
Ages 41-50: 64.11
Ages 51-60: 10.58

Category: Income Income bracket 1 (e.g., 0-5000): 126.82
Income bracket 2 (e.g., 5001-10000): 373.62
Income bracket 3 (e.g., 10001-15000): 234.87
Income bracket 4 (e.g., 15001-20000): 77.40
Income bracket 5 (e.g., 20001-25000): 35.58
Income bracket 6 (e.g., 25001-30000): 2.58
Income bracket 7 (e.g., 30000-999999): 11.09

Category: Education Illiterate: 39.91
1-5th Grade Completed: 157.84
6-9th Grade Completed: 281.36
10th Grade Passed: 197.64
12th Grade Passed: 103.18
Graduate: 21.13
Post graduate: 60.89

Function Number 1: Reward Function: state * agent_feats[10]
Reflection: ' Category: Age Ages 10-20: 134.22
Ages 21-30: 469.16
Ages 31-40: 270.44
Ages 41-50: 72.80
Ages 51-60: 11.96

Category: Income Income bracket 1 (e.g., 0-5000): 138.40
Income bracket 2 (e.g., 5001-10000): 414.44
Income bracket 3 (e.g., 10001-15000): 266.44
Income bracket 4 (e.g., 15001-20000): 85.33
Income bracket 5 (e.g., 20001-25000): 40.20
Income bracket 6 (e.g., 25001-30000): 2.80
Income bracket 7 (e.g., 30000-999999): 10.96

Category: Education Illiterate: 45.07
1-5th Grade Completed: 173.82
6-9th Grade Completed: 314.07
10th Grade Passed: 217.31
12th Grade Passed: 113.02
Graduate: 29.36
Post graduate: 65.93

Based on the above reward distributions and the given goal: Focus on those with high education., please identify the FUNCTION NUMBER of the most effective reward function. Provide your answer EXACTLY IN the following format: 'The best reward function is at number: [FUNCTION NUMBER]'.

Output:

The best reward function is at number: 1

Figure 7: Prompt passed to the LLM to choose a reward function based on the context of problem scenario in Real World Domain, the generated reward functions and the reward distribution corresponding to every reward function.

DLM Choice with Prompt Engineering (DLM-PromptEngg): Real World Domain

Prompt

My goal was to create a Python reward function for RL in resource allocation, with the objective of: Focus on those with high education. I tried several reward functions for this task. Below, I have the given reward function, and the corresponding distribution of reward achieved across 20 agent features.
Below are the reward functions I used and their corresponding reward distributions:
Function Number 0: Reward Function: -agent_feats[5] -agent_feats[6]-agent_feats[7]-agent_feats[8]-agent_feats[9]-agent_feats[10]-agent_feats[11]
Reflection: '
Category: Age Ages 10-20: 121.73
Ages 21-30: 421.04
Ages 31-40: 244.49
Ages 41-50: 64.11
Ages 51-60: 10.58

Category: Income Income bracket 1 (e.g., 0-5000): 126.82
Income bracket 2 (e.g., 5001-10000): 373.62
Income bracket 3 (e.g., 10001-15000): 234.87
Income bracket 4 (e.g., 15001-20000): 77.40
Income bracket 5 (e.g., 20001-25000): 35.58
Income bracket 6 (e.g., 25001-30000): 2.58
Income bracket 7 (e.g., 30000-999999): 11.09

Category: Education Illiterate: 39.91
1-5th Grade Completed: 157.84
6-9th Grade Completed: 281.36
10th Grade Passed: 197.64
12th Grade Passed: 103.18
Graduate: 21.13
Post graduate: 60.89

Function Number 1: Reward Function: state * agent_feats[10]
Reflection: ' Category: Age Ages 10-20: 134.22
Ages 21-30: 469.16
Ages 31-40: 270.44
Ages 41-50: 72.80
Ages 51-60: 11.96

