ELEMENTAL: Interactive Learning from Demonstrations and Vision-Language Models for Reward Design in Robotics

Letian Chen [1] **Nina Moorman** [1] **Matthew Gombolay** [1]

Abstract

Reinforcement learning (RL) has demonstrated compelling performance in robotic tasks, but its success often hinges on the design of complex, ad hoc reward functions. Researchers have explored how Large Language Models (LLMs) could enable non-expert users to specify reward functions more easily. However, LLMs struggle to balance the importance of different features, generalize poorly to out-of-distribution robotic tasks, and cannot represent the problem properly with only text-based descriptions. To address these challenges, we propose ELEMENTAL (intEractive LEarning froM dEmoNstraTion And Language), a novel framework that combines natural language guidance with visual user demonstrations to align robot behavior with user intentions better. By incorporating visual inputs, ELEMENTAL overcomes the limitations of text-only task specifications, while leveraging inverse reinforcement learning (IRL) to balance feature weights and match the demonstrated behaviors optimally. ELEMENTAL also introduces an iterative feedback-loop through self-reflection to improve feature, reward, and policy learning. Our experiment results demonstrate that ELEMENTAL outperforms prior work by 42.3% on task success, and achieves 41.3% better generalization in out-of-distribution tasks, highlighting its robustness in LfD.

1. Introduction

Reinforcement Learning (RL) has been shown to be a powerful tool for enabling robots to perform complex tasks across a wide range of domains, from manipulation (Kober et al., 2013; Levine et al., 2016) to navigation (Tai et al., 2017; Zhu et al., 2017). However, the effectiveness of RL hinges on the availability of a carefully designed reward function that accurately encapsulates the desired behavior. Without sophisticated reward functions, RL agents often struggle to learn competent policies (Matignon et al., 2006); worse yet, poorly designed reward functions can lead RL agents to undesirable outcomes (Gupta et al., 2024). Booth et al. (2023) shows reward specification is non-trivial even for experts, and designing reward functions that align with end-users' expectations is particularly challenging due to the varied and latent user preferences (Abouelazm et al., 2024).

Considering recent advancements in language models (e.g., large language models (LLMs) and vision-language models (VLMs)) on text understanding (Bommasani et al., 2021; Touvron et al., 2023) and emergent abilities (Kojima et al., 2022; Wei et al., 2022), researchers have explored utilizing LLMs for reward engineering. For instance, the EUREKA framework takes as input a text description of the task, queries language models for a draft of the reward function, and trains a policy with the reward function (Ma et al., 2023b). This paradigm, while promising, presents several limitations. First, describing complex robotic tasks purely through language is imprecise: humans often have latent, unspoken preferences that are difficult to articulate fully, leading to incomplete or ambiguous descriptions of their objectives (Nisbett & Wilson, 1977; Ericsson & Simon, 1980; Hoffman et al., 1995; Feldon, 2007). Second, even if all objective function components are accurately conveyed, determining the relative importance or weights of these components poses another significant challenge: assigning these weights involves subtle mathematical trade-offs, something that LLMs are not particularly equipped to handle. As a result of these limitations, methods like EUREKA struggle to generalize well to out-of-distribution tasks.

Given these limitations, a more natural and effective approach is for users to provide demonstrations of the desired behavior to supplement a general task description. Demonstrations offer rich, illustrative information that can capture not only the task objectives but also the nuanced, latent preferences that may be difficult to express verbally. Learning from Demonstration (LfD) approaches seek to leverage human-provided demonstrations to reverse-engineer the un-

[1]School of Interactive Computing, Georgia Institute of Technology, Atlanta, United States. Correspondence to: Letian Chen <letian.chen@gatech.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

derlying objective and optimize a policy accordingly (Chen et al., 2020; 2022; Suay et al., 2016; Ravichandar et al., 2020). However, a key challenge in LfD is the ambiguity in interpreting demonstrations – there can be an infinite number of possible reward functions that could explain the same set of demonstrations, a problem commonly referred to as the *reward ambiguity problem* in Inverse Reinforcement Learning (IRL) (Abbeel & Ng, 2004). Prior methods have sought to address this ambiguity by pre-designing features to constrain the space of possible reward functions, but this often limits the flexibility and generalizability of the learned rewards and policies (Zhu & Hu, 2018; Arora & Doshi, 2021). Our key insight is that language models are well-suited to contextualize demonstrations and infer task features, narrowing down the possible interpretations and enabling robots to learn more robustly.

We propose to integrate the strengths of language models and LfD methods, leveraging (1) the emergent reasoning capabilities of language models to identify robust and relevant objective function components, and (2) the demonstration-matching capabilities of LfD to determine the optimal weighing of these components. Crucially, we incorporate visual demonstrations into Vision-Language Models (VLMs), facilitating a more comprehensive understanding of human objectives. Additionally, we introduce a self-reflection mechanism that enables VLMs and LfD to iteratively improve both feature extraction and reward & policy learning. This fusion of LfD and VLMs offers a novel pathway for more effective robotic learning from demonstrations. This paradigm also mirrors how humans naturally learn from others. When observing a demonstration, humans typically (1) identify the key aspects of the task, (2) formulate a policy to match the demonstration based on those salient features, and (3) reflect on the discrepancies between their own behavior and the demonstration to refine their understanding and execution (Locke, 1987; Di Stefano et al., 2014). By iterating through these steps, humans progressively improve their performance. This iterative cycle of observation, reflection, and refinement is not only fundamental to human learning but also serves as an ideal framework for robotic learning (Chernova & Thomaz, 2014).

To develop this novel integration between VLM and LfD, we introduce ELEMENTAL (intEractive LEarning froM dEmoNstraTion And Language). ELEMENTAL enables robots to identify key task features from human demonstrations, learn rewards and policies that align on these features, and iteratively reflect on their performance to improve over time. Our key contributions are three-folds:

1. We propose a novel, general framework that integrates VLMs and LfD and introduces an iterative self-reflecting mechanism for autonomous performance improvement. ELEMENTAL is the first to incorporate visual demonstration inputs into language models to accomplish LfD, which facilitates more accurate behavior understanding.
2. We evaluate ELEMENTAL on a set of challenging, standard robotic benchmarks in IsaacGym, demonstrating its superior performance over previous state-of-the-art (SOTA) reward design and LfD methods by 42.3%, showcasing its effectiveness.
3. We further assess ELEMENTAL's generalization capabilities by designing novel variants of the standard benchmarks. Our results show that ELEMENTAL achieves 41.3% better generalization than existing methods, underscoring the importance of combining VLMs with LfD.

2. Related Work

Learning from Demonstration (LfD) – LfD approaches, such as Behavior Cloning (BC) and IRL, have long been used to enable robots to learn from human-provided demonstrations. BC (Ross et al., 2011), a supervised learning approach, is effective for relatively simple tasks, but is prone to compounding errors (known as *covariate shift*). IRL (Ng & Russell, 2000; Abbeel & Ng, 2004; Ziebart et al., 2008; Ziebart, 2010) seeks to infer the underlying reward function that explains the demonstrated behavior. However, reward ambiguity poses a significant challenge – an infinite number of reward functions could explain the same behavior. This challenge becomes exacerbated in complex domains with limited, hetergeneous demonstrations (Chen et al., 2020; Peng et al., 2024a). ELEMENTAL addresses this key limitation by integrating VLMs to inject emergent reasoning capabilities into the learning process. VLMs reduce ambiguity by providing semantic context that allows robots to better understand relevant task features.

Language Models as Reward Engineers – Recent works, such as EUREKA (Ma et al., 2023b) and L2R (Yu et al., 2023), leverage LLMs to convert language descriptions into reward functions, offering a promising alternative to manual reward engineering. However, these methods are limited by their reliance solely on concise task descriptions, restricting their ability to capture the full complexity of robotic tasks and the subtle preferences of users. Additionally, determining the appropriate weighting of different objective function components is particularly challenging, as the reward design process is disconnected from policy training, resulting in poor out-of-distribution generalization. ELEMENTAL addresses these limitations by integrating IRL with VLMs and supplementing task descriptions with demonstrations. In ELEMENTAL, the responsibility of assigning reward component weights is shifted from the VLM to IRL, which matches the reward components to the demonstrated behaviors. This allows the VLM to focus on its strength—semantic understanding and task feature identification. ELEMENTAL differs from prior work that uses language as a semantic prior or task modeling

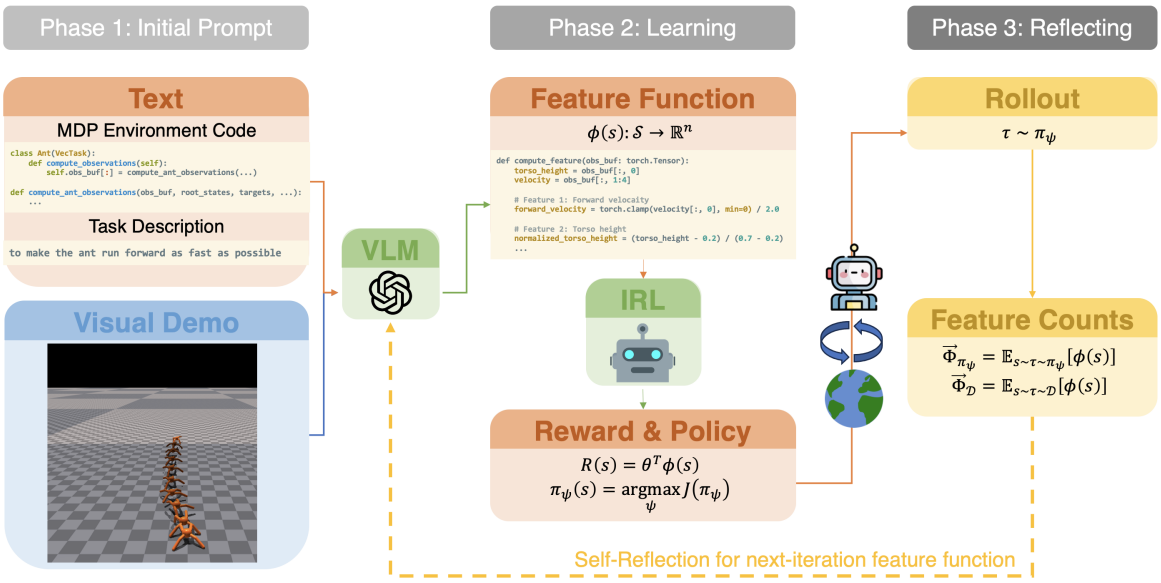


Figure 1. This figure illustrates the overall pipeline of ELEMENTAL. The process begins with an initial prompt to the VLM, which generates a draft of the feature function based on both textual descriptions and visual demonstrations. In the learning phase, ELEMENTAL infers the reward and policy from the drafted feature function and the demonstration. In the final phase, ELEMENTAL performs self-reflection by comparing the feature counts from the generated trajectory and the demonstration, again utilizing the drafted feature function. This self-reflection loop updates the feature function by feeding the results back to the VLM for iterative refinement.

aid (Lin et al., 2023; Ma et al., 2023a; Fan et al., 2022; Nair et al., 2022; Myers et al., 2024; Adeniji et al., 2023). These methods often employ language to pretrain representations (Nair et al., 2022), construct modular task plans (Myers et al., 2024), or inject commonsense priors into embodied agents (Fan et al., 2022; Lin et al., 2023). In contrast, ELEMENTAL leverages VLMs to directly generate structured, executable feature functions from multimodal input, which are then used for downstream IRL-based reward inference. This unique pipeline grounds reward learning in both language and demonstrations and supports iterative refinement through feedback.

