PURSUING THE PARETO FRONTIER FOR INTERACTIVE INTERVENTION WITH LLM-BASED REWARD DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward design is challenging when agents must balance multiple deployment objectives. In interactive intervention, a useful reward should sustain task performance across a broad range of expert-assistance budgets, yet existing LLM-based methods rank rewards with a scalar objective, reducing multi-objective reward design to single-point evaluation. We introduce PARSE, a Pareto-aware framework that evaluates each reward by the preference-response set induced by a single preference-conditioned policy. For every candidate reward, one policy is trained across preferences , screened for Interaction Rate span and monotonicity, and evaluated by the hypervolume contribution of its nondominated operating points. Trajectory diagnoses of failed and inefficient episodes then guide reward mutation and crossover, while global exploration maintains search diversity. On PointNav in unseen Habitat scenes , PARSE attains the highest hypervolume among methods with multi-point trade-off sets and reaches 82.8% Success at a 20% Interaction Rate, outperforming the strongest baseline by 4.9 percentage points . The selected reward also transfers unchanged to ObjectNav, where it outperforms no-help and random-help baselines.

1 INTRODUCTION

Reinforcement learning (RL) has proven effective for complex tasks, including navigation, manipulation, and related domains. Nevertheless, policies often fail when deployed in novel environments or under distribution shifts. One way to make such deployment safer is asking for help. It allows the agent to request expert assistance in some states while acting autonomously in others (Chernova & Veloso, 2009). The objective is to achieve task completion while minimizing expert interaction. Reducing expert interactions too aggressively risks failure, while calling the expert too often defeats the purpose of automation. Different deployment settings imply different preferences for expert interaction, so we need a balance between performance and interaction under different preferences.

Existing methods do not directly provide this balance. One line of work triggers assistance on agent confidence or uncertainty, but these are indirect proxies for the need for interaction (Chernova & Veloso, 2009; Xie et al., 2022; Igbinedion & Karaman, 2024; Singi et al., 2024). A policy may be overconfident in an incorrect action or overly conservative about a correct one. Another line learns an interaction policy for when to request expert assistance. However, its behavior is shaped by a manually designed reward (Igbinedion & Karaman, 2024). That reward must balance requesting assistance to prevent failure, acting autonomously to conserve expert resources, and maintaining path efficiency. The central bottleneck is how to design rewards that can meet the demands.

Reward design is inherently multi-objective (Roijers et al., 2013). An effective interaction policy should improve task completion and path efficiency while reducing the interaction rate. These objectives define a Pareto frontier between performance and interaction. Different deployment settings assign different costs to expert intervention. Retraining a policy for every preference is undesirable. Ideally, a reward function trains a preference-conditioned interaction policy that performs well across preferences (Abels et al., 2019; Reymond et al., 2022).

Large language models (LLMs) make it possible to automate this search. Given a task description and access to training statistics, an LLM can generate and iteratively refine reward code, as demonstrated by Eureka (Ma et al., 2024a). Existing LLM reward design methods commonly optimize a scalar task objective (Yu et al., 2023; Xie et al., 2024; Ma et al., 2024a;b). This evaluation is not

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

directly suited to settings in which agents request expert assistance. The same reward function can induce a set of policy behaviors under different preference values. What matters is whether these responses span distinct levels of interaction, vary monotonically with preference, and contribute useful regions to the current Pareto frontier.

We introduce **P**areto-**A**ware **R**eward Design with **S**et-Valued **E**valuation (PARSE), which reframes LLM-based reward design for interactive intervention as set-valued optimization. The search object is the preference-response set induced by a single reward, rather than a single operating point. A candidate reward is valuable when its preference-response set covers controllable, nondominated trade-offs between task fitness and expert assistance.

Our contributions are as follows .

1. We formulate LLM-based reward design as optimization over the preference-response set induced by one preference-conditioned policy, rather than a scalar or single operating point.
2. We screen candidates for controllability, rank them by marginal hypervolume contribution, and use trajectory diagnosis to guide mutation, crossover, and global exploration.
3. On unseen HM3D scenes , PARSE attains the highest hypervolume among methods with trade-off sets and exceeds the strongest baseline by 4.9 percentage points in Success at a 20% Interaction Rate. The PointNav reward transfers unchanged to ObjectNav and outperforms no-help and random-help baselines.

2 Related Work

Interactive intervention. The problem of introducing expert intervention has roots in interactive imitation learning. Early work studied confidence-based querying and iterative expert supervision to reduce distribution shift (Chernova & Veloso, 2009; Ross et al., 2011). Later methods introduced human gated control and selective intervention based on novelty or task risk (Kelly et al., 2019; Hoque et al., 2022). Recent work focuses more directly on deciding when assistance is needed. AIM learns an adaptive intervention criterion from the alignment between agent actions and expert actions (Cai et al., 2025). ASK estimates policy uncertainty through dropout and queries a language model when uncertainty is high (Monteiro et al., 2026). UniIntervene detects unproductive exploration from value dynamics and performs autonomous recovery (Deng et al., 2026). Other approaches estimate intervention needs from implicit uncertainty or return variance (Igbinedion & Karaman, 2024; Singi et al., 2024). PARSE likewise learns when to intervene, but differs in using preferences over expert assistance as part of reward design to shape the intervention policy.

Reward design with LLMs. LLMs can generate executable reward code for reinforcement learning. Language to Rewards and Text2Reward generate reward functions or dense reward code from environment descriptions (Yu et al., 2023; Xie et al., 2024). Eureka uses iterative feedback to refine reward, and DrEureka extends this idea to simulator adaptation (Ma et al., 2024a;b). Recent work develops more structured procedures for reward refinement. R* decomposes reward design into reward structure evolution and parameter alignment optimization (Li et al., 2025). CARD evaluates reward code through preferences over policy trajectories and uses the resulting feedback for refinement (Sun et al., 2025). RDA visually analyzes policy trajectories, identifies failure modes, and revises reward code accordingly (Lee et al., 2026). Diagnostic Driven Refinement uses training diagnostics and structured failure categories to guide reward revision (Wang et al., 2026). PARSE considers a different evaluation object. We evaluate each reward by the preference-response set induced by a preference-conditioned policy. Reward refinement therefore considers multiple operating points rather than a single policy outcome.

Multi-objective reinforcement learning. Multi-objective reinforcement learning represents conflicting objectives with vector-valued returns and characterizes their trade-offs through Pareto optimality (Roijers et al., 2013). Conditional policies and value functions allow a single model to represent multiple operating points. Dynamic-weight methods condition value estimates on the relative importance of objectives (Abels et al., 2019). Pareto Conditioned Networks condition policies on desired Pareto-efficient returns (Reymond et al., 2022). Recent work further improves controllability and Pareto coverage. PCRL trains a preference-conditioned meta-policy across regions of the Pareto frontier (Yang et al., 2025). PSL-MORL uses a hypernetwork to generate policy parameters

for different decomposition weights (Liu et al., 2025). D^{3}PO addresses gradient interference and representational collapse through decomposed optimization and diversity regularization (Ambadkar et al., 2026). Deterministic Pareto-Optimal Policy Synthesis introduces a preference-conditioned Bellman operator based on Chebyshev scalarization and establishes convergence to a Pareto coverage set (Joshi et al., 2026). These methods focus on policy optimization under predefined multi-objective returns, whereas PARSE addresses the upstream problem of reward design.

3 PRELIMINARIES

3.1 TASK DESCRIPTION

At each decision step, the agent uses two policies. The first is a pretrained autonomous policy with frozen parameters, π_{aut}, which receives observation o_t and proposes an autonomous action $a_t^{\text{aut}} \sim \pi_{\text{aut}}(\cdot \mid o_t)$. The state encoding z_t is extracted from an intermediate layer of this autonomous policy. The second is a trainable interaction policy, π_{int}, which uses z_t to determine whether to execute the autonomous action or request expert assistance. To control the trade-off between autonomy and expert assistance, we introduce a preference parameter $\lambda \in [0, 1]$, where a larger value indicates a stronger preference for autonomous execution. The interaction policy uses z_t and λ to produce the binary decision below.

$$h_t \sim \pi_{\text{int}}(\cdot \mid z_t, \lambda), \qquad h_t \in \{0, 1\}. \tag{1}$$

Here, $h_t = 0$ means that the agent executes a_t^{aut}, whereas $h_t = 1$ means that it requests expert assistance . We call $h_t = 1$ an attempted intervention. An attempt is effective only when an expert action is available and executed. An unavailable attempt is still counted in the Interaction Rate. The preference parameter remains fixed within an episode. When the requested expert action is available, the executed action is

$$a_t = \begin{cases} a_t^{\text{aut}}, & h_t = 0, \\ a_t^{E}, & h_t = 1. \end{cases} \tag{2}$$

The autonomous policy remains fixed during training, and only the interaction policy is optimized.