Category: Income Income bracket 1 (e.g., 0-5000): 138.40
Income bracket 2 (e.g., 5001-10000): 414.44
Income bracket 3 (e.g., 10001-15000): 266.44
Income bracket 4 (e.g., 15001-20000): 85.33
Income bracket 5 (e.g., 20001-25000): 40.20
Income bracket 6 (e.g., 25001-30000): 2.80
Income bracket 7 (e.g., 30000-999999): 10.96

Category: Education Illiterate: 45.07
1-5th Grade Completed: 173.82
6-9th Grade Completed: 314.07
10th Grade Passed: 217.31
12th Grade Passed: 113.02
Graduate: 29.36
Post graduate: 65.93

Figure 8: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional examples on what to look at when choosing a reward function aligned with the preference.

Based on the above reward distributions and the given goal: Focus on the those with low education., please identify the FUNCTION NUMBER of the most effective reward function. **You can look at the reward distributions for different features and based on them, judge the effectiveness of the correponding reward function. For instance, if the query wants to prioritize low income agents, you should look if the rewards are indeed high for low income features. it is upto you to decide which features describe low income preference.** Provide your answer EXACTLY IN the following format: 'The best reward function is at number: [FUNCTION NUMBER]'..

Output:

The best reward function is at number: 1

Figure 9: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional examples on what to look at when choosing a reward function aligned with the preference.

DLM Choice with Extended Prompt for Minimizing Utility Shifts (DLM-EP): Real World Domain

Prompt

My goal was to create a Python reward function for RL in resource allocation, with the objective of: **Focus on the young mothers by age and also focus on those with low education**. I tried several reward functions for this task. Below, I have the given reward function, and the corresponding distribution of reward achieved across 20 agent features.
Below are the reward functions I used and their corresponding reward distributions:

Function Number 0: Reward Function: state * (agent_feats[0] or agent_feats[1]) and (agent_feats[5] or agent_feats[6])
Reflection: '
Category: Age Ages 10-20: 163.24
Ages 21-30: 547.98
Ages 31-40: 269.78
Ages 41-50: 72.11
Ages 51-60: 10.91

Category: Income Income bracket 1 (e.g., 0-5000): 154.40
Income bracket 2 (e.g., 5001-10000): 472.98
Income bracket 3 (e.g., 10001-15000): 293.53
Income bracket 4 (e.g., 15001-20000): 89.82
Income bracket 5 (e.g., 20001-25000): 40.84
Income bracket 6 (e.g., 25001-30000): 2.91
Income bracket 7 (e.g., 30000-999999): 9.53

Category: Education Illiterate: 66.47
1-5th Grade Completed: 257.87
6-9th Grade Completed: 312.69
10th Grade Passed: 224.22
12th Grade Passed: 113.53
Graduate: 23.42
Post graduate: 65.82

Function Number 1: Reward Function: state * (agent_feats[0] or agent_feats[1]) * (agent_feats[5] or agent_feats[6])
Reflection: ' Category: Age Ages 10-20: 163.24
Ages 21-30: 547.98
Ages 31-40: 269.78
Ages 41-50: 72.11
Ages 51-60: 10.91

Category: Income Income bracket 1 (e.g., 0-5000): 154.40
Income bracket 2 (e.g., 5001-10000): 472.98
Income bracket 3 (e.g., 10001-15000): 293.53
Income bracket 4 (e.g., 15001-20000): 89.82
Income bracket 5 (e.g., 20001-25000): 40.84
Income bracket 6 (e.g., 25001-30000): 2.91
Income bracket 7 (e.g., 30000-999999): 9.53

Category: Education Illiterate: 66.47
1-5th Grade Completed: 257.87
6-9th Grade Completed: 312.69
10th Grade Passed: 224.22
12th Grade Passed: 113.53
Graduate: 23.42
Post graduate: 65.82
Additional Information - Rewards from Default reward function *(Reward distribution from Default reward function. Truncated for brevity.)*

Figure 10: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional information to minimize the unintended utility shifts in dimensions not specified in the preference.