A closely related line of research is that by Peng et al. (2024b), which also integrates feature design using LLMs with IRL. While this prior work is promising, it operates under several limiting assumptions. First, it assumes that demonstrations can be provided as input exclusively as text-based state-vector sequences. Due to this constraint, their approach is only applied to simpler tasks (e.g., problem horizons of fewer than five steps) with all but one or two features of the ground-truth features already specified. Additionally, as seen in other works (Yu et al., 2023; Kwon et al., 2023), this method relies on specially designed prompts tailored to each experimental domain. These limitations hinder its scalability to more complex, high-dimensional tasks typically encountered in real-world robotic applications. In contrast, ELEMENTAL leverages visual modality for demonstration inputs, making it well-suited for more complex tasks where textual descriptions alone are insufficient. In addition, to account for the complex domains, we introduce an upgraded MaxEnt-IRL approach, detailed in Sec. 4.2. By incorporating visual modality, using a general prompt across domains, and developing enhanced IRL, ELEMENTAL provides a more scalable solution, as demonstrated in our results on standard robotic benchmarks without requiring any prior knowledge of task features.

LM-Assisted Robot Learning – Several recent works have sought to leverage LMs to assist in robotic learning, such as RL-VLM-F (Wang et al., 2024), RoboCLIP (Sontakke et al., 2024), and (Du et al., 2023). While promising, these methods depend on a surrogate reward derived from the LM's understanding of the task-state alignment, which can introduce inaccuracies. Furthermore, these approaches lack interactivity with users, a key component shown to be helpful in gaining human trust (Chi & Malle, 2024). In contrast, ELEMENTAL potentially allows engineers and users to interactively refine the robot's behavior, ensuring that the robot is user-aligned.

Other works, such as Wang et al. (2023) and Mahadevan et al. (2024), directly query LMs to output robot actions or primitives. These approaches rely heavily on the LLM's ability to plan and optimize actions, but LLMs are not inherently designed for mathematical optimization required for robot control. ELEMENTAL addresses these limitations by using VLMs to understand task features and deferring

the demonstration-to-policy alignment to IRL algorithms, which are better suited to optimize behavior.

3. Preliminary

In this section, we introduce Markov Decision Processes, Inverse RL, and Maximum-Entropy IRL.

Markov Decision Process: We formulate the robot learning problem as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$. $\mathcal{S}$ is the set of states the agent can occupy. $\mathcal{A}$ is the set of actions the agent can take. $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ represents the transition probability function, where $T(s, a, s')$ gives the probability of transitioning from state s to state s' after taking action a. $R : \mathcal{S} \to \mathbb{R}$ is the reward function that assigns a scalar reward to each state. $\gamma \in [0, 1)$ is the temporal discount factor. The goal of Reinforcement Learning (RL) is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative reward, given by: $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$.

Inverse Reinforcement Learning: In IRL, instead of explicitly programming a robot's behavior, we aim to learn the underlying objective or reward function, $R(s)$, which explains the behavior demonstrated by a human expert. A set of demonstrations are given, $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where each trajectory $\tau_i = (s_1^i, a_1^i, s_2^i, a_2^i, \dots)$ consists of a sequence of states and actions. We assume that the true reward function, $R(s)$, is a linear combination of features: $R(s) = \theta^T \phi(s)$, where $\phi(s) \in \mathbb{R}^d$ is a feature vector representing the task-relevant or user-preference-relevant properties of the state, and $\theta \in \mathbb{R}^d$ is a weight vector that specifies the relative importance of each feature. The goal of IRL is to recover θ based on the provided demonstrations, $\mathcal{D}$. MaxEnt-IRL (Ziebart et al., 2008) models the likelihood of a trajectory under the assumption that the expert's behavior is stochastically optimal, as shown in Eq. 1. In this equation, $Z(\theta)$ is the partition function, $Z(\theta) = \sum_\tau \exp\left(\sum_{s_t\in\tau} R(s_t)\right)$.

$$P(\tau|\theta) = \frac{1}{Z(\theta)} e^{\sum_{s_t\in\tau} R(s_t)} = \frac{1}{Z(\theta)} e^{\sum_{s_t\in\tau} \theta^T \phi(s)} \quad (1)$$

MaxEnt IRL seeks to infer the reward weights θ that maximize the likelihood of the expert demonstrations (Eq. 2).

$$\hat{\theta} = \arg\max_{\theta} \sum_{\tau\in\mathcal{D}} \log P(\tau|\theta) \quad (2)$$

In ELEMENTAL, we upgrade MaxEnt-IRL for high-dimensional robotic tasks, allowing us to link features from VLMs to demonstrations via the weight vector θ.

4. Method

In this section, we present ELEMENTAL, which consists of three interconnected phases (Figure 1). The first phase involves constructing an initial feature function through a VLM based on visual human demonstrations and environment specifications (Sec. 4.1). In the second phase, this feature function is integrated with IRL to learn a reward function and policy that best align with the demonstrated behaviors (Sec. 4.2). The final phase introduces a self-reflection mechanism that automatically compares the learned behavior with the demonstrations, enabling iterative refinement of the feature function, reward function, and policy (Sec. 4.3).

4.1. Phase 1: Initial Prompt

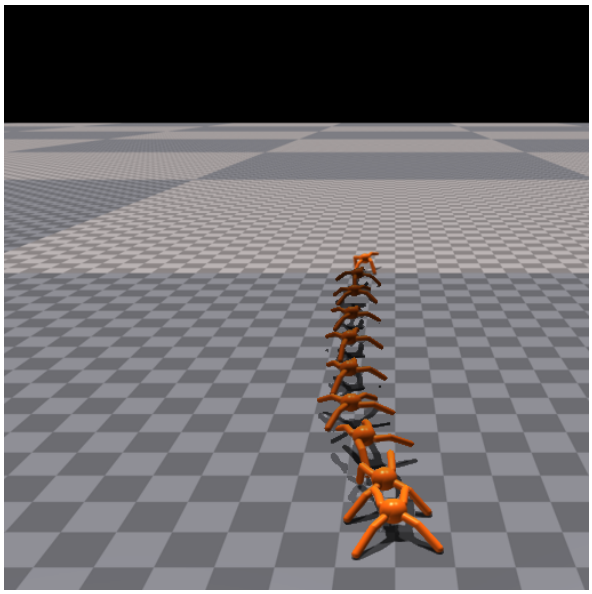
The first phase begins with an initial prompt of three inputs: (1) the MDP environment code specifying the environment's state space $\mathcal{S}$, (2) a text description of the task, and (3) a visual human demonstration of the desired task. The form of the visual demonstration depends on the task domain (Figure 2): for tasks where superimposition is meaningful (e.g., locomotion tasks such as the moving-forward ant or humanoid), we compose a superimposed image that displays the sequences of demonstrated motion (shading indicates temporal progression). For manipulation tasks, where superimposition can result in cluttered robot poses (e.g., Franka-Panda Cabinet), four keyframes picked by robotic experts are provided to the VLM to illustrate stages of the task execution. By incorporating visual demonstrations alongside language-based task descriptions, ELEMENTAL allows the VLM to generate feature functions that more accurately reflect the user's latent goals.

These inputs are then processed by the VLM, which leverages its emergent capabilities to infer task-relevant features. The goal is not just to describe the state but also to capture important factors that align with the user's intentions for the task. The output of this phase is a feature function $\phi : \mathcal{S} \to \mathbb{R}^n$, where $\phi(s) = (\phi_1(s), \phi_2(s), \dots, \phi_n(s))$ is an n-dimensional vector of features describing the key aspects of the task. Given VLM coding capabilities (Piccolo et al., 2023), we ask the VLM to output feature functions as Python code, providing a structured representation. If the returned code is not executable (e.g., wrong function signature), we re-prompt the VLM up to three times with the trackback information. A successfully executable feature function serves as the foundation for reward learning and policy optimization in the subsequent phase.

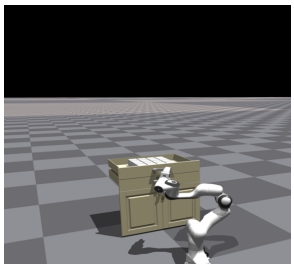
4.2. Phase 2: Learning

After the feature function has been drafted in Phase 1, we next optimize the reward function, $R_\theta(s) = \theta^T \phi(s)$, to match the demonstrations using the feature function, $\phi(s)$. As in Sec. 3, we modify the MaxEnt-IRL algorithm to accomplish reward and policy learning in more complex tasks. We name it *Approximate MaxEnt-IRL* (Algorithm 1).

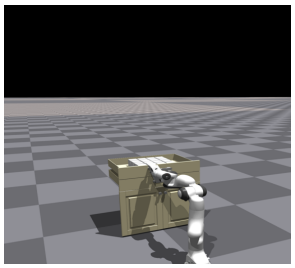
Directly computing the partition function $Z(\theta)$ is intractable



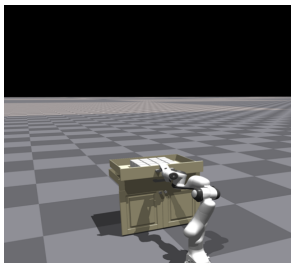
(a) Superimposing ant demonstration.



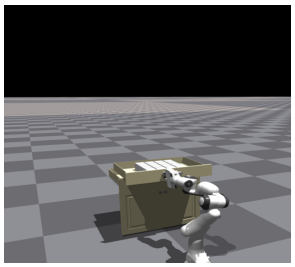
(b) start of the trajectory.



(c) reaching the cabinet handle.



(d) opening the cabinet.



(e) the cabinet is opened.

Figure 2. This figure illustrates the visual demonstrations for both locomotion and manipulation tasks. (a) shows an example from the Ant locomotion task, where superimposed images are used. For manipulation tasks, superimposed images can result in cluttered robot poses, so we use key frames as visual demonstration inputs instead. (b-e) present four key frames from the FrankaCabinet manipulation task.