3.2 PROBLEM FORMULATION

We formulate interactive reward design as a multi-objective optimization problem. The problem is evaluated by an objective vector $\mathbf{m} = (\mathbf{F}, I)$, where $\mathbf{F} \in \mathbb{R}^D$ denotes task fitness and I denotes the interaction rate, which is defined as the proportion of decision steps at which the interaction policy requests expert assistance, regardless of whether an expert action is available. For an evaluation set of episodes $\mathcal{E}$ with lengths T_n,

$$I = \frac{\sum_{n \in \mathcal{E}} \sum_{t=0}^{T_n - 1} h_{n,t}}{\sum_{n \in \mathcal{E}} T_n}. \tag{3}$$

We seek to maximize $\mathbf{F}$ while minimizing I.

A single evaluation point cannot characterize this balance. For each candidate reward R_i, we train a preference-conditioned interaction policy π_{int}^i and evaluate it on a discrete preference set $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$, where $\lambda_1 < \lambda_2 < \cdots < \lambda_K$. For each R_i and λ, we record the objective vector $\mathbf{m}_{i,\lambda} = (\mathbf{F}_{i,\lambda}, I_{i,\lambda})$. The preference-response set is

$$\mathcal{C}_i = \{\mathbf{m}_{i,\lambda} \mid \lambda \in \Lambda\}. \tag{4}$$

Operating points from different rewards and preferences jointly form a multi-objective solution space, whose nondominated points comprise the current Pareto frontier. Our objective is to find a reward whose trained interaction policy produces a broad range of interaction rates as λ varies. These operating points should contribute nondominated points to the global Pareto frontier.

4 METHODS

PARSE treats LLM reward design for interactive intervention as a set-valued optimization problem. Each candidate reward is characterized by the preference-response set induced by its preference-conditioned interaction policy, which captures the trade-off between task fitness and expert assistance across deployment preferences. PARSE combines Pareto-aware evaluation with trajectory

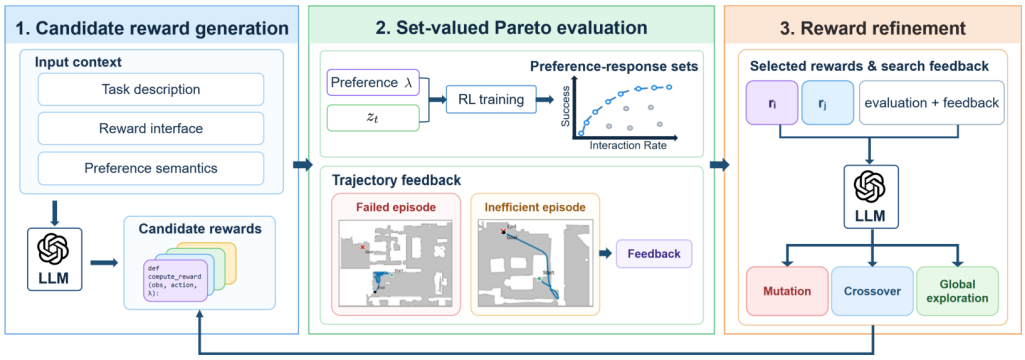


Figure 1: Overview of PARSE, the proposed LLM framework for reward design with expert intervention. The framework generates candidate rewards, evaluates their preference-response sets, collects trajectory feedback, and refines selected rewards.

feedback to guide reward refinement. These signals support mutation, reward crossover, and global exploration. Figure 1 summarizes this design, and the following subsections define its components.

4.1 Candidate Reward Generation

Each candidate reward R_i is executable Python reward code. Let C_i denote the generation context available for candidate i. Reward generation is written as

$$R_i = \mathrm{LLM}(C_i). \tag{5}$$

In the first iteration, the context includes the task objective, the semantics and available fields of the reward interface, the original navigation reward, and the meaning of the preference parameter λ. In later search iterations, the context additionally contains reward code from previous iterations, policy training statistics, Pareto evaluations, and trajectory feedback. The reward interface defines the state information available to generated code. A candidate may use only variables explicitly exposed by this interface. Some exposed fields, such as distance to the goal, distance progress, and terminal success, are privileged training signals used only to compute rewards. They are not inputs to the interaction policy and are not required at inference time. The interface therefore constrains generated code to declared training signals rather than arbitrary simulator internals. The preference parameter λ is provided explicitly to both the generated reward and the interaction policy. We require generated rewards to use λ so that the learned policy can alter its interaction behavior according to the preference. Every generated reward is checked before policy training. We verify that the code is executable, that it accesses only declared variables, and that it actually uses λ. Invalid candidates are excluded from downstream evaluation. Full prompt templates and generation configurations are provided in Appendices A.3 and C.

4.2 Set-valued Pareto evaluation

The central difference between PARSE and methods based on scalar objectives lies in how a candidate is evaluated. Each candidate R_i is used to train a single preference-conditioned interaction policy π^i_{int}. During training, the preference parameter λ is sampled uniformly from a discrete set $\Lambda = \{\lambda_1, \ldots, \lambda_K\}, \lambda_1 < \cdots < \lambda_K$. After training, we evaluate the trained policy at every $\lambda \in \Lambda$ to obtain its preference-response set $\mathcal{C}_i$. We assess both whether this set provides a controllable response as preference changes, and whether it improves the current Pareto frontier.

Quantitative Evaluation. The Interaction Rate span Span_i measures the range induced by different preferences .

$$\mathrm{Span}_i = \max_{\lambda \in \Lambda} I_{i,\lambda} - \min_{\lambda \in \Lambda} I_{i,\lambda}. \tag{6}$$

We define the preference monotonicity violation count V_i as the number of cases in which the interaction rate increases as λ increases .

$$V_i = \sum_k \mathbf{1}\left[I_{i,\lambda_{k+1}} > I_{i,\lambda_k}\right]. \tag{7}$$

During parent selection, a candidate is considered only when $\min_{\lambda\in\Lambda} I_{i,\lambda} < 0.05$, $\mathrm{Span}_i > \tau_{\mathrm{span}}$, and $V_i = 0$. These conditions remove candidates that respond weakly to preference, respond non-monotonically, or cannot approach autonomous execution.

Among candidates that satisfy these conditions, we rank preference-response sets by their contribution to the global Pareto frontier. For two objective vectors $\mathbf{x}$ and $\mathbf{y}$, the dominance relation is

$$\mathbf{x} \succ \mathbf{y} \iff F_{x,d} \geq F_{y,d} \text{ for all } d \in \{1,\ldots,D\}, \quad I_y - I_x \geq 0, \tag{8}$$

where at least one inequality is strict. For hypervolume computation, all objectives are converted to maximization form

$$\widetilde{\mathcal{C}}_i = \{(\mathbf{F}_{i,\lambda}, 1 - I_{i,\lambda}) \mid \lambda \in \Lambda\}. \tag{9}$$

Let $\mathcal{U} = \bigcup_j \widetilde{\mathcal{C}}_j$ and $\mathcal{P} = \mathrm{ND}(\mathcal{U})$ be the global Pareto frontier. The hypervolume (HV) indicator measures the Lebesgue measure of the region dominated by the Pareto frontier relative to a reference point (Zitzler et al., 2003). We use the origin in $\mathbb{R}^{D+1}$ as the reference point.

$$\mathrm{HV}(\mathcal{P}) = \mu\left(\bigcup_{x\in\mathcal{P}} \prod_{d=1}^{D} [0, F_{x,d}] \times [0, 1 - I_x]\right). \tag{10}$$

Here, μ denotes the $(D+1)$-dimensional Lebesgue measure. For results aggregated over multiple independent training runs, we first average Success, SPL, and Interaction Rate across runs separately at each preference value λ. We then construct the mean operating points from these averaged metrics and compute HV once from their nondominated set. The reported HV is therefore not an average of seed-wise HV values. For each reward R_i, its frontier hypervolume contribution is

$$\Delta\mathrm{HV}_i = \mathrm{HV}(\mathcal{P}) - \mathrm{HV}(\mathcal{P}_{-i}), \qquad \mathcal{P}_{-i} = \mathrm{ND}\left(\bigcup_{j\neq i} \widetilde{\mathcal{C}}_j\right), \tag{11}$$

where $\mathcal{P}_{-i}$ is the Pareto frontier recomputed after removing all operating points of R_i. This contribution is the core criterion for reward ranking and parent selection. The quantitative evaluation determines candidate eligibility, ranking, and parent selection. The selected parents are then analyzed through their trajectories to guide reward refinement.