Based on the above reward distributions and the given goal: **Focus on the young mothers by age and also focus on those with low education**, please identify the FUNCTION NUMBER of the most effective reward function. **Also make sure that that you choose a reward function that does not cause unintended shifts in reward. Unintended shifts in reward here means that the chosen reward function shouldn't drastically change the distribution in reward with respect to features not specified in the prompt For example, if the prompt is to prefer agents with low education, then the chosen reward function shouldn't change the distribution in reward w.r.t the default reward distribution too much in the income feature buckets** . Provide your answer EXACTLY IN the following format: 'The best reward function is at number: [FUNCTION NUMBER]'.

Output:

The best reward function is at number: 1

Figure 11: Continued: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional information to minimize the unintended utility shifts in dimensions not specified in the preference.

DLM Choice with Extended Prompt for Maximizing Overall Utility (DLM-EP): Real World Domain

Prompt

My goal was to create a Python reward function for RL in resource allocation, with the objective of: **Focus on the young mothers by age and also focus on those with low education**. I tried several reward functions for this task. Below, I have the given reward function, and the corresponding distribution of reward achieved across 20 agent features.
Below are the reward functions I used and their corresponding reward distributions:

Function Number 0: Reward Function: state * (agent_feats[0] or agent_feats[1]) and (agent_feats[5] or agent_feats[6])
Reflection: '
Category: Age Ages 10-20: 163.24
Ages 21-30: 547.98
Ages 31-40: 269.78
Ages 41-50: 72.11
Ages 51-60: 10.91

Category: Income Income bracket 1 (e.g., 0-5000): 154.40
Income bracket 2 (e.g., 5001-10000): 472.98
Income bracket 3 (e.g., 10001-15000): 293.53
Income bracket 4 (e.g., 15001-20000): 89.82
Income bracket 5 (e.g., 20001-25000): 40.84
Income bracket 6 (e.g., 25001-30000): 2.91
Income bracket 7 (e.g., 30000-999999): 9.53

Category: Education Illiterate: 66.47
1-5th Grade Completed: 257.87
6-9th Grade Completed: 312.69
10th Grade Passed: 224.22
12th Grade Passed: 113.53
Graduate: 23.42
Post graduate: 65.82

Function Number 1: Reward Function: state * (agent_feats[0] or agent_feats[1]) * (agent_feats[5] or agent_feats[6])
Reflection: ' Category: Age Ages 10-20: 163.24
Ages 21-30: 547.98
Ages 31-40: 269.78
Ages 41-50: 72.11
Ages 51-60: 10.91

Category: Income Income bracket 1 (e.g., 0-5000): 154.40
Income bracket 2 (e.g., 5001-10000): 472.98
Income bracket 3 (e.g., 10001-15000): 293.53
Income bracket 4 (e.g., 15001-20000): 89.82
Income bracket 5 (e.g., 20001-25000): 40.84
Income bracket 6 (e.g., 25001-30000): 2.91
Income bracket 7 (e.g., 30000-999999): 9.53

Category: Education Illiterate: 66.47
1-5th Grade Completed: 257.87
6-9th Grade Completed: 312.69
10th Grade Passed: 224.22
12th Grade Passed: 113.53
Graduate: 23.42
Post graduate: 65.82

Figure 12: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional information to maximize the overall utility.

Based on the above reward distributions and the given goal: **Focus on the young mothers by age and also focus on those with low education**, please identify the FUNCTION NUMBER of the most effective reward function. **Also make sure that that you choose a reward function which also maximizes the total reward. You can calculate this by adding up rewards in each feature bucket.**. Provide your answer EXACTLY IN the following format: 'The best reward function is at number: [FUNCTION NUMBER]'.

Output:

The best reward function is at number: 1

Figure 13: Continued: Enhanced Prompt passed to the LLM to choose a reward function based on the context of the problem scenario in the Real World Domain, the generated reward functions, the reward distributions corresponding to every reward function and additional information to maximize the overall utility.