Algorithm 1: Approximate MaxEnt-IRL

Input : feature function $\phi : \mathcal{S} \to \mathbb{R}^n$, demonstration $\mathcal{D}$, number of IRL iterations m, learning rate α, policy learning steps k

1 Initilize reward function feature weights $\theta = \{1/n\}_{i=1}^{n}$ and policy weights ψ.
2 **for** $i \leftarrow 1$ **to** m **do**
3 **for** $j \leftarrow 1$ **to** k **do**
4 Optimize π_ψ based on $J(\pi_\psi)$ with $R_\theta(s) = \theta^T \phi(s)$ via PPO
5 Obtain ∇_θ by Eq. 4 and normalize to get ∇'_θ by Eq. 5
6 Update θ by $\theta \leftarrow \theta + \alpha \nabla'_\theta$
7 Normalize θ by Eq. 6

Output : R_θ, π_ψ

due to the summation over all possible trajectories. We circumvent the need for explicit computation of $Z(\theta)$ with the key insight (Ziebart et al., 2008) that the gradient of the MaxEnt IRL objective is given by the feature expectation difference between the demonstrated trajectories and the stochastically optimal trajectory under θ, as shown in Eq. 3.

$$\nabla_\theta = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{s \in \tau} \phi(s) \right] - \mathbb{E}_{\tau \sim P(\tau|\theta)} \left[\sum_{s \in \tau} \phi(s) \right] \quad (3)$$

We approximate $P(\tau|\theta)$ by a parameterized policy, π_ψ, that optimizes the estimated reward, $R_\theta(s) = \theta^T \phi(s)$, in Eq. 4.

$$\nabla_\theta \approx \mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{s \in \tau} \phi(s) \right] - \mathbb{E}_{\tau \sim \pi_\psi} \left[\sum_{s \in \tau} \phi(s) \right] \quad (4)$$

In particular, we optimize the policy π_ψ interleaved with R_θ, following a paradigm similar to that of AIRL (Fu et al., 2018), as shown in Algorithm 1 lines 3-4 and line 6. Intuitively, by applying this gradient, we refine θ to ensure that the reward function $R_\theta(s)$ more accurately reflects the features emphasized in the demonstration and guide π_ψ to align the policy's behavior closer to the demonstrated behavior. The learned policy, π_ψ, is also a natural byproduct of this process, serving as the ultimate goal of LfD and laying the foundation for the reflection phase in Phase 3.

As the feature's numerical magnitude varies, we apply a normalization procedure to the reward gradient to ensure stable learning. First, we normalize the L1-norm of the gradient vector as in Eq. 5.

$$\nabla'_\theta = \frac{\nabla_\theta}{||\nabla_\theta||_1} \quad (5)$$

Next, we apply gradient ascent on the reward weight vector, θ, using a learning rate α: $\theta \leftarrow \theta + \alpha \nabla'_\theta$. After each update, we normalize θ (Eq. 6) to stabilize the training of the policy, π_ψ, as the semantics of a reward function do not change by scaling (Fu et al., 2018).

$$\theta \leftarrow \frac{\theta}{||\theta||_1} \quad (6)$$

4.3. Phase 3: Self-Reflecting on Features

The third phase introduces a self-reflection mechanism, designed to close the loop and iteratively refine the feature function drafted in Phase 1. After Phase 2, the learned policy, $\pi_\psi(s)$, is executed in the environment, and its behavior is compared to the demonstration, $\mathcal{D}$, based on the drafted feature function $\phi(s)$. Specifically, we calculate the expected feature counts of the generated trajectories under the current policy and for the demonstration trajectories (Eq. 7).

$$\vec{\Phi}_{\pi_\psi} = \mathbb{E}_{\tau\sim\pi_\psi}\left[\sum_{s\in\tau}\phi(s)\right], \vec{\Phi}_{\mathcal{D}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{s\in\tau}\phi(s)\right] \quad (7)$$

Discrepancies between the two feature counts indicate that the current feature function may not fully capture the relevant aspects of the task as demonstrated.

The two feature count vectors are then fed back to the VLM, which uses the feature count differences to revise the feature function $\phi(s)$. By accounting for previously overlooked or misinterpreted features, the VLM's understanding of the task becomes progressively more aligned with the demonstrated behavior. This process of self-reflection continues iteratively, alternating between Phase 2 (reward function and policy optimization) and Phase 3 (feature refinement), allowing the robot to improve its behavior over time.

The reflecting phase is fully automatic, leveraging the policy, $\pi_\psi(s)$, and the feature function, $\phi(s)$, generated in previous phases. As both the policy and the feature function are available, ELEMENTAL can continuously refine its understanding of the task without additional human input. However, should the user wish to intervene and provide further feedback or corrections, ELEMENTAL could accommodate interaction in future work through prompting.

5. Results

In this section, we evaluate the performance of ELEMENTAL on challenging robotic tasks from IsaacGym (Makoviychuk et al., 2021) against SOTA baselines (Sec. 5.1) and show generalization to task variants (Sec. 5.2).

Environments and Tasks – In benchmark experiments, we test ELEMENTAL and baseline algorithms on nine challenging IsaacGym Robotics tasks, using GPU-accelerated training to enable efficient experiments. These tasks span various domains, including locomotion and manipulation, and are recognized for their complexity in the robot learning community (Makoviychuk et al., 2021). For all methods that utilize a LLM/VLM, we use the OpenAI GPT-4o model unless otherwise noted. We use five demonstrations for each task collected with RL-trained policies.

To the best of our knowledge, this is the first successful application of IRL to IsaacGym – with or without large models – due to its high-dimensional state and action spaces. The performance of ELEMENTAL in these tasks demonstrates its robustness and scalability, making it a suitable framework for real-world robotics problems using IRL.

Baselines – We compare ELEMENTAL against key LfD and LLM-powered reward engineering approaches:

1. LfD methods – We include standard LfD techniques, i.e. Behavior Cloning (BC) and IRL. These baselines learn from demonstration but do not incorporate the VLM feature-extraction or visual input.

2. EUREKA – This is the previous SOTA method for reward design with RL in IsaacGym. EUREKA relies on LLMs to infer task features from textual inputs but does not utilize demonstrations, visual inputs, or IRL.

3. Random Policy – A random policy establishes an approximate lower bound for competent task performance.

4. Ground-Truth (GT) Reward – The performance of a policy trained with GT reward predefined in IsaacGym provides an upper bound of the task performance. Note that although we call it "upper bound", it is not necessarily the maximum performance one can achieve, as the RL is given the same budget of environment steps to train, and the GT reward may not be the best.

5. Ablations of ELEMENTAL – To better understand the contribution of each component in ELEMENTAL, we conduct ablation studies across six variants: 1) Without (w/o) Self-Reflection: This variant removes the self-reflection mechanism introduced in Phase 3, keeping only the VLM-guided feature inference and policy learning phases; 2) w/o Gradient Normalization: This ablation removes the gradient normalization in Eq. 5; 3) w/o Weight Normalization: This ablation removes the weight normalization step in Eq. 6; 4) w/o Visual Input: This variant removes the visual demonstration input to the VLM, leaving only language-based task descriptions for feature inference; 5) w/ Text Demo: Instead of visual demonstrations, this ablation provides textual input by listing observation vectors from a subsampled sequence of ten states from the demonstration, same as (Peng et al., 2024b); 6) w/ Random Visual Demo: To test whether the VLM effectively extracts task-relevant information, we replace the high-quality visual demonstration input with random visual demonstrations (e.g., a falling Cartpole or Humanoid). The IRL phase, however, still utilizes the original high-quality demonstrations.

Performance is evaluated using an average task success rate of the final-100 steps during training, which is a more reliable metric versus only using the *max* success during training reported by Eureka. In all experiments, we report

Table 1. This table shows benchmarking results on IsaacGym tasks. Bold denotes the best performance except the GT Reward.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
Random (LB)	25.42	87.39	-1.63	0.00	0.00	-0.04	-2.45	0.00	0.02
BC	149.85	344.55	-1.19	0.01	-0.05	-0.43	-2.14	0.04	0.03
IRL	28.15	162.06	-1.87	0.00	0.88	2.13	-2.22	0.01	0.01
EUREKA	215.91	454.18	**-0.22**	0.21	6.88	3.78	-4.24	11.12	0.00[1]
ELEMENTAL (Ours)	233.92	**464.40**	-0.30	**0.36**	**8.49**	**4.70**	**-0.83**	**22.97**	**2.71**
w/o Self-Reflection	114.66	153.52	-0.93	0.00	5.05	3.65	-1.71	0.02	0.03
w/o Gradient Normalization	186.52	423.78	-1.20	0.02	7.29	2.73	-0.95	13.39	2.32
w/o Weight Normalization	192.51	459.33	-0.54	0.02	3.23	4.87	-1.15	0.04	1.57
w/o Visual Input	178.68	304.58	-1.01	0.00	8.16	4.49	-1.41	18.52	0.03
w/ Text Demo[2]	207.51	412.17	-0.92	0.00	7.43	4.60	-0.88	7.07	0.04
w/ Random Visual Demo.	**269.46**	352.53	-1.07	0.02	7.13	3.93	-1.07	20.75	2.33
GT Reward (UB)	260.14	461.90	-0.27	0.40	7.00	5.07	-0.03	23.70	0.15

[1] We tried running with six seeds, but Eureka with GPT-4o failed to generate any executable reward function for ShadowHand. In the calculation of percentage improvements over Eureka, we treat the improvements on ShadowHand to be 100%.
[2] As the original implementation in (Peng et al., 2024b) is not available, we implement the demonstration in text form with ELEMENTAL.

the best performance across three random seeds to account for variable responses from GPT-4o. Hyperparameters and prompts are provided in supplementary.

5.1. Benchmarking Results

In the first set of experiments, we benchmark ELEMENTAL and the baselines on the IsaacGym tasks, with the results presented in Table 1. BC performs adequately on simpler tasks such as Cartpole and BallBalance, but fails on more complex tasks due to covariate shift. IRL without VLM-based feature extraction struggles to learn effectively in most tasks, highlighting the challenges posed by IsaacGym's high-dimensional state spaces. While EUREKA is able to learn capable policies, ELEMENTAL achieves on-average 42.3% higher performance and outperforms EUREKA on eight out of nine tasks, demonstrating the effectiveness of integrating IRL with VLM-derived features and visual demonstration information. The correlation between learned reward and ground-truth reward shown in Figure 2 also illustrates ELEMENTAL's strong ability to learn a well-aligned reward function by matching with demonstrations. Expanded results with statistical significance tests and additional baselines (e.g., VLM+BC and ablations with different VLMs) are shown in Supplementary C.