Qualitative Trajectory Feedback. Aggregate metrics can identify agent performance but provide limited information about why the agent fails or completes tasks inefficiently in certain cases. Therefore, we employ trajectory feedback as a qualitative diagnostic signal for reward refinement. For this analysis, the selected parent rewards are rerun on the feedback split to collect trajectories. Candidate ranking and trajectory analysis use different data splits. We focus on failed episodes and successful but inefficient episodes. The former reveal behaviors that prevent task completion, while the latter reveal behaviors that complete the task but use inefficient trajectories.

To obtain diverse evidence within a limited LLM context window, episodes are grouped by preference λ. We traverse each group cyclically and prioritize scenes not yet represented in the feedback set. When the number of sampled trajectories is smaller than the number of preference values, we rotate the starting preference value across analysis iterations to distribute coverage over time. Each selected trajectory is compressed before being provided to the trajectory reviewer. At most 12 key frames are preserved, including the initial frame, the final frame, and extrema of continuous fields. A local window of radius two is retained around each key frame, and overlapping windows are merged. This representation preserves local behavioral context while reducing prompt length. The trajectory reviewer summarizes recurrent behavioral patterns and proposes guidance for reward modification. Typical patterns observed include prolonged stagnation, premature termination near the goal, and repeated expert requests that do not recover task progress. The resulting textual feedback is associated with the corresponding parent reward and passed to the next iteration. The review prompt is provided in Appendix C.

4.3 Reward Refinement

After parent evaluation and trajectory analysis, reward refinement constructs the next population with three complementary operators. Mutation modifies selected parent rewards, reward crossover combines complementary structures, and global exploration introduces a candidate outside the selected parent lineages.

Mutation. Parent selection is based on frontier hypervolume contribution. We retain at most three candidate rewards with the largest $\Delta\mathrm{HV}$ values from the previous iteration. Candidate quotas are allocated in proportion to the contribution of each selected parent. For each selected parent, the generation prompt includes its reward code, trajectory feedback, evaluation results, and training statistics. The LLM then generates new candidate rewards by mutating the selected parent rewards.

Crossover. Reward crossover combines candidate rewards that perform well in complementary regions of the Pareto frontier. For example, one candidate may perform well at low interaction rate, whereas another may perform well at high interaction rate. Each iteration maintains a crossover pool containing selected parents and additional candidates with strong performance in a particular interaction interval. The additional candidates are retained even when they do not satisfy the span criterion for parent selection. When a complementary pair is available, the LLM receives the two reward codes, trajectory feedback, and evaluation results, and performs crossover to generate a new candidate reward that combines their structures. The crossover branch generates at most $\lfloor N/4 \rfloor$ candidates per iteration.

Global exploration. To maintain candidate diversity, one independently generated candidate is included in every iteration. Its generation prompt is identical to the prompt used for initial candidate generation. This candidate explores reward designs outside the lineages of selected parents.

The resulting population is evaluated by the same procedure in the next iteration. The search continues for the fixed number of iterations . After completing the final search iteration, we compute the HV of each final-round candidate's own preference-response set on the MP3D ranking split. We select the candidate with the largest such HV and use its reward program as the final reward. This final-reward criterion uses candidate-set HV rather than the frontier contribution $\Delta\mathrm{HV}$ used for parent selection during search.

5 Experiments

5.1 Setup

We evaluate PARSE on PointNav in the Habitat simulator (Savva et al., 2019). The base navigation policy is pretrained and kept frozen, and the expert is a shortest-path planner. Reward design uses Matterport3D (MP3D) scenes (Chang et al., 2017). We use separate training , ranking , and feedback splits for policy training, candidate selection, and trajectory feedback, respectively. After selecting the final candidate, we evaluate it on unseen Habitat-Matterport3D (HM3D) scenes (Ramakrishnan et al., 2021). Detailed dataset splits, evaluation episodes, random seeds, and statistical reporting are provided in Appendix A.1.

Each interaction policy is trained for 250k steps. During training, each episode samples its preference parameter uniformly from $\Lambda = \{0, 0.1, \ldots, 1.0\}$. We set the interaction rate span threshold for parent selection to $\tau_{\mathrm{span}} = 0.6$. The search runs for five iterations with $N = 9$ candidate rewards per iteration.

For PointNav, the task-fitness vector $\mathbf{F}$ contains Success (S), the successful completion rate, and SPL (P), Success weighted by Path Length (Anderson et al., 2018). Since different methods yield different interaction rates, we compare curves showing performance as a function of interaction rate and examine navigation performance at comparable interaction rates. Hypervolume is computed in the objective space of Success, SPL, and $1 - \mathrm{Interaction\ Rate}$ using the definition in Section 4.2. Additional implementation details are provided in Appendix A.

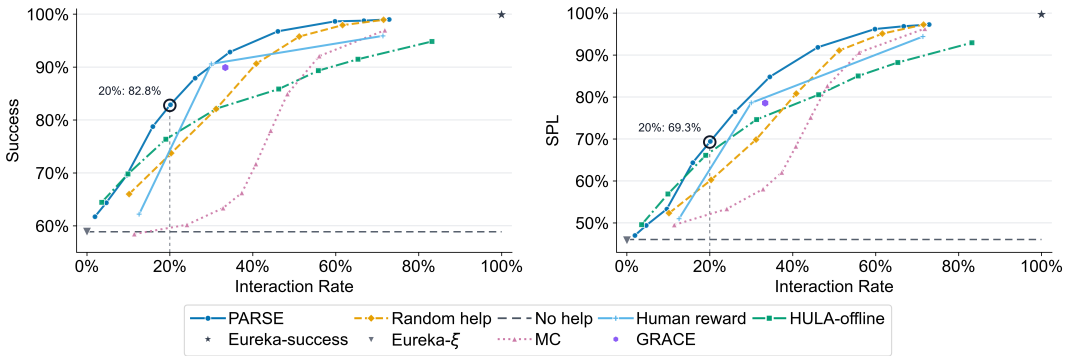


Figure 2: Success and SPL versus interaction rate on unseen HM3D scenes. The left panel reports Success and the right panel reports SPL. Points closer to the upper left indicate a better trade-off between navigation performance and expert interaction.

Table 1: Hypervolume comparison across methods on unseen HM3D scenes. Each method is evaluated over its operating points in the normalized objective space of Success, SPL, and $1 - \text{Interaction Rate}$ with the origin as the reference point. N/A denotes a method that produces only one operating point and therefore does not define a trade-off set for HV evaluation.

Method	PARSE	Random	HULA	Human	MC	No help	Eureka-S	Eureka-ξ	GRACE
HV	**0.7581**	0.6715	0.6407	0.6089	0.5850	N/A	N/A	N/A	N/A

5.2 Baselines

We compare PARSE with the following baselines. (1) **No help** always executes the frozen navigation policy without expert assistance. (2) **Random help** requests expert assistance with a fixed probability to produce different interaction rates. (3) **Human reward** trains the interaction policy with the manually designed reward from Igbinedion & Karaman (2024). (4) **Model Confusion (MC)** requests assistance when the action distribution of the frozen navigation policy is insufficiently decisive (Chi et al., 2020). (5) **HULA-offline** estimates action-return uncertainty from offline trajectories and requests assistance when the uncertainty exceeds a threshold (Singi et al., 2024). (6) **Eureka** uses LLM-generated rewards optimized with scalar objectives, with separate variants based on Success and assistance efficiency (Ma et al., 2024a). (7) **GRACE** uses offline preference pairs and evolutionary inverse reinforcement learning to generate rewards (Sapora et al., 2026). Detailed implementation and adaptation settings are provided in Appendix A.4.