To further evaluate whether VLMs are more suitable for feature extraction or for drafting full reward functions, we compare the code execution rates of ELEMENTAL and EUREKA across three algorithm iterations. A higher code execution rate indicates fewer coding errors, suggesting better compatibility with language model capabilities. As shown in Figure 3, ELEMENTAL achieves a successful code execution rate of approximately 80% in the first iteration, compared to less than 50% for EUREKA. While both methods improve with successive iterations, ELEMENTAL consistently generates more executable code across all tasks. A two-way repeated measures ANOVA across nine paired tasks reveals a significant main effect of algorithm ($F(1, 8) = 7.00, p = .030$) and round ($F(2, 16) = 10.03, p = .002$), indicating that ELEMENTAL achieves statistically significantly higher execution rates than EUREKA. The interaction between algorithm and round is not significant ($F(2, 16) = 2.21, p = .144$). These results support the interpretation that GPT-4o is more effective at generating executable feature extraction code than complete reward functions, thereby validating ELEMENTAL's design choice to delegate reward weighting to IRL.

The ablation results in Table 1 demonstrate the importance of ELEMENTAL's key components. Removing self-reflection significantly reduces performance, confirming its role in refining both the feature function via VLM and the policy via IRL. Notably, even without self-reflection, ELEMENTAL outperforms standard IRL, indicating that the initial VLM-generated feature function already provides benefits. The normalization steps are also crucial—removing them leads to unstable reward learning and inconsistent scaling, negatively impacting policy performance.

Performance degrades more sharply when substituting visual demonstrations with textual inputs, as in the setup from Peng et al. (2024b), suggesting that VLMs leverage visual semantics more effectively than structured state vectors—particularly on tasks that are difficult to describe via language alone. For example, on FrankaCabinet, replacing

Table 2. This table shows reward correlation of inferred reward functions for EUREKA and ELEMENTAL (ours) on IsaacGym tasks.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
EUREKA	0.77	-0.53	0.96	0.93	1.00	0.59	-0.86	0.34	0.00
ELEMENTAL (Ours)	0.99	0.85	0.89	0.98	1.00	0.98	0.97	0.58	0.31

Table 3. This table compares the generalization performance of ELEMENTAL and EUREKA on Ant-variant environments. Results are maximized over three seeds. Bold denotes the best performance.

Method	Ant Original	w/ Reversed Obs	w/ Lighter Gravity	Ant Running Backward
EUREKA	6.88	5.96	4.39	5.62
ELEMENTAL	**8.49**	**8.47**	**5.89**	**9.30**
w/o Visual Input	8.16	7.63	3.14	7.46

visual demonstrations with text inputs reduces performance from 0.36 to 0.00, underscoring how VLMs extract more actionable features from visual inputs than from tokenized state vectors (Peng et al., 2024b).

Additionally, using random visual demonstrations leads to a significant drop in performance, confirming that VLMs extract meaningful task information when provided with high-quality demonstrations. These results highlight the necessity of self-reflection, normalization, and multimodal demonstrations in achieving ELEMENTAL's stronger performance.

Under the same computational resource setup, EUREKA averaged 68.21 minutes of runtime across the nine tasks, while ELEMENTAL averaged 168.36 minutes. This increase is primarily due to the environment rollouts ELEMENTAL uses to estimate the reward gradient. However, this trade-off enables ELEMENTAL to learn a better-aligned reward function, leading to improved task success and generalization. We discuss future work to reduce runtime in Sec. 6.

5.2. Generalization Results

In the second set of experiments, we ask the question "are the LLMs/VLMs just remembering the reward function, as the knowledge cut-off date is Oct 2023, after IsaacGym is public?" To answer this question, we test the generalization capability of ELEMENTAL by applying it to modified versions of the IsaacGym environments, particularly variants of the Ant task. For these modified environments, we change certain properties, such as state vector order, physics property (e.g., gravity coefficient), and the task, to evaluate whether ELEMENTAL's VLM-driven feature inference combined with IRL can adapt to new, unseen environments better than EUREKA. We test on four Ant variants:

1. Ant Original: The standard Ant task without modifications, serving as the baseline environment.
2. Ant with Reversed Observations: The order of the state vector is reversed, testing the algorithms' ability to adapt to changes in the structure of the input data.
3. Ant with Lighter Gravity: The gravity coefficient is reduced from 9.81 to 5.00, requiring the features and policy to adjust for different dynamics. A performance drop is expected; the ant moves with less friction.
4. Ant Running Backward: The task is modified to require the Ant to move backward rather than forward, assessing how well the approaches generalize to a different task objective in the same environment.

We show the comparison between ELEMENTAL and EUREKA in Table 3. In out-of-distribution tasks that language models have not seen in the training set, EUREKA's performance declines, suggesting GPT-4o might have memorized the IsaacGym task rewards, which is not helpful when the state vector is reversed, when the environment dynamics change, or when the task objective is altered. In contrast, ELEMENTAL queries the VLM only for feature functions—information not available in the IsaacGym public data—and uses the demonstration-matching IRL process to determine the reward weights. ELEMENTAL accomplishes an average performance improvement of 41.3%, highlighting its robustness in adapting to changes in both the environment's physical properties and the task's nature.

5.3. Real-World User Study: Teaching Salad Mixing

To evaluate ELEMENTAL's real-world effectiveness, we conducted a within-subject user study where twelve participants taught a Kinova JACO arm to perform a salad mixing task. ELEMENTAL was able to operate interactively in real time: each learning round completed within 4 minutes using a remote server with an NVIDIA A40 GPU. Demonstrations were captured using a ZED stereo camera and automatically converted into ten equally spaced visual frames per skill.

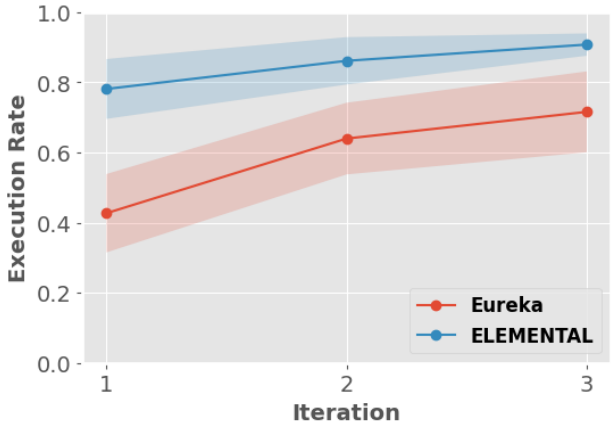


Figure 3. A comparison of the code execution rate between ELEMENTAL and EUREKA in three iterations in the nine IsaacGym environments. The shades show the standard error.

Participants taught three core skills: *go grasp mushroom*, *go drop at mixture bowl*, and *mix bowl with spoon*, using both ELEMENTAL and the baseline EUREKA (order randomized). Both methods used the same hardware, interaction rounds, and input text format, ensuring a fair comparison. The key distinction is that ELEMENTAL uses demonstrations and IRL to learn and refine reward functions, while EUREKA relies on LLM-generated reward code from language only. The remaining skills were predefined and shared across both conditions. Teaching followed two interaction rounds per method: for each skill, participants provided a kinesthetic demonstration and natural language intent description, observed robot execution, and gave feedback used to refine the feature function and/or the reward function. After teaching, participants observed blind executions of the full salad mixing task and evaluated both methods using 7-point Likert scales on two criteria: *task performance* (successful execution) and *strategy alignment* (intent matching). Each criterion included four questions (max score: 28).

ELEMENTAL significantly outperformed EUREKA on both metrics. Task scores were 20.58 ± 4.93 vs. 12.42 ± 4.72, $t(11) = -4.65$, $p < .001$; strategy scores were 19.83 ± 6.13 vs. 10.50 ± 4.32, $t(11) = -4.20$, $p < .001$. Participants remarked positively on ELEMENTAL's robustness to noisy demonstrations. For example, one participant noted: *"Even if my demonstration was slightly to the left of the mixture bowl, ELEMENTAL can help me fix this when I give it feedback and successfully put ingredients in the mixing bowl."*

These results support the hypothesis that ELEMENTAL aligns better with user intent, even under imperfect input. Moreover, the study further supports the system's scalability: no manual keyframe engineering was needed, and the same hyperparameters used in simulation were applied without tuning. Notably, the real-world pipeline required no hand-labeling—demonstrations were automatically extracted from ZED cameras, and ELEMENTAL operated entirely from these visual inputs and user feedback, without manual annotation. We provide an illustration of the user study setup in the supplementary material (Figure 4), with additional examples and insights in Appendix A.

6. Limitation and Future Work

While ELEMENTAL demonstrates strong improvements in LfD with VLMs, it has several limitations that open avenues for future research. One key limitation is its reliance on environment rollouts for IRL updates, which increases wall-clock running time. This is a trade-off that allows ELEMENTAL to infer reward functions that better align with demonstrations. Future work could explore more stable and efficient RL/IRL methods (Henderson et al., 2018; Hussenot et al., 2021; Adkins et al., 2024), such as Adversarial IRL (AIRL) or model-based RL, to reduce sample complexity. Additionally, parallelized or hardware-accelerated implementations could help mitigate runtime concerns, making ELEMENTAL more feasible for real-world deployment.

ELEMENTAL also assumes access to reasonably high-quality demonstrations. While our real-world user study showed that ELEMENTAL can recover from imperfect inputs through feedback, we observed that performance degrades with completely random or low-quality demonstrations (e.g., Table 1, ELEMENTAL w/ random visual demo). Future research could explore filtering and weighting strategies to better handle noisy or suboptimal demonstrations. For example, filtering could be based on user confidence, VLM scores, or automated ranking mechanisms. Moreover, recent advances in learning from suboptimal demonstrations (Brown et al., 2019; Chen et al., 2021; Beliaev et al., 2022) could be integrated into ELEMENTAL's IRL pipeline to further improve robustness.

Finally, given the demonstrated utility of visual inputs, future work could explore alternative demonstration modalities, such as continuous video, which may be more natural for users and improve generalization across robotics tasks. Incorporating real-time human feedback during the self-reflection process also remains a promising direction for adapting the feature and reward refinement loop.

7. Conclusion

We introduced ELEMENTAL, a novel framework that integrates VLMs with LfD to draft feature functions from visual demonstrations. ELEMENTAL leverages IRL to match feature functions with demonstrations and introduces an iterative self-reflection mechanism for continuous improvement. Experiments showed that ELEMENTAL outperforms previous state-of-the-art methods by 42.3% on standard IsaacGym tasks and 41.3% in generalizing to novel tasks.

Acknowledgement

We wish to thank our reviewers and area chairs for their valuable feedback in revising our manuscript. This work was supported by the National Science Foundation (NSF) under grant IIS-2340177, grant IIS-2112633, and grant CPS 2219755, as well as the National Institutes of Health (NIH) under grant 1R01HL157457.

Impact Statement

ELEMENTAL advances LfD by integrating VLMs with IRL to improve reward function engineering. Our contribution enables robots to learn from both natural language and visual demonstrations, reducing the need for hand-crafted reward functions and making robot learning more accessible to non-experts. By improving the alignment of learned behaviors with human demonstrations, ELEMENTAL has the potential to enhance human-robot interaction in areas such as assistive robotics, industrial automation, and adaptive learning systems.

Despite its benefits, ELEMENTAL, like all AI and robotics research, is a dual-use technology. While ELEMENTAL offers significant benefits, responsible use will require safeguards around demonstration quality, oversight of iterative reward refinement, and transparency in how VLM-derived features influence learned behaviors. Additionally, while ELEMENTAL improves transparency in reward learning, its dependence on VLMs introduces potential challenges such as hallucinations or overfitting to language priors.

Another ethical consideration is the potential misuse of ELEMENTAL for applications beyond its intended scope. While the framework is designed for positive use cases, such as making AI training more interpretable and user-aligned, the ability to iteratively refine reward functions could be leveraged in ways that raise ethical concerns, such as in surveillance or military applications. We advocate for responsible AI development and encourage the use of human-in-the-loop safeguards, bias auditing, and clear ethical guidelines for deployment.

Overall, ELEMENTAL represents a step toward more intuitive and generalizable AI systems by integrating demonstrations with language guidance. While its potential benefits are significant, careful attention must be given to ensuring fairness, transparency, and responsible use as these technologies continue to evolve.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In Brodley, C. E. (ed.), *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004. doi: 10.1145/1015330.1015430. URL https://doi.org/10.1145/1015330.1015430.

Abouelazm, A., Michel, J., and Zoellner, J. M. A review of reward functions for reinforcement learning in the context of autonomous driving. *ArXiv preprint*, abs/2405.01440, 2024. URL https://arxiv.org/abs/2405.01440.

Adeniji, A., Xie, A., Sferrazza, C., Seo, Y., James, S., and Abbeel, P. Language reward modulation for pretraining reinforcement learning. *ArXiv preprint*, abs/2308.12270, 2023. URL https://arxiv.org/abs/2308.12270.

Adkins, J., Bowling, M., and White, A. A method for evaluating hyperparameter sensitivity in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:124820–124842, 2024.

Arora, S. and Doshi, P. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

Beliaev, M., Shih, A., Ermon, S., Sadigh, D., and Pedarsani, R. Imitation learning by estimating expertise of demonstrators. In *International Conference on Machine Learning*, pp. 1732–1748. PMLR, 2022.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *ArXiv preprint*, abs/2108.07258, 2021. URL https://arxiv.org/abs/2108.07258.

Booth, S., Knox, W. B., Shah, J., Niekum, S., Stone, P., and Allievi, A. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Brown, D. S., Goo, W., Nagarajan, P., and Niekum, S. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 783–792. PMLR, 2019. URL http://proceedings.mlr.press/v97/brown19a.html.

Chen, L., Paleja, R., Ghuy, M., and Gombolay, M. Joint goal and strategy inference across heterogeneous demonstrators via reward network distillation. In *Proceedings of the*

International Conference on Human-Robot Interaction (HRI), pp. 659–668, 2020.

Chen, L., Paleja, R., and Gombolay, M. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on robot learning*, pp. 1262–1277. PMLR, 2021.

Chen, L., Jayanthi, S., Paleja, R. R., Martin, D., Zakharov, V., and Gombolay, M. Fast lifelong adaptive inverse reinforcement learning from demonstrations. In *Proceedings of Conference on Robot Learning (CoRL)*, 2022.

Chernova, S. and Thomaz, A. L. *Robot Learning from Human Teachers*. Morgan & Claypool Publishers, 2014.

Chi, V. B. and Malle, B. F. Interactive human-robot teaching recovers and builds trust, even with imperfect learners. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, pp. 127–136, 2024.

Di Stefano, G., Gino, F., Pisano, G., and Staats, B. Learning by thinking: How reflection improves performance. *Harvard Business Review. HBS Working Paper*, 2014.

Du, Y., Konyushkova, K., Denil, M., Raju, A., Landon, J., Hill, F., de Freitas, N., and Cabi, S. Vision-language models as success detectors. *ArXiv preprint*, abs/2303.07280, 2023. URL https://arxiv.org/abs/2303.07280.

Ericsson, K. A. and Simon, H. A. Verbal reports as data. *Psychological review*, 87(3):215, 1980.

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

Feldon, D. F. The implications of research on expertise for curriculum and pedagogy. *Educational Psychology Review*, 19:91–110, 2007.

Fu, J., Luo, K., and Levine, S. Learning robust rewards with adverserial inverse reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rkHywl-A-.

Gupta, D., Chandak, Y., Jordan, S., Thomas, P. S., and C da Silva, B. Behavior alignment via reward function optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3207–3214. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16669.

Hoffman, R. R., Shadbolt, N. R., Burton, A. M., and Klein, G. Eliciting knowledge from experts: A methodological analysis. *Organizational behavior and human decision processes*, 62(2):129–158, 1995.

Hussenot, L., Andrychowicz, M., Vincent, D., Dadashi, R., Raichuk, A., Ramos, S., Momchev, N., Girgin, S., Marinier, R., Stafiniak, L., et al. Hyperparameter selection for imitation learning. In *International Conference on Machine Learning*, pp. 4511–4522. PMLR, 2021.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Kwon, M., Xie, S. M., Bullard, K., and Sadigh, D. Reward design with language models. *ArXiv preprint*, abs/2303.00001, 2023. URL https://arxiv.org/abs/2303.00001.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Lin, J., Du, Y., Watkins, O., Hafner, D., Abbeel, P., Klein, D., and Dragan, A. Learning to model the world with language. *ArXiv preprint*, abs/2308.01399, 2023. URL https://arxiv.org/abs/2308.01399.

Locke, E. A. Social foundations of thought and action: A social-cognitive view, 1987.

Ma, Y. J., Kumar, V., Zhang, A., Bastani, O., and Jayaraman, D. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning*, pp. 23301–23320. PMLR, 2023a.

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

Mahadevan, K., Chien, J., Brown, N., Xu, Z., Parada, C., Xia, F., Zeng, A., Takayama, L., and Sadigh, D. Generative expressive robot behaviors using large language models. *ArXiv preprint*, abs/2401.14673, 2024. URL https://arxiv.org/abs/2401.14673.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *ArXiv preprint*, abs/2108.10470, 2021. URL https://arxiv.org/abs/2108.10470.

Matignon, L., Laurent, G. J., and Le Fort-Piat, N. Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *Artificial Neural Networks – ICANN 2006*, pp. 840–849, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

Myers, V., Zheng, B. C., Mees, O., Levine, S., and Fang, K. Policy adaptation via language optimization: Decomposing tasks for few-shot imitation. *ArXiv preprint*, abs/2408.16228, 2024. URL https://arxiv.org/abs/2408.16228.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. R3m: A universal visual representation for robot manipulation. *ArXiv preprint*, abs/2203.12601, 2022. URL https://arxiv.org/abs/2203.12601.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In Langley, P. (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 663–670. Morgan Kaufmann, 2000.

Nisbett, R. E. and Wilson, T. D. Telling more than we can know: Verbal reports on mental processes. *Psychological review*, 84(3):231, 1977.

Peng, A., Bobu, A., Li, B. Z., Sumers, T. R., Sucholutsky, I., Kumar, N., Griffiths, T. L., and Shah, J. A. Preference-conditioned language-guided abstraction. *ArXiv preprint*, abs/2402.03081, 2024a. URL https://arxiv.org/abs/2402.03081.

Peng, A., Li, B. Z., Sucholutsky, I., Kumar, N., Shah, J. A., Andreas, J., and Bobu, A. Adaptive language-guided abstraction from contrastive explanations. *ArXiv preprint*, abs/2409.08212, 2024b. URL https://arxiv.org/abs/2409.08212.

Piccolo, S. R., Denny, P., Luxton-Reilly, A., Payne, S. H., and Ridge, P. G. Evaluating a large language model's ability to solve programming exercises from an introductory bioinformatics course. *PLOS Computational Biology*, 19 (9):e1011511, 2023.

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3, 2020.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Sontakke, S., Zhang, J., Arnold, S., Pertsch, K., Bıyık, E., Sadigh, D., Finn, C., and Itti, L. Roboclip: One demonstration is enough to learn robot policies. *Advances in Neural Information Processing Systems*, 36, 2024.

Suay, H. B., Brys, T., Taylor, M. E., and Chernova, S. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pp. 429–437, 2016.

Tai, L., Paolo, G., and Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 31–36. IEEE, 2017.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971, 2023. URL https://arxiv.org/abs/2302.13971.

Wang, Y., Sun, Z., Zhang, J., Xian, Z., Biyik, E., Held, D., and Erickson, Z. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *ArXiv preprint*, abs/2402.03681, 2024. URL https://arxiv.org/abs/2402.03681.

Wang, Y.-J., Zhang, B., Chen, J., and Sreenath, K. Prompt a robot to walk with large language models. *ArXiv preprint*, abs/2309.09969, 2023. URL https://arxiv.org/abs/2309.09969.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *ArXiv preprint*, abs/2206.07682, 2022. URL https://arxiv.org/abs/2206.07682.

Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Gonzalez Arenas, M., Lewis Chiang, H.-T., Erez, T., Hasenclever, L., Humplik, J., Ichter, B., Xiao, T., Xu, P., Zeng, A., Zhang, T., Heess, N., Sadigh, D., Tan, J., Tassa, Y., and Xia, F. Language to rewards for robotic skill synthesis. *ArXiv preprint*, abs/2306.08647, 2023. URL https://arxiv.org/abs/2306.08647.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364. IEEE, 2017.

Zhu, Z. and Hu, H. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2), 2018. ISSN 2218-6581. doi: 10.3390/robotics7020017. URL https://www.mdpi.com/2218-6581/7/2/17.

Ziebart, B. D. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the National Conference on Artificial intelligence (AAAI)*, pp. 1433–1438, 2008.

A. User Study Details

To evaluate ELEMENTAL in practical, real-world settings, we conducted a user study. This section provides additional details on the task setup and experimental design.

A.1. Task and Robot Setup

The study centers around a salad mixing task, where participants teach a Kinova JACO robotic arm to sequentially transport ingredients and perform a mixing motion. To enable targeted evaluation within a limited study duration, we decomposed the task into modular skills—three of which were taught by participants, while others were predefined.

Taught skills (via ELEMENTAL and EUREKA):

1. **Go to mushroom**: Navigate to the mushroom ingredient.
2. **Go to mixture bowl**: Move to the bowl location in preparation to drop ingredients.
3. **Mix bowl with spoon**: Stir the bowl contents using a spoon.

Predefined skills:

1. **Go to pepper, Go to tomato, Go to spoon, Go to home**
2. **Pick / Drop**: Grasp or release ingredients or utensils.