5.3 Main Results

We compare PARSE with baseline methods on unseen HM3D scenes. Figure 2 shows Success and SPL across different interaction rates in its two panels, while Table 1 summarizes the overall trade-off for methods that produce multiple operating points using the hypervolume HV defined in Eq. 10. PARSE achieves a hypervolume of 0.7581 and provides higher Success and SPL at comparable Interaction Rates across most of the range. The human reward is competitive at a few interaction rates, but most of its operating points are also dominated by our frontier. For this matched-rate comparison, we fit a standard cubic spline separately to each method's measured Success–Interaction Rate operating points and evaluate every spline at $I = 0.20$. We apply this procedure only when $I = 0.20$ lies within the method's measured Interaction Rate range and do not extrapolate. At only a 20% Interaction Rate, cubic interpolation gives PARSE a Success of 82.8%, which is 23.9 percentage points above no expert interaction and 4.9 percentage points above the top-performing baseline. We additionally evaluate the final policy at preference values not used during training. The policy preserves monotonic Interaction Rate control and yields smooth Success and SPL trade-offs, with detailed results provided in Appendix E. No help, Eureka-success, Eureka-ξ, and GRACE each yield a single operating point rather than a controllable trade-off set, so HV is not applicable under our evaluation protocol. Their pointwise Success and SPL results are included in Figure 2.

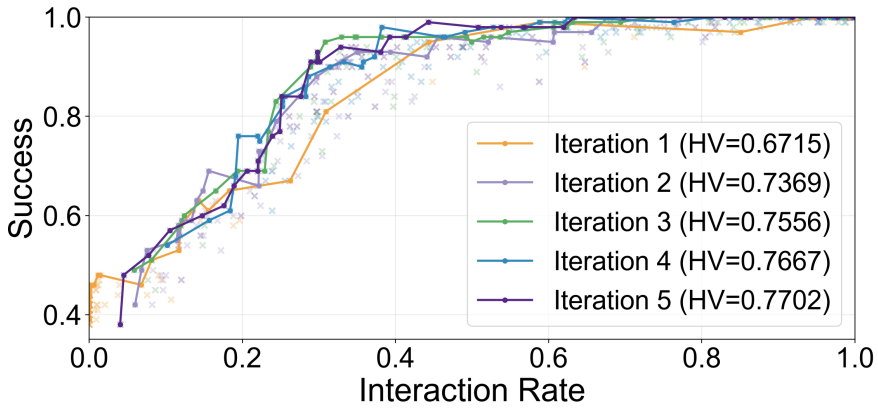


Figure 3: Pareto frontier evolution on the MP3D ranking split. Frontiers are projected onto Success and Interaction Rate, and the legend reports HV for each iteration.

Scalar objectives induce extreme interaction. Eureka-success converges to almost full expert control, with an Interaction Rate of 99.98%. Eureka-ξ shows the opposite behavior and converges to almost full autonomy, with an Interaction Rate of only 0.01%. These extremes suggest that optimizing a single scalar objective encourages degenerate interaction behaviors. GRACE learns an intermediate operating point by modeling when expert behavior should replace the frozen navigation policy, but its solution is still dominated by PARSE in both Success and SPL.

Progressive frontier improvement. To examine search progress, we analyze the candidate populations generated over five iterations on the MP3D ranking split. For each iteration, we collect the operating points from all candidate rewards generated in that iteration and compute a separate Pareto frontier. Figure 3 shows these five frontiers, while the legend reports the HV at each iteration. The HV increases from 0.6715 in the first iteration to 0.7369 in the second, and then to 0.7556, 0.7667, and 0.7702. The main increase occurs in the second iteration, with an improvement of 0.0654, while later iterations continue to improve the frontier. This trend shows that the search progressively improves the Pareto frontier, supporting the effectiveness of iterative reward refinement.

Reward Structure and Design Insights. We compare the final PARSE reward and the human reward on the same 100 HM3D validation episodes at operating points matched by interaction rate. Pearson correlations are low and range from 0.123 to 0.287, indicating that the two rewards assign different values to the same navigation transitions. The searched reward combines the native navigation reward with preference-dependent expert intervention value, autonomous stall cost, and terminal failure cost. Its intervention value depends jointly on preference, navigation uncertainty, progress stall , and recent intervention history. The correlation analysis, reward equations, and cohort details are provided in Appendices B.1 and F. The complete implementation is provided in Appendix D.

5.4 Ablation Studies

We study the contributions of set-valued Pareto evaluation, trajectory feedback, and reward crossover. All variants use the same number of search iterations and candidate rewards and the same evaluation protocol. Detailed Success and SPL trade-off curves are provided in Appendix B.2.

Set-valued evaluation ablation. We retain the full PARSE framework and replace set-valued Pareto evaluation with a scalar task objective. We use either Success or assistance efficiency ξ. The Success variant matches Eureka-success and converges to almost full expert control with an interaction rate of 99.98%. The ξ variant matches Eureka-ξ and converges to almost full autonomy with an interaction rate of 0.01%. This comparison shows that set-valued Pareto evaluation prevents the search from converging to the extreme solutions, and instead preserves operating points that balance task performance and assistance across different preferences.

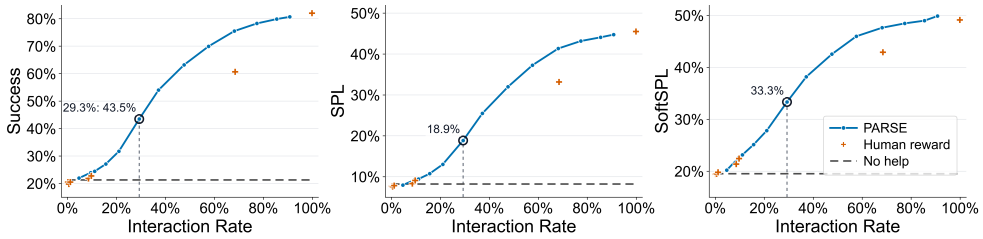


Figure 4: ObjectNav trade-offs on MP3D. The panels report Success , SPL , and SoftSPL against Interaction Rate. The PARSE curve is compared with Human reward operating points and the No help baseline. Circles mark the PARSE operating point at a 29.3% Interaction Rate.

Trajectory feedback ablation. We remove trajectory feedback from candidate evaluation. This produces a weaker frontier in both Success and SPL , with HV decreasing from 0.7581 to 0.6782. Aggregate metrics indicate whether a candidate performs well, but provide limited evidence about the behavior behind failures or inefficient paths. Trajectory feedback provides this information and gives the LLM more specific guidance for reward refinement.

Crossover ablation. We remove crossover from reward refinement. This variant also produces a weaker frontier than the full PARSE framework. The HV decreases from 0.7581 to 0.7058. The gap is smaller than that caused by removing trajectory feedback. This result suggests that crossover helps combine useful reward structures discovered by different parent candidates and improves the quality of the resulting frontier.

5.5 Reward Transfer to ObjectNav

We transfer the final PARSE reward from PointNav to ObjectNav on MP3D without modification. The frozen autonomous policy is the EmbodiedCLIP RGB ObjectNav pretrained model (Khandelwal et al., 2022) , and we train only a new interaction policy . Figure 4 reports the complete trade-off curves. No help obtains 21.28% Success, 8.21% SPL, and 19.51% SoftSPL. At a measured 29.3% Interaction Rate, PARSE reaches 43.5% Success, 18.9% SPL, and 33.3% SoftSPL, improving the three metrics by 22.2, 10.7, and 13.8 percentage points over No help. Random help , configured for the same target rate, realizes a 31.79% Interaction Rate and obtains 28.97% Success, 11.50% SPL, and 27.04% SoftSPL . PARSE therefore performs better than Random help while using a lower realized Interaction Rate .

6 Discussion

PARSE formulates LLM-based reward design as set-valued optimization over preference-conditioned interaction policies. Our results show that a single reward supports distinct intervention regimes while maintaining task performance, allowing assistance budgets to change without retraining. This finding suggests that reward design should optimize the range of behaviors induced across deployment preferences rather than one operating point.

The final reward offers a mechanism-level explanation. Expert-intervention value rises with stalled progress and navigation uncertainty, while stronger autonomy preferences and recent requests suppress repeated intervention. Penalties for autonomous stall and near-goal failure discourage waiting and premature termination. Unchanged transfer to ObjectNav suggests that these principles extend beyond PointNav.

Our evaluation remains limited to embodied navigation with a frozen autonomous policy and planner-based expert assistance. Future work should study broader tasks, richer intervention forms, learned or imperfect experts, and more efficient reward search. Overall, PARSE shows that Pareto-aware reward search can produce a controllable set of deployment behaviors instead of a separate policy for each assistance budget.

AI USE STATEMENT

Generative AI tools were used in this work for manuscript drafting and editing, literature retrieval and discovery, and research ideation and execution. In the proposed PARSE framework, GPT-5.5 was used to generate and refine executable reward functions , and trajectory feedback analysis. Generative AI tools were also used to assist with identifying relevant literature , drafting portions of the manuscript, and improving the clarity and readability of the writing. All AI-assisted content, references, code, and experimental outputs were reviewed by the authors. The authors take full responsibility for the final content, claims, code, and results presented in this paper.