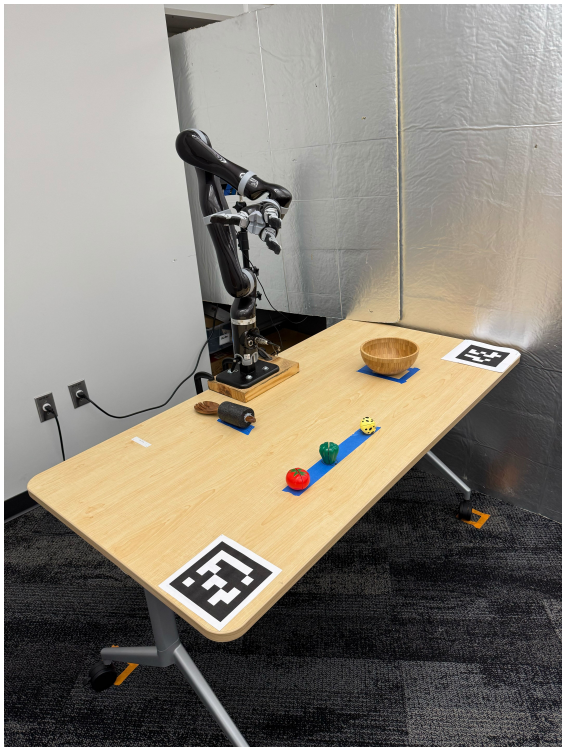


Figure 4. Real-world salad-mixing user study setup. Users taught three skills: *go grasp mushroom*, *go drop at mixture bowl*, and *mix bowl with spoon*.

The robot was controlled via low-level joint position commands. Visual demonstrations were automatically extracted from ZED stereo camera footage as ten equally spaced keyframes per skill, requiring no manual annotation. This setup enabled scalable and consistent demonstration capture across participants.

A.2. Experiment Design

We used a within-subject design where each of the twelve participants taught the three skills using both ELEMENTAL and EUREKA, with method order randomized. Each teaching session consisted of two interaction rounds per method: participants first provided a kinesthetic demonstration and a textual intent description, then observed a robot rollout and gave natural language feedback. The learning algorithm used this input to refine the reward and policy in EUREKA, or the feature representation, reward, and policy in ELEMENTAL. This observation–feedback loop was repeated twice for each skill and each method.

After completing all skill teachings, participants watched blind rollouts of the full salad mixing task using both systems. Each rollout followed the full sequence of ingredient pick-and-drop and bowl mixing using both learned and predefined skills. Participants then completed a final survey evaluating both methods on two dimensions: *task performance* (e.g., successful execution of actions) and *strategy alignment* (e.g., matching user preferences). Each criterion was rated via four 7-point Likert scale questions (range 4–28). ELEMENTAL significantly outperformed EUREKA on both metrics, as shown in Figure 5.

A.3. System Responsiveness

Despite relying on IRL and VLM-based reward learning, ELEMENTAL was tuned to support interactive teaching. Each learning round—including training and policy/reward refinement—was completed in under 4 minutes using a remote server with an NVIDIA A40 GPU. The same hyperparameters were used across simulation and real-world settings, demonstrating ELEMENTAL's sample efficiency and robustness without task-specific tuning.

A.4. Example in Reducing Language Ambiguity through Demonstrations

A key motivation for ELEMENTAL is that visual demonstrations can reduce ambiguity in underspecified natural language instructions by grounding them in observable behavior. In our user study, participants frequently used vague spatial or temporal terms when describing tasks. For instance, when teaching the *mix bowl with spoon* skill, one participant instructed: *"First, the robot should lower its gripper toward the inside of the bowl with the spoon pointing downward. Then, the robot should move in a way to make the spoon move in a circular motion for mixing."* This instruction includes vague temporal phrases and spatial instructions.

EUREKA, which relies solely on language, interpreted this instruction by constructing a reward function that encourages a fixed orientation using a static dot-product term, as shown in Box 1.

BOX 1: EUREKA REWARD FUNCTION.

```
def compute_reward(ee_pos: torch.Tensor, bowl_position_tensor:
↪  torch.Tensor) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # other contents omitted

    # Orientation reward
    ee_orientation = ee_pos[:, 3:7]
    dot_product = torch.abs(torch.sum(ee_orientation * desired_orientation,
    ↪  dim=-1))
    orientation_reward = torch.exp(orientation_reward_temp * (dot_product -
    ↪  1))
```

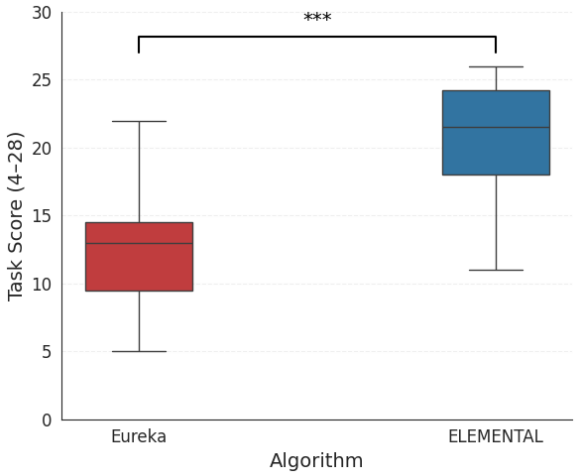
In contrast, ELEMENTAL incorporates visual demonstrations to disambiguate when and how the robot should reorient (Box 2). Based on the demonstration, it infers that the robot should only begin reorienting when far from the bowl, according to the "first... then..." temporal structure. This feature shows that the VLM correctly inferred that reorientation should occur only when the robot is far from the bowl, capturing the "first... then..." structure described in the language.

BOX 2: ELEMENTAL FEATURE FUNCTION.

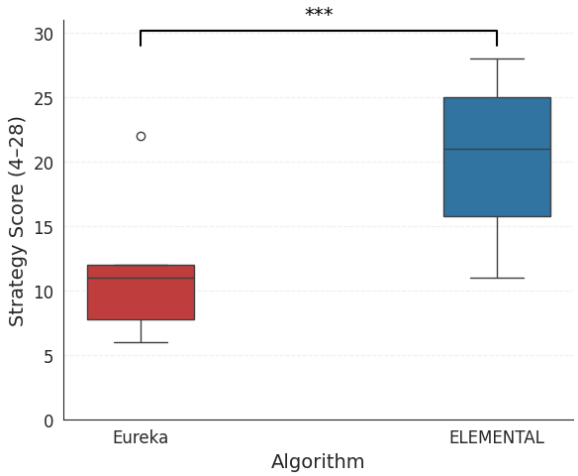
```
def compute_feature(obs_buf: torch.Tensor) -> Dict[str, torch.Tensor]:
    # other contents omitted

    # 3. Reorient while distant to avoid collision
    down_direction = torch.tensor([0.0, 0.0, -1.0], device=obs_buf.device)
    orientation_similarity_far =
    ↪  torch.nn.functional.cosine_similarity(ee_orientation[:, :3],
    ↪  down_direction.unsqueeze(0), dim=-1)
    is_far = distance_to_bowl >= 0.2
    reorientation_early = torch.where(is_far, orientation_similarity_far,
    ↪  torch.tensor(0.0, device=obs_buf.device))
```

This comparison demonstrates how ELEMENTAL reduces ambiguity by grounding vague or implicit instructions. Whereas EUREKA encodes the user's description as a constant reward over orientation, ELEMENTAL uses demonstration context to condition behavior on spatial distance—capturing temporal dependencies not specified explicitly in language.

Such distinctions are not just academic: participants frequently expressed uncertainty when relying on language alone. One participant asked, *"Should I describe this (for example, moving to the left/right side) from my perspective or the robot's?"* This highlights the challenge of specifying tasks precisely through language. ELEMENTAL mitigates this challenge by aligning demonstrations with language, enabling more robust and intent-aligned reward inference.



(a) Comparison of task scores between Eureka and ELEMENTAL. Participants rated how well the robot completed the overall task after watching the full execution sequence. ELEMENTAL achieved significantly higher task scores.



(b) Comparison of strategy scores between Eureka and ELEMENTAL. Participants rated how closely the robot's execution matched their intended strategies. ELEMENTAL outperformed Eureka with statistical significance.

Figure 5. Participant ratings from the final evaluation phase. ELEMENTAL significantly outperformed Eureka on both task success and strategy alignment. Error bars denote interquartile ranges; stars indicate statistical significance (*** $p < .001$).

B. Prompts

We show the prompts used in ELEMENTAL for feature drafting and self-reflection in Box 3–6.

BOX 3: INITIAL SYSTEM PROMPT.

```
You are a feature engineer trying to write relevant features for the reward
↪  function to solve learning-from-demonstration (inverse reinforcement
↪  learning) tasks as effective as possible.

Your goal is to write a feature function for the environment that will help
↪  the agent construct a linear reward function with the constructed
↪  features via inverse reinforcement learning to accomplish the task
↪  described in text and the demonstration.

Your feature function should use useful variables from the environment as
↪  inputs. The feature function signature must follow:
@torch.jit.script
def compute_feature(obs_buf: torch.Tensor) -> Dict[str, torch.Tensor]:
    ...
    return {}

Since the feature function will be decorated with @torch.jit.script, please
↪  make sure that the code is compatible with TorchScript (e.g., use torch
↪  tensor instead of numpy array).

You should not wrap the function within a class.

Make sure any new tensor or variable you introduce is on the same device as
↪  the input tensors.
```

BOX 4: INITIAL USER PROMPT.

```
The Python environment is
{task_obs_code_string}

Write a feature function for the following task: {task_description}.

Three keyframes of a demonstration for how to accomplish the task are shown
↪  in the image (superimposing agent pos).
```

BOX 5: CODE OUTPUT INSTRUCTION.

```
The input of the feature function is a torch.Tensor named `obs_buf` that is
↪  a batched state (shape: [batch, num_obs]).

The output of the feature function should be a dictionary where the keys
↪  are the names of the features and the values are the corresponding
↪  feature values for the input state.

You must respect the function signature.

The code output should be formatted as a python code string: "```python ...
↪  ```".

Some helpful tips for writing the feature function code:

    (1) You may find it helpful to normalize the features to a fixed range
    ↪  by applying transformations

    (2) The feature code's input variables must be obs_buf: torch.Tensor,
    ↪  which corresponds to the state observation (self.obs_buf) returned
    ↪  by the environment compute_observations() function. Under no
    ↪  circumstance can you introduce new input variables.

    (3) Each output feature should only one a single dimension (shape:
    ↪  [batch]).

    (4) You should think step-by-step: first, think what is important in
    ↪  the task based on the task description and the demonstration and
    ↪  come up with names of the features, then, write code to calculate
    ↪  each feature

    (5) You should be aware that the downstream inverse reinforcement
    ↪  learning only creates reward functions that are linear function of
    ↪  the constructed features; thus, it is important to construct
    ↪  expressive features that humans do care in this task

    (6) Do not use unicode anywhere such as \u03c0 (pi)
```