REFERENCES

Axel Abels, Diederik M. Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 11–20. PMLR, 2019.

Tanmay Ambadkar, Sourav Panda, Shreyash Kale, Jonathan Dodge, and Abhinav Verma. Preference conditioned multi-objective reinforcement learning: Decomposed, diversity-driven policy optimization, 2026. URL https://arxiv.org/abs/2602.07764.

Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents, 2018. URL https://arxiv.org/abs/1807.06757.

Haoyuan Cai, Zhenghao Peng, and Bolei Zhou. Robot-gated interactive imitation learning with adaptive intervention mechanism. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 6243–6256. PMLR, 2025.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *2017 International Conference on 3D Vision*, pp. 667–676, 2017. doi: 10.1109/3DV.2017.00081.

Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009. doi: 10.1613/jair.2584.

Ta-Chung Chi, Minmin Shen, Mihail Eric, Seokhwan Kim, and Dilek Hakkani-Tür. Just ask: An interactive learning framework for vision and language navigation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 2459–2466, 2020. doi: 10.1609/aaai.v34i03.5627.

Haoyuan Deng, Yitong Gao, Yudong Lin, Haichao Liu, Zhenyu Wu, and Ziwei Wang. Uniintervene: Agentic intervention for efficient real-world reinforcement learning, 2026. URL https://arxiv.org/abs/2606.12372.

Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S. Brown, and Ken Goldberg. ThriftyDAgger: Budget-aware novelty and risk gating for interactive imitation learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164, pp. 598–608. PMLR, 2022.

Ifueko Igbinedion and Sertac Karaman. Learning when to ask for help: Efficient interactive navigation via implicit uncertainty estimation. In *2024 IEEE International Conference on Robotics and Automation*, pp. 9593–9599, 2024. doi: 10.1109/ICRA57147.2024.10610965.

Aniruddha Joshi, Niklas Lauffer, and Sanjit Seshia. Deterministic pareto-optimal policy synthesis for multi-objective reinforcement learning, 2026. URL https://arxiv.org/abs/2606.26397.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation*, pp. 8077–8083, 2019. doi: 10.1109/ICRA.2019.8793698.

Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: CLIP embeddings for embodied AI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14829–14838, June 2022. doi: 10.1109/CVPR52688.2022.01441.

Hojoon Lee, Ajay Subramanian, Ben Abbatematteo, Vijay Veerabadran, Pedro Matias, Karl Ridgeway, and Nitin Kamra. Rda: Reward design agent for reinforcement learning, 2026. URL https://arxiv.org/abs/2606.01672.

Pengyi Li, Jianye Hao, Hongyao Tang, Yifu Yuan, Jinbin Qiao, Zibin Dong, and Yan Zheng. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 34509–34527. PMLR, 2025.

Erlong Liu, Yu-Chang Wu, Xiaobin Huang, Chengrui Gao, Ren-Jian Wang, Ke Xue, and Chao Qian. Pareto set learning for multi-objective reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18789–18797, 2025. doi: 10.1609/aaai.v39i18.34068. URL https://ojs.aaai.org/index.php/AAAI/article/view/34068.

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. In *The Twelfth International Conference on Learning Representations*, 2024a.

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. In *Robotics: Science and Systems XX*, 2024b. doi: 10.15607/RSS.2024.XX.094.

Juarez Monteiro, Nathan Gavenski, Gianlucca Zuin, and Adriano Veloso. When to ASK: Uncertainty-gated language assistance for reinforcement learning. In *IEEE World Congress on Computational Intelligence*, Maastricht, Netherlands, 2026. URL https://kclpure.kcl.ac.uk/portal/en/publications/when-to-ask-uncertainty-gated-language-assistance-for-reinforceme/. 2026 International Joint Conference on Neural Networks (IJCNN), Epub ahead of print.

Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3D): 1000 large-scale 3d environments for embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/34173cb38f07f89ddbebc2ac9128303f-Abstract-round2.html.

Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowé. Pareto conditioned networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 1110–1118, 2022.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013. doi: 10.1613/JAIR.3987.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 627–635. PMLR, 2011.

Silvia Sapora, R. Devon Hjelm, Omar Attia, Alexander Toshev, and Bogdan Mazoure. GRACE: A language model framework for explainable inverse reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *2019 IEEE/CVF International Conference on Computer Vision*, pp. 9338–9346, 2019. doi: 10.1109/ICCV.2019.00943.

Siddharth Singi, Zhanpeng He, Alvin Pan, Sandip Patel, Gunnar A. Sigurdsson, Robinson Piramuthu, Shuran Song, and Matei T. Ciocarlie. Decision making for human-in-the-loop robotic agents via uncertainty-aware reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation*, pp. 7939–7945, 2024. doi: 10.1109/ICRA57147.2024.10611425.

Shengjie Sun, Runze Liu, Jiafei Lyu, Jing-Wen Yang, Liangpeng Zhang, and Xiu Li. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *Knowledge-Based Systems*, 326:114065, 2025. doi: 10.1016/j.knosys.2025.114065.

Youting Wang, Yuan Tang, Bowen Liu, Xuan Liu, and Dingyan Shang. When llm reward design fails: Diagnostic-driven refinement for sparse structured rl, 2026. URL `https://arxiv.org/abs/2605.28918`.

Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. When to ask for help: Proactive interventions in autonomous reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-1231.

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*, 2024.

Yucheng Yang, Tianyi Zhou, Mykola Pechenizkiy, and Meng Fang. Preference controllable reinforcement learning with advanced multi-objective optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 71612–71647. PMLR, 2025.

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 *Proceedings of The 7th Conference on Robot Learning*, volume 229, pp. 374–404. PMLR, 2023.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003. doi: 10.1109/TEVC.2003.810758.

A Implementation Details

A.1 Dataset Splits and Evaluation Protocol

Reward design uses three MP3D subsets for distinct purposes. The training split is used to train candidate interaction policies, the ranking split is used for candidate evaluation and selection, and the feedback split is used to collect trajectories for qualitative feedback. These three subsets are constructed from the standard MP3D train, test, and validation split files and contain 61, 18, and 4 scenes and 4,999,813, 100, and 100 episodes, respectively. Their scene-level overlap is none among the three cohorts. Training uses MP3D train, ranking uses MP3D test, and feedback uses an MP3D validation cohort. Evaluation episodes are not shared across all methods and operating points. The ranking cohort is fixed across candidates, feedback episodes are resampled for each preference value, and final evaluation uses the full HM3D validation set. Evaluation episodes are not shared across all methods and operating points, using random seed 0.

After selecting the final reward, we evaluate it on unseen HM3D scenes. The final evaluation uses 100 scenes and 2500 episodes from the official HM3D validation (val) split. HM3D data are not used during reward generation, candidate ranking, parent selection, or trajectory feedback; they are used only for final evaluation.

Unless otherwise specified, all LLM calls use GPT-5.5. For the Eureka baselines and PARSE, we run five independent PPO training runs and report the average values of the evaluation metrics. For PARSE, means and standard deviations are computed separately at each preference value λ. Table 2 reports the resulting Success, SPL, and Interaction Rate values across the five seeds. For comparison, Eureka-success obtains Success of 0.9992 ± 0.0000, SPL of 0.9969 ± 0.00002, and Interaction Rate of 0.99986 ± 0.00006, while Eureka-ξ obtains Success of 0.5888 ± 0.0000, SPL of 0.4583 ± 0.00001, and Interaction Rate of 0.0001 ± 0.0000 across the same five seeds.

Table 2: PARSE results across five independent PPO runs on the HM3D validation set. Values are reported as mean $\pm$ standard deviation at each preference value.

Success	SPL	Interaction Rate
0.9901 ± 0.0078	0.9726 ± 0.0251	0.7289 ± 0.1050
0.9877 ± 0.0110	0.9686 ± 0.0323	0.6677 ± 0.0863
0.9864 ± 0.0135	0.9619 ± 0.0383	0.5982 ± 0.0929
0.9676 ± 0.0233	0.9185 ± 0.0499	0.4605 ± 0.0648
0.9284 ± 0.0234	0.8480 ± 0.0430	0.3448 ± 0.0827
0.8792 ± 0.0247	0.7648 ± 0.0251	0.2608 ± 0.0468
0.8288 ± 0.0106	0.6942 ± 0.0193	0.2013 ± 0.0121
0.7876 ± 0.0396	0.6435 ± 0.0470	0.1589 ± 0.0153
0.6958 ± 0.0350	0.5328 ± 0.0381	0.0963 ± 0.0221
0.6436 ± 0.0226	0.4939 ± 0.0397	0.0467 ± 0.0146
0.6172 ± 0.0328	0.4700 ± 0.0202	0.0193 ± 0.0133

A.2 Interaction Policy and RL Training

The frozen navigation policy uses an RGB ResNet18 encoder followed by a one-layer GRU with hidden size 512. The trainable interaction branch uses hidden size 256 and conditions its binary intervention decision on the navigation state encoding z_t and preference parameter λ. Preference conditioning is enabled, and each episode samples λ uniformly from the eleven-point grid $\Lambda = \{0, 0.1, \ldots, 1.0\}$.

We train the interaction policy with PPO using Adam with learning rate 2.5×10^{-4} and optimizer epsilon 10^{-5}. The PPO clip parameter is 0.2, with four PPO epochs and two mini-batches per update. Rollouts contain 128 steps from four parallel environments, giving 512 transitions per rollout and 256 transitions per mini-batch. We use discount factor $\gamma = 0.99$, GAE parameter $\tau = 0.95$, value loss coefficient 0.5, entropy coefficient 0.05, and maximum gradient norm 0.5. Linear learning-rate decay and linear clip decay are enabled. Each candidate interaction policy is trained for 250k steps.

A.3 LLM Reward Search Details

Reward search uses five iterations with $N = 9$ candidate rewards per iteration. At each iteration, the selection stage retains at most three parent rewards with the largest frontier hypervolume contributions. Mutation quotas are allocated in proportion to the selected parent contributions. The crossover branch uses complementary candidates from different Pareto regions and generates at most $\lfloor N/4 \rfloor$ candidates. One additional candidate is generated by global exploration from the base prompt.

The LLM is called through an OpenAI-compatible chat-completions endpoint with model GPT-5.5 and temperature 1.0. The API client allows two retries after a failed request. Candidate generation allows up to three complete generation attempts, and a candidate that fails reward validation can receive one repair attempt using checker feedback. Candidates that remain invalid are excluded from training and downstream evaluation.

The mutation prompt supplies the selected parent reward code, measured metrics, training statistics, the measured Interaction Rate at each preference value, selected-candidate evidence, and trajectory reviewer guidance when available. The crossover prompt supplies two parent rewards together with their measured metrics, the measured Interaction Rate at each preference value, frontier complementarity evidence, and trajectory reviewer guidance. Global exploration uses the initial reward-generation prompt without parent lineage information. In the prompt templates, *Help Rate* and *attempted help rate* both refer to the Interaction Rate I defined from attempted interventions.

A.4 Baseline Details

No help. The agent always uses the frozen navigation policy and never calls the expert.

Random help. At every step, the agent requests expert assistance with a fixed probability p. Sweeping p produces performance at different interaction rates.

Human reward. This baseline uses the manually designed reward in Igbinedion & Karaman (2024).

Model Confusion (MC). We implement the Model Confusion mechanism from Just Ask (Chi et al., 2020). At every time step, the frozen navigation policy outputs a distribution over candidate actions. Let $p_{t,\mathrm{sort}[0]}$ and $p_{t,\mathrm{sort}[1]}$ be the largest action probability and the action probability that ranks second after sorting, respectively. If $p_{t,\mathrm{sort}[0]} - p_{t,\mathrm{sort}[1]} < \delta$, the agent requests the expert action a_t^E, otherwise, it executes a_t^{nav}. Sweeping δ yields performance at different interaction rates.

HULA-offline. We adapt the core idea of HULA (Singi et al., 2024) to offline navigation trajectories. For each discrete navigation action a, we learn the first and second moments of the discounted return, $Q(s,a) = \mathbb{E}[G \mid s,a]$ and $M(s,a) = \mathbb{E}[G^2 \mid s,a]$, using MLPs. The variance of the action return is $\sigma^2(s,a) = M(s,a) - Q(s,a)^2$. Sweeping the assistance threshold produces the HULA-offline curve showing the balance between performance and assistance.

Eureka. We use LLM reward generation based on Eureka (Ma et al., 2024a). It iteratively generates reward code but ranks candidates with a predefined scalar objective. We use either Success or assistance efficiency $\xi = \frac{\Delta\mathrm{Success}}{\mathrm{Interaction\ Rate}}$, where ΔSuccess is the Success improvement over No help. For each objective, Eureka runs for five iterations with nine candidate rewards per iteration. Each candidate interaction policy is trained for 250k steps.

GRACE. We adapt the evolutionary inverse reinforcement learning framework proposed by GRACE (Sapora et al., 2026). We collect trajectories with forced expert control and forced autonomy on MP3D and use an oracle to label whether expert control or autonomous control is appropriate at each state. These labels define preference pairs (s_t, a_t^+, a_t^-), where a_t^+ is the oracle action and a_t^- is the alternative. The LLM generates reward code from the offline pairs, which are screened by pairwise fitness and revised with error feedback. The retained rewards train interaction policies, and learner trajectories expand the offline data and repeat the evolutionary loop. The final reward code and interaction policy are evaluated on the HM3D validation scenes.

Compute and tuning budget. We report the following method-specific budgets so that search effort, policy optimization, and hyperparameter selection can be compared directly.

PARSE. 5 search iterations use 9 generated candidates per iteration. In total, 45 valid candidates are trained for 250k environment steps each. The search uses 45 reward-generation calls, 96 total LLM calls including repair and trajectory-review calls. Final evaluation uses 5 independent PPO runs.

Eureka-success and Eureka-ξ. Each objective uses 5 search iterations with 9 generated candidates per iteration. Each variant trains 45 valid candidates for 250k environment steps per candidate. Each variant uses 45 reward-generation calls, 81 total LLM calls including repairs. Final evaluation uses 5 independent PPO runs per variant.

GRACE. The search uses 5 evolutionary iterations with 9 generated rewards per iteration. It starts with 1125 preference pairs and adds 560, 825, 709, 764, and 729 learner-generated pairs per iteration. In total, 45 valid reward candidates are trained for 250k environment steps per interaction policy. The search uses 45 reward-generation calls, 81 total LLM calls including revisions, 5 independent final training runs.

Human reward. We use the Human reward results reported in Igbinedion & Karaman (2024). **HULA-offline.** We train on 250k offline transitions collected from 1579 completed MP3D training trajectories, perform 250k optimizer updates, use 8 calibrated assistance thresholds corresponding to target ask rates $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, and report 1 independent run. **MC.** We evaluate $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ values of δ in 1 run. **Random help.** We evaluate $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$ values of p in 1 run.

All reward-search decisions and baseline hyperparameter choices use the standard MP3D train, test, and validation splits appropriate to each stage. HM3D is not used for reward search or baseline threshold calibration. It is used only for final evaluation. Candidate ranking uses a fixed 100-episode MP3D test cohort, trajectory feedback resamples approximately 100 MP3D validation episodes per preference value, and final evaluation uses the full 2500-episode HM3D validation set.

B Additional Experimental Analysis

B.1 Reward Structure and Correlation

We compare the final PARSE reward and the human reward on the same 100 HM3D validation episodes at operating points matched by interaction rate. The cohort and trace schema are described in Appendix F. Figure 5 reports the Pearson correlations at these matched interaction rates. Pearson correlations are low and range from 0.123 to 0.287, indicating that the two rewards assign different values to the same navigation transitions.

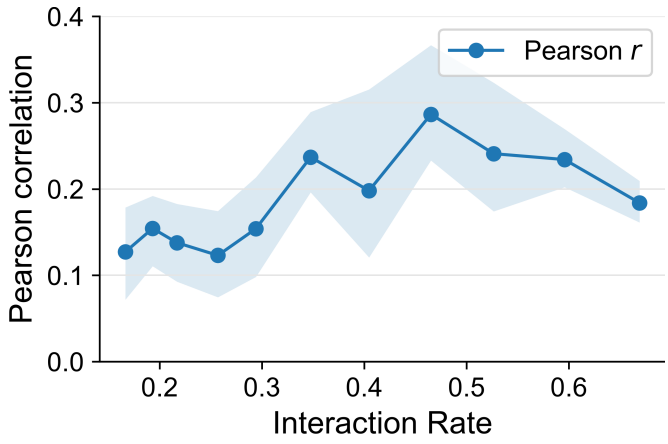


Figure 5: Pearson correlation coefficients between our searched reward and the human reward on HM3D.