BOX 6: SELF-REFLECTION PROMPT.

```
We trained reward and policy via inverse reinforcement learning using the
↪  provided feature function code with the demonstration.

We tracked the feature values as well as episode lengths.

The mean values of the last {eval_avg_horizon} steps from the learned
↪  policy are:
{insert}

Please carefully analyze the feedback and provide a new, improved feature
↪  function that can better solve the task. Some helpful tips for
↪  analyzing the feedback:

    (1) If the episode lengths are low, it likely means the policy is
    ↪  unsuccessful

    (2) If the feature counts are significantly different between demo and
    ↪  learned behavior, then this means IRL cannot match this feature
    ↪  with the demo as it is written. You may consider

        (a) Change its scale

        (b) Re-writing the feature: check error in the feature computation
        ↪  (e.g., indexing the observation vector) and be careful about
        ↪  outlier values that may occur in the computation

        (c) Discarding the feature

    (3) If a feature has near-zero weight, the feature may be unimportant.
    ↪  You can consider discarding the feature or rewriting it.

    (4) You may add/remove features as you see appropriate.

Please analyze each existing features in the suggested manner above first,
↪  and then write the feature function code.
```

Table 4. This table shows benchmarking results on IsaacGym tasks between ELEMENTAL and Eureka. Each cell reports the mean (standard deviation) across five seeds. The statistical significance shows $*$ if $p < .05$ and $**$ if $p < .01$. The statistical test (p) shows the p-value for independent t-test (one-sided) for comparisons that satisfies the normality and homoscedasticity assumptions, and otherwise shows the p-values for Mann-Whitney U-test (denoted with †). Bold denotes the better performance between ELEMENTAL and Eureka. Additionally, we added a baseline, VLM+BC, which utilizes the VLM drafted feature functions to train Behavior Cloning (BC) by transforming all observations into the feature space. We also include preliminary results of ELEMENTAL and Eureka with OpenAI o1 model with 1-5 seed experiments done to show ablation on the VLM used.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
EUREKA	192.66 (26.17)	412.16 (33.20)	**-0.27 (0.10)**	0.10 (0.14)	4.09 (2.26)	3.69 (0.56)	-2.14 (1.59)	2.36 (4.91)	0.00 (0.00)
ELEMENTAL (Ours)	**258.50 (42.67)**	**438.90 (43.64)**	-0.44 (0.15)	**0.18 (0.17)**	**7.00 (1.03)**	**4.70 (0.12)**	**-0.37 (0.34)**	**20.54 (3.98)**	**1.57 (1.35)**
Statistical significance	**				*	**	*	**	**
Statistical test (p)	0.009	0.201†	0.970	0.265†	0.015	0.002	0.021	0.004†	0.004†
VLM+BC	80.95 (17.19)	68.77 (17.01)	-1.44 (0.12)	0.00 (0.00)	0.01 (0.09)	0.08 (0.23)	-2.5 (0.27)	0.00 (0.00)	0.01 (0.01)
EUREKA o1	143.49 (24.70)	434.14 (12.99)	-0.26 (0.03)	0.02 (0.00)	6.91 (0.42)	3.33 (1.19)	-1.78 (1.59)	8.43 (2.46)	2.14 (0.91)
ELEMENTAL o1	264.92 (69.70)	456.51 (26.16)	-0.77 (0.15)	0.06 (0.01)	7.00 (1.05)	4.52 (0.33)	-0.54 (0.05)	21.62 (0.73)	2.91 (1.16)

Table 5. This table shows benchmarking results on IsaacGym tasks between ELEMENTAL and Eureka. Each cell reports the max across five seeds. Bold denotes the better performance.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
EUREKA	215.91	454.18	**-0.18**	0.34	6.88	4.36	-0.70	11.12	0.00
ELEMENTAL (Ours)	**309.38**	**474.63**	-0.30	**0.36**	**8.49**	**4.81**	**-0.02**	**23.38**	**2.71**

C. Additional Simulated Results

In this section, we present expanded experimental results that provide deeper insight into the performance of ELEMENTAL relative to baselines across benchmark and generalization tasks. These results are computed across five random seeds with statistical tests to assess the significance of the observed improvements.

Benchmark Performance (Table 4 and 5). Table 4 reports the mean and standard deviation of task performance across five seeds on the nine IsaacGym environments. ELEMENTAL achieves better performance than EUREKA on eight out of nine tasks with statistically significant improvements ($p < .05$ or $p < .01$) in five tasks. Table 5 further confirms these gains by showing that ELEMENTAL reaches a higher maximum performance than EUREKA in eight out of nine tasks. We also include results from a new VLM+BC baseline, which trains a behavior cloning policy using features extracted by VLMs. This baseline performs poorly across all tasks, underscoring that simply combining VLM-generated features with demonstrations is insufficient—BC suffers from covariate shift and lacks ELEMENTAL's self-reflection loop, which iteratively refines the feature function.

Generalization to Variants (Table 6 and 7). We also assess the generalization capabilities of ELEMENTAL by evaluating performance on four Ant-environment variants. Table 6 shows that ELEMENTAL outperforms EUREKA in all four variants, achieving statistical significance in two of them ($p < .05$). The consistent improvement across these perturbed settings demonstrates that ELEMENTAL is better equipped to handle out-of-distribution environments, due in part to the tighter alignment between learned rewards and demonstrations enabled by IRL. Table 7 confirms that ELEMENTAL attains superior max performance across all variants.

Reward Function Quality (Table 8 and 9). Tables 8 and 9 compare the reward correlation between learned reward functions and the ground-truth rewards provided in IsaacGym. ELEMENTAL yields better or comparable correlation in all tasks. These results support the claim that ELEMENTAL produces reward functions that are more consistently aligned with task objectives due to its iterative self-reflection loop and embedded IRL that improve feature and reward alignment.

Table 6. This table compares the generalization performance of ELEMENTAL and EUREKA on Ant-variant environments. Each cell reports the mean (standard deviation) across five seeds. The statistical significance shows $*$ if $p < .05$ and $**$ if $p < .01$. The statistical test (p) shows the p-value for independent t-test (one-sided) for comparisons that satisfies the normality and homoscedasticity assumptions, and otherwise shows the p-values for Mann-Whitney U-test (denoted with †). Bold denotes the better performance.

Method	Ant Original	w/ Reversed Obs	w/ Lighter Gravity	Ant Running Backward
EUREKA	4.09 (2.26)	2.56 (2.93)	2.64 (1.04)	3.94 (2.76)
ELEMENTAL	**7.00 (1.03)**	**5.71 (3.23)**	**3.77 (1.68)**	**7.41 (1.25)**
Statistical significance	*			*
Statistical test (p)	0.015	0.072	0.119	0.017

Table 7. This table compares the generalization performance of ELEMENTAL and EUREKA on Ant-variant environments. Each cell reports the max across five seeds. Bold denotes the better performance.

Method	Ant Original	w/ Reversed Obs	w/ Lighter Gravity	Ant Running Backward
EUREKA	6.88	5.96	4.39	6.90
ELEMENTAL	**8.49**	**8.47**	**5.89**	**9.30**

Ablation: VLM Variants. We also report results using OpenAI's o1 model, in addition to the GPT-4o-based results in Table 4. ELEMENTAL continues to outperform EUREKA when both use the o1 model (seven out of nine tasks, on average 37% gain), suggesting that ELEMENTAL's architecture is robust to the specific VLM backbone and benefits from the integration of demonstrations and iterative refinement regardless of the underlying model.

C.1. Case Study

In this subsection, we present a case study illustrating the iterative process of ELEMENTAL on the Humanoid task. The initial feature function drafted by the Vision-Language Model (VLM) is shown in Box 7. The proposed features—forward_velocity, uprightness, and heading_alignment—are well-aligned with the task objectives of running efficiently while maintaining balance and direction. These features provide a strong starting point for the learning process.

Using this initial feature function, ELEMENTAL trains the IRL process, calculates the feature counts for both the generated trajectories and the demonstration, and feeds this feedback back to the VLM, as shown in Box 8. The feedback reveals key discrepancies, such as lower forward_velocity. Based on this analysis, the VLM revises the feature function, as shown in Box 9. Notably, the revised function introduces a new feature, lateral_velocity, which captures stride consistency by taking the absolute value of the lateral movement. This demonstrates the VLM's capability to construct nonlinear features, expanding the expressiveness of the feature function.

Finally, ELEMENTAL trains the IRL process again using the updated feature function and compiles the feedback, as shown in Box 10. The resulting reward weights assign the highest importance to forward_velocity, with relatively minor contributions from stability-related objectives such as uprightness and lateral_velocity. This distribution aligns well with human intuition for the task, where speed is the primary objective, and stability features serve as secondary constraints. The evolvement of the feature function is illustrated in Figure 6.

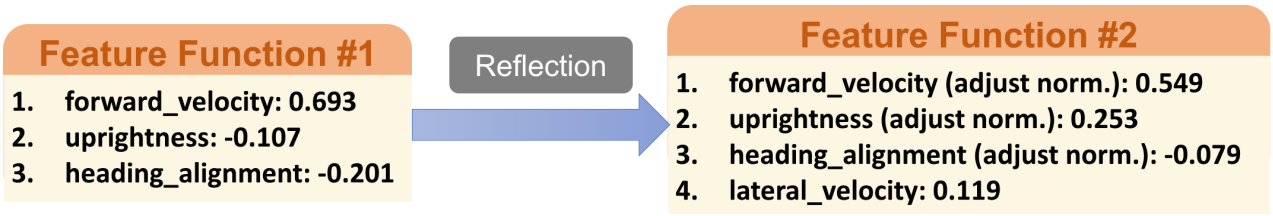


Figure 6. This figure illustrates the feature evolution in one example.

This case study highlights the potential of ELEMENTAL to produce models at every stage of its pipeline. The feature functions generated by the VLM are human-readable and meaningful, allowing practitioners to inspect and refine them as needed. Additionally, the linear weights learned by ELEMENTAL during the IRL process indicate the relative importance

Table 8. This table shows reward correlation of inferred reward functions for EUREKA and ELEMENTAL (ours) on IsaacGym tasks. Each cell reports the mean (standard deviation) across five seeds.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
EUREKA	0.77 (0.09)	0.48 (0.61)	0.94 (0.02)	0.49 (0.36)	0.97 (0.04)	0.84 (0.17)	0.32 (0.70)	0.32 (0.40)	0.00 (0.00)
ELEMENTAL (Ours)	0.77 (0.15)	0.80 (0.16)	0.32 (0.64)	0.73 (0.33)	0.85 (0.17)	0.84 (0.11)	0.93 (0.08)	0.61 (0.20)	0.27 (0.42)

Table 9. This table shows reward correlation of inferred reward functions for EUREKA and ELEMENTAL (ours) on IsaacGym tasks. Each cell reports the max across five seeds.