To make this structure explicit, let $h_t \in \{0,1\}$ indicate an attempted intervention and let $e_t \in \{0,1\}$ indicate effective expert intervention. Thus, $e_t \le h_t$, and an unavailable expert request has $h_t = 1$ and $e_t = 0$. The reported Interaction Rate is computed from h_t, whereas the reward term for expert intervention uses e_t. Let n_t denote the nonnegative progress stall duration, Δd_t denote distance progress, H_t denote navigation entropy, C_t denote the number of attempted interventions in the previous five steps, and d_t denote distance to the goal. We use $done_t$ and $success_t$ for the terminal indicators. Nonfinite inputs are mapped to zero as in the implementation. The normalized variables are

$$\begin{aligned} \bar{\lambda} &= \mathrm{clip}(\lambda, 0, 1), & s_t &= \mathrm{clip}(n_t/6, 0, 1), & q_t &= \mathrm{clip}((n_t - 4)/12, 0, 1), \\ u_t &= \mathrm{clip}(H_t/1.5, 0, 1), & c_t &= \mathrm{clip}(C_t, 0, 5), & g_t &= \mathrm{clip}((0.55 - d_t)/0.55, 0, 1). \end{aligned} \tag{12}$$

The complete reward can then be written as

$$\begin{aligned} r_t &= r_t^{\mathrm{nav}} + r_t^E + r_t^A + r_t^F, \\ r_t^E &= e_t \Big[(1 - \bar{\lambda}) \big(1.55 + 0.75(0.60 s_t + 0.40 u_t)\big) \\ &\qquad - \bar{\lambda}(3.35 + 0.20 c_t) - 0.18 c_t + 0.45 + 3.45 q_t \Big], \\ r_t^A &= -(1 - e_t)\mathbf{1}\{\Delta d_t \le 0 \wedge n_t \ge 4\}\big(0.30 q_t + 3.35 q_t^2\big), \\ r_t^F &= -\mathbf{1}\{done_t \wedge \neg success_t\}\big(1.40 + 2.60 g_t\big). \end{aligned} \tag{13}$$

Here, r_t^{nav} is the native navigation reward. The terms r_t^E, r_t^A, and r_t^F represent expert intervention value, autonomous stall cost, and terminal failure cost. Expert intervention value rises with stall, navigation entropy, and severe stall, while preference and recent intervention count reduce this value. Autonomous stall incurs a cost after at least four steps of nonpositive distance progress, and terminal failure receives a larger cost near the goal. The complete searched reward implementation is provided in Appendix D.

B.2 Ablation Trade-off Curves

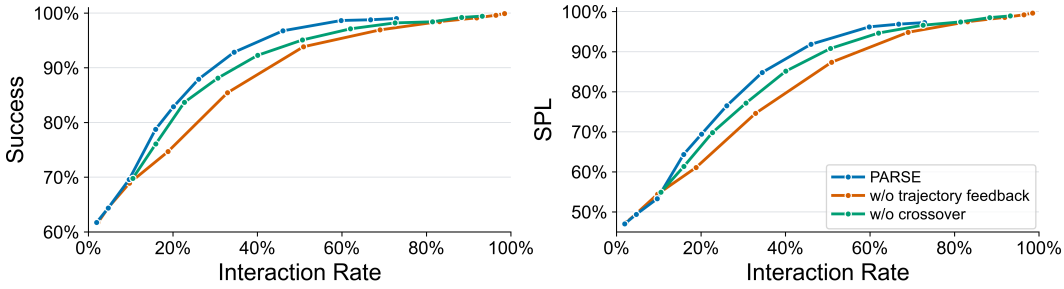


Figure 6: Impact of trajectory feedback and reward crossover on the Success and SPL trade-offs. The left panel shows Success and the right shows SPL. For each ablation, the full PARSE framework and the corresponding variant are evaluated under the same evaluation conditions.

C Prompt Templates

The implementation loads the following templates from text files and fills the braced fields at runtime. The initial system and user templates are shared by initial generation, mutation, crossover, and global exploration. Mutation and crossover append the lineage blocks shown below.

Initial reward-generation prompt.

```
--- initial_system.txt ---
You are a reward engineer designing effective reward functions for reinforcement
learning. Write the complete rollout reward for a Habitat navigation agent that
learns when to request optional oracle help while navigating successfully and
efficiently across attempted-help rates.
```

```
--- initial_user.txt ---
Use these inputs to generate the reward.

# Interface
{reward_signature}

# Available RewardInputs
{environment_interface_code}

# Previous Reward
{previous_reward_section}

# Evaluation Metrics
```json
{feedback_summary_json}
```

Preference-Response Requirements
Treat `RewardInputs.preference_lambda` as a continuous control in `[0, 1]`.
The evaluated attempted help rate must be monotone non-increasing as lambda
increases: for `lambda_a < lambda_b`, `HelpRate(lambda_a) >= HelpRate(lambda_b)`.
Design the lambda-dependent reward terms explicitly so this relation is robust
to rollout noise; target zero monotonicity violations.

At the same time, maximize useful coverage of the help-rate operating range:
generate a reward with a broad help-rate span across the lambda values while
preserving Success and SPL. Use a continuous lambda-dependent function that
yields a smooth, measurable preference-response curve.

Generate a reward whose evaluated help-rate curve on the 11-point grid
`Lambda = {{0.0, 0.1, ..., 1.0}}` has
`max(HelpRate(Lambda)) - min(HelpRate(Lambda)) > 0.6` and
`min(HelpRate(Lambda)) <= 0.05`. Measure both from evaluated rollout metrics.
Make its lambda-dependent reward influence strong enough to meet these
standards while preserving the monotonicity, Success, and SPL objectives.

For candidate-generation feedback, use the paired `lambda -> help_rate`
measurements to maintain a continuous, monotone relationship and improve the
parent-selection measurements across the full lambda range.

Training Feedback
```json
{reward_training_feedback_json}
```

{selected_candidate_evidence_section}

{trajectory_analysis_section}

Output
{candidate_output_tip}
```

Reward interface and output blocks.

```
--- reward_signature.txt ---
```python
from avrm.reward_components import RewardBreakdown, make_breakdown
from avrm.reward_inputs import bind_avrm_reward

def compute_avrm_reward(
    preference_lambda: float,
    navigation_reward: float,
) -> RewardBreakdown:
    components = {
        "navigation_reward": navigation_reward,
        "preference_anchor": 0.0 * preference_lambda,
    }
    return make_breakdown(components)

compute_reward = bind_avrm_reward(compute_avrm_reward)
```

Define `compute_avrm_reward(...)` with explicit parameters selected from
`RewardInputs`. Return `RewardBreakdown` containing the complete final rollout
reward. `navigation_reward` is Habitat's native navigation reward and must be
treated as the initial reward input. The function may retain, replace, or
transform that value. The trainer will use `total` directly, without adding the
navigation reward again and without scaling or clipping it. Both
`preference_lambda` and `navigation_reward` are required parameters and must be
```

```
used. Bind the Habitat callback exactly as shown above.

--- candidate_output_tip.txt ---
Return only Python code in one fenced block.

- Compare reward terms by meaning, not only by exact names, formulas, or
  conditions. If multiple terms express the same reward purpose or behavior
  signal, merge them into one component even when their implementations differ;
  keep only genuinely distinct terms and do not reward or penalize the same
  meaning repeatedly.
- Limit code to 200 non-empty, non-comment-only lines.
- Use imports from `avrm.reward_components`, `avrm.reward_inputs`, `__future__`,
  `math`, or `typing`.
- Keep execution pure and local: numeric computation with no external effects.
- Write only the requested candidate file.
```

Mutation prompt.

```
The shared initial system and user templates are followed by this lineage block.

# Candidate Lineage: Parent-Specific Branch
Parent candidate: <candidate_id>
Measured parent metrics: <metrics_json>
Use the following reward as this candidate's parent.

## Selected Parent Evidence
```json
<selected_parent_evidence_json>
```

Parent Lambda -> Help Rate
```text
lambda  help_rate
<paired_lambda_help_rate_values>
```

[Parent Help Feedback, when available]
[Trajectory Reviewer Guidance, when available]

```python
<parent_reward_code>
```
```

Crossover prompt.

```
The shared initial system and user templates are followed by this lineage block.

# Candidate Lineage: Cross-Reward Recombination
Direct parent candidate: <direct_parent_id>
Direct parent measured metrics: <direct_parent_metrics_json>
Cross-family source candidate: <cross_family_source_id>

## Direct Parent Reward
```python
<direct_parent_reward_code>
```

Cross-Family Source Reward
```python
<cross_family_source_reward_code>
```

Cross-Source Selection Reason
```json
<cross_source_selection_reason_json>
```

Frontier Complementarity
```json
<frontier_complementarity_json>
```

Direct Parent Lambda -> Help Rate
```text
lambda  help_rate
<direct_parent_curve>
```

Cross-Family Source Lambda -> Help Rate
```

```
```text
lambda  help_rate
<cross_family_source_curve>
```

Selected Elite Evidence
```json
<selected_elite_evidence_json>
```

Form and implement a recombination hypothesis grounded in the supplied reward
structures and frontier complementarity.

[Trajectory Reviewer Guidance, when available]

The generated Python file begins with these structured comments.
Recombination Hypothesis: <concise hypothesis>
Structure Sources: <structures taken from each reward>
Evidence: <frontier complementarity used>
```

Trajectory reviewer prompt.

```
# Trajectory bucket review <reviewer_index + 1>

Review scope:
```json
<allowed_scope_json>
```

Review only the supplied parent trajectory evidence and active bucket history.
Do not infer or request dataset, ranking, frontier, training, or reward-family
metadata. Analyse failed or successful-but-inefficient trajectories at the listed
preference lambdas.

preference_lambda controls the trade-off between autonomy and help: smaller
lambda permits more oracle help, while larger lambda should produce a
non-increasing Help Rate and therefore less help.

For a single-parent review, guidance applies only to parent_A. For a
cross-parent review, guidance must be shared advice for modifying the combined
reward formed from parent_A and parent_B; do not split guidance by parent.

Produce bucket create, update, merge, or noop proposals only when they are
grounded in the supplied parent trajectory evidence and active bucket history.

Return one JSON object with this exact top-level form:
{"schema_version": <schema_version>, "proposals": [...], "guidance": [...]}

`guidance` is a list of at most <MAX_GUIDANCE_ITEMS> short, actionable
reward-edit directions. It is independent of bucket mutations, must be grounded
in the supplied evidence, and may be empty when no additional direction is
warranted. Each item must be at most <MAX_GUIDANCE_ITEM_CHARS> characters.

Each proposal uses one action:
- create: action, pattern, support, uncertainty, evidence_refs
- update: the same fields plus bucket_id
- merge: the same fields plus target_bucket_id and merged_bucket_ids
- noop: action, reason, evidence_refs; set reason to
  `evidence_insufficient` or `already_represented`.

`support` may be a concise non-empty text summary, a non-empty list of
supporting samples or facts, or a positive numeric support count. The precise
grounding still belongs in `evidence_refs`.

Evidence refs are objects shaped as {"sample_id": "sample_001", "steps": [1, 7]}.
For update and merge, include the current bucket ids. Ground every proposal in
the supplied summary and retained windows.

Every create, update, and merge proposal includes one `reward_instruction`.
Write one sentence that names the RewardInputs fields and states the reward
change to implement with them. The candidate reward receives this instruction
directly.

Trajectory interpretation contract:
```text
<trajectory_contract>
```

Active parent bucket history:
```json
```
```

````
<registry_context_json>
```

Trajectory evidence:
```json
<evidence_context_json>
```
````

When a reviewer response fails schema validation, the implementation appends the validation error and the instruction `Return a corrected JSON object.` to the prompt before retrying.

D FINAL SEARCHED REWARD

The complete final searched reward used by the reported method is reproduced below. It exposes only the declared reward inputs and returns the four reward components used by the rollout trainer.

```python
from __future__ import annotations

import math

from avrm.reward_components import RewardBreakdown, make_breakdown
from avrm.reward_inputs import bind_avrm_reward


def _finite(x: float, default: float = 0.0) -> float:
    x = float(x)
    return x if math.isfinite(x) else default


def _clip(x: float, low: float, high: float) -> float:
    return min(high, max(low, x))


def compute_avrm_reward(
    preference_lambda: float,
    navigation_reward: float,
    effective_help: bool,
    progress_stall_steps: int,
    distance_progress: float,
    nav_entropy: float,
    recent_attempted_help_count_5: int,
    done: bool,
    success: bool,
    distance_to_goal: float,
) -> RewardBreakdown:
    """Navigation utility with continuous help pricing and failure recovery."""

    lam = _clip(_finite(preference_lambda), 0.0, 1.0)
    progress = _finite(distance_progress)
    stall_steps = max(0.0, _finite(progress_stall_steps))
    stall = _clip(stall_steps / 6.0, 0.0, 1.0)
    severe_stall = _clip((stall_steps - 4.0) / 12.0, 0.0, 1.0)
    uncertainty = _clip(_finite(nav_entropy) / 1.5, 0.0, 1.0)
    recent_help = _clip(_finite(recent_attempted_help_count_5), 0.0, 5.0)

    help_value = 0.0
    if effective_help:
        urgency = 0.60 * stall + 0.40 * uncertainty
        help_value = (
            (1.0 - lam) * (1.55 + 0.75 * urgency)
            - lam * (3.35 + 0.20 * recent_help)
            - 0.18 * recent_help
            + 0.45
            + 3.45 * severe_stall
        )

    stall_cost = 0.0
    if not effective_help and progress <= 0.0 and stall_steps >= 4.0:
        stall_cost = -(0.30 * severe_stall + 3.35 * severe_stall**2)

    failure_cost = 0.0
    if done and not success:
        remaining = max(0.0, _finite(distance_to_goal))
        near_goal = _clip((0.55 - remaining) / 0.55, 0.0, 1.0)
        failure_cost = -(1.40 + 2.60 * near_goal)

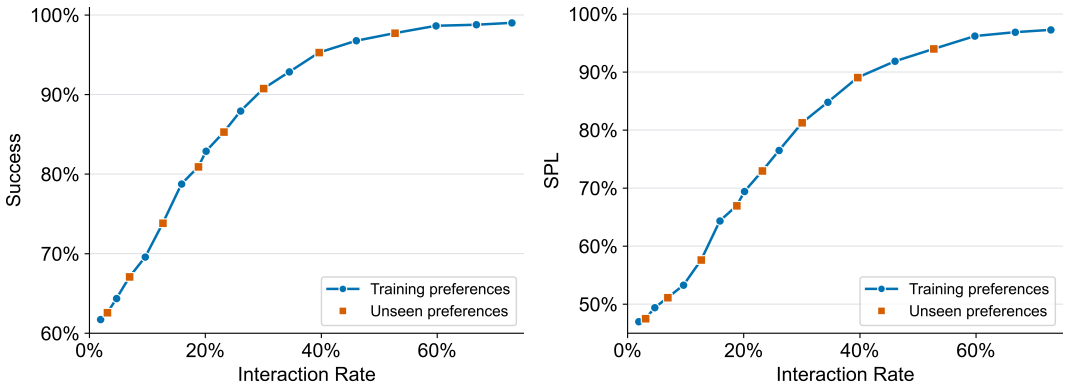
    return make_breakdown({
```
```

```
        "navigation_reward": _finite(navigation_reward),
        "help_decision_value": _finite(help_value),
        "autonomous_stall_cost": _finite(stall_cost),
        "terminal_failure_cost": _finite(failure_cost),
    })


compute_reward = bind_avrm_reward(compute_avrm_reward)
```

E Interpolation Evaluation

We evaluate the final policy at preference values not used during training. This interpolation protocol tests whether control extends beyond the discrete training grid. The policy is trained using $0, 0.1, \ldots, 1.0$ and evaluated additionally at $0.05, 0.15, \ldots, 0.75$. No retraining or reward modification is performed. Figure 7 shows the responses at training and unseen preferences.



(a) Success versus Interaction Rate.

(b) SPL versus Interaction Rate.

Figure 7: Interpolation responses of the final policy on HM3D. Circles denote the training preference grid and squares denote additional preference evaluations. The two panels report Success and SPL against Interaction Rate.

F Reward Correspondence Details

The cohort contains 100 randomly selected HM3D validation episodes. It is shared by all eleven candidate preference values. All 1,100 evaluation episodes reach a terminal transition. The replay requires the logged navigation reward, terminal flag, Success, effective expert intervention, unavailable expert intervention, and the state fields used by the candidate reward. Both effective and unavailable intervention attempts contribute to the reported Interaction Rate, while only effective interventions activate the expert-intervention reward term. The cohort list, hashes, trace schema, per-transition scores, and component values are retained as source data.