Method	IsaacGym Environments								
	Cartpole	Ball Balance	Quadcopter	Franka Cabinet	Ant	Humanoid	Anymal	Allegro Hand	Shadow Hand
EUREKA	0.85	0.95	0.97	0.93	1.00	0.98	0.99	0.77	0.00
ELEMENTAL (Ours)	0.99	0.95	0.89	0.98	1.00	0.98	1.00	0.77	0.82

of each feature in matching the demonstration. The iterative self-reflection mechanism ensures that the learned policies and reward functions evolve in alignment with the demonstration.

BOX 7: 1ST ROUND VLM RESULT

````
To create a feature function aimed at helping a humanoid robot run as fast as possible, we should focus on features that are
↪  most relevant to running speed and ensuring the humanoid's balance and stability. Useful features could include the
↪  forward velocity, balance (uprightness), and perhaps a component that ensures the humanoid maintains forward momentum
↪  without falling.

Here's the feature function based on the observation buffer structure:

```python
import torch
from typing import Dict

@torch.jit.script
def compute_feature(obs_buf: torch.Tensor) -> Dict[str, torch.Tensor]:
    # Extract relevant features from the observation buffer
    # Assuming obs_buf shape: [batch_size, num_obs]

    # Feature: forward velocity (x direction in local frame)
    forward_velocity = obs_buf[:, 1]

    # Feature: uprightness (use the up projection)
    uprightness = obs_buf[:, 10]

    # Feature: heading alignment (use the heading projection)
    heading_alignment = obs_buf[:, 11]

    # Normalize the features
    # Assuming value normalizations such as the velocity range, uprightness [-1, 1], heading alignment [-1, 1]
    max_forward_velocity = 10.0  # This is an arbitrary normalization
    forward_velocity = forward_velocity / max_forward_velocity  # Normalizing forward velocity

    # Combine features into output dictionary
    features = {
        "forward_velocity": forward_velocity,
        "uprightness": uprightness,
        "heading_alignment": heading_alignment,
    }

    return features
```

### Explanation:

1. **`forward_velocity`**: The observation buffer's second column (`obs_buf[:, 1]`) captures the forward velocity in the
↪  local frame.

2. **`uprightness`**: The observation buffer's 11th column (`obs_buf[:, 10]`) captures the upright projection, which is
↪  crucial to ensure the humanoid isn't falling over.

3. **`heading_alignment`**: The observation buffer's 12th column (`obs_buf[:, 11]`) captures heading alignment, which helps
↪  ensure the humanoid is moving forward in the correct direction.

These features will enable the inverse reinforcement learning process to construct a reward function that emphasizes running
↪  fast while maintaining balance and a straight heading.
````

BOX 8: 1ST ROUND FEEDBACK

```
Calculating mean features on the demonstration results in the following:
episode_lengths: 1000.0
forward_velocity: 0.57
uprightness: 0.94
heading_alignment: 0.91
We trained reward and policy via inverse reinforcement learning using the provided feature function code with the
↪  demonstration.
We tracked the feature values as well as episode lengths.
The mean values of the last 100 steps from the learned policy are:
forward_velocity: 0.4
uprightness: 0.99
heading_alignment: 0.99
IRL reward: 0.05
episode_lengths: 691.47
IRL feature weights: {'forward_velocity': 0.693, 'uprightness': -0.107, 'heading_alignment': -0.201}
Please carefully analyze the feedback and provide a new, improved feature function that can better solve the task. Some
↪  helpful tips for analyzing the feedback:
    (1) If the episode lengths are low, it likely means the policy is unsuccessful
    (2) If the feature counts are significantly different between demo and learned behavior, then this means IRL cannot match
    ↪  this feature with the demo as it is written. You may consider
        (a) Change its scale
        (b) Re-writing the feature: check error in the feature computation (e.g., indexing the observation vector) and be
        ↪  careful about outlier values that may occur in the computation
        (c) Discarding the feature
    (3) If a feature has near-zero weight, the feature may be unimportant. You can consider discarding the feature or
    ↪  rewriting it.
    (4) You may add/remove features as you see appropriate.
Please analyze each existing features in the suggested manner above first, and then write the feature function code.The input
↪  of the feature function is a torch.Tensor named `obs_buf` that is a batched state (shape: [batch, num_obs]).
The output of the feature function should be a dictionary where the keys are the names of the features and the values are the
↪  corresponding feature values for the input state.
You must respect the function signature.
The code output should be formatted as a python code string: "```python ... ```".

Some helpful tips for writing the feature function code:
    (1) You may find it helpful to normalize the features to a fixed range by applying transformations
    (2) The feature code's input variables must be obs_buf: torch.Tensor, which corresponds to the state observation
    ↪  (self.obs_buf) returned by the environment compute_observations() function. Under no circumstance can you introduce
    ↪  new input variables.
    (3) Each output feature should only one a single dimension (shape: [batch]).
    (4) You should think step-by-step: first, think what is important in the task based on the task description and the
    ↪  demonstration and come up with names of the features, then, write code to calculate each feature
    (5) You should be aware that the downstream inverse reinforcement learning only creates reward functions that are linear
    ↪  function of the constructed features; thus, it is important to construct expressive features that humans do care in
    ↪  this task
    (6) Do not use unicode anywhere such as \u03c0 (pi)
```

BOX 9: 2ND ROUND VLM RESULT

````
Based on the provided feedback and analysis of how the current features are performing, we can observe a few key points:

1. **Forward Velocity:**
   - The policy is achieving a lower forward velocity than in the demonstration. This suggests that the current method of
   ↪  calculating or normalizing this feature might need adjustment.
   - The forward velocity weight is quite significant (`0.693`).

2. **Uprightness:**
   - The policy has a higher uprightness than the demonstration. The weight for uprightness is negative (`-0.107`), which may
   ↪  indicate that the model learns to be overly cautious about staying upright, potentially at the expense of moving fast.
   ↪  This negative weight suggests we might need to rethink how this feature is influencing the reward.

3. **Heading Alignment:**
   - The policy has a higher heading alignment than the demonstration. Similar to uprightness, the weight is also negative
   ↪  and relatively significant (`-0.201`), indicating it might be overemphasizing this feature.

Given these points, we aim to improve the feature function by:
1. Adjusting the scaling or normalization of the forward velocity.
2. Revisiting and potentially revising the uprightness and heading alignment features to better match the demonstration
↪  characteristics.
3. Considering new or additional features that might encourage the desired behavior such as stability or stride consistency.

Here is the improved feature function:

```python
import torch
from typing import Dict

@torch.jit.script
def compute_feature(obs_buf: torch.Tensor) -> Dict[str, torch.Tensor]:
    # Extract relevant features from the observation buffer
    # Assuming obs_buf shape: [batch_size, num_obs]

    # Feature: forward velocity (x direction in local frame)
    forward_velocity = obs_buf[:, 1]

    # Adjust scaling of forward velocity to emphasize higher speeds, assuming range [-1, 1]
    max_forward_velocity = 1.0  # New scaling factor to more accurately reflect achievable max speeds
    forward_velocity = forward_velocity / max_forward_velocity  # Normalizing forward velocity

    # Feature: uprightness (use the up projection)
    uprightness = obs_buf[:, 10]

    # Re-normalize uprightness to give more incentive to be upright but not be overly conservative
    uprightness = (uprightness - 0.5) * 2  # Adjusting the range to new scaling [-1, 1]

    # Feature: heading alignment (use the heading projection)
    heading_alignment = obs_buf[:, 11]

    # Normalize heading alignment to confirm alignment direction
    heading_alignment = (heading_alignment - 0.5) * 2  # Adjusting the range similarly

    # Introducing a new feature: lateral velocity (to ensure stability)
    lateral_velocity = obs_buf[:, 2]
    lateral_velocity = torch.abs(lateral_velocity)  # Use absolute value to avoid penalizing/stabilize sideway motions

    # Normalize lateral velocity, supposing it ranges [-1, 1]
    max_lateral_velocity = 1.0
    lateral_velocity = lateral_velocity / max_lateral_velocity

    # Combine features into output dictionary
    features = {
        "forward_velocity": forward_velocity,
        "uprightness": uprightness,
        "heading_alignment": heading_alignment,
        "lateral_velocity": lateral_velocity
    }

    return features
```

### Explanation:

1. **`forward_velocity`**: Adjusted scaling to make achievable velocities more impactful.
2. **`uprightness`**: Adjusted to a new normalization to try to balance upright positioning without making the humanoid
↪  overly cautious.
3. **`heading_alignment`**: Similar adjustment for normalization.
4. **`lateral_velocity`**: Added a new feature to penalize sideways movement, ensuring stability and straightforward motion.

These changes aim to provide a more nuanced reward function, hopefully promoting behaviors closer to the demonstration while
↪  balancing speed and stability.
````

BOX 10: 2ND ROUND FEEDBACK

```
Calculating mean features on the demonstration results in the following:
episode_lengths: 1000.0
forward_velocity: 5.67
uprightness: 0.89
heading_alignment: 0.83
lateral_velocity: 2.31
We trained reward and policy via inverse reinforcement learning using the provided feature function code with the
↪  demonstration.
We tracked the feature values as well as episode lengths.
The mean values of the last 100 steps from the learned policy are:
forward_velocity: 5.04
uprightness: 0.65
heading_alignment: 0.92
lateral_velocity: 2.05
IRL reward: 3.54
episode_lengths: 932.42
IRL feature weights: {'forward_velocity': 0.549, 'uprightness': 0.253, 'heading_alignment': -0.079, 'lateral_velocity':
↪  0.119}
Please carefully analyze the feedback and provide a new, improved feature function that can better solve the task. Some
↪  helpful tips for analyzing the feedback:
    (1) If the episode lengths are low, it likely means the policy is unsuccessful
    (2) If the feature counts are significantly different between demo and learned behavior, then this means IRL cannot match
    ↪  this feature with the demo as it is written. You may consider
        (a) Change its scale
        (b) Re-writing the feature: check error in the feature computation (e.g., indexing the observation vector) and be
        ↪  careful about outlier values that may occur in the computation
        (c) Discarding the feature
    (3) If a feature has near-zero weight, the feature may be unimportant. You can consider discarding the feature or
    ↪  rewriting it.
    (4) You may add/remove features as you see appropriate.
Please analyze each existing features in the suggested manner above first, and then write the feature function code.The input
↪  of the feature function is a torch.Tensor named `obs_buf` that is a batched state (shape: [batch, num_obs]).
The output of the feature function should be a dictionary where the keys are the names of the features and the values are the
↪  corresponding feature values for the input state.
You must respect the function signature.
The code output should be formatted as a python code string: "```python ... ```".

Some helpful tips for writing the feature function code:
    (1) You may find it helpful to normalize the features to a fixed range by applying transformations
    (2) The feature code's input variables must be obs_buf: torch.Tensor, which corresponds to the state observation
    ↪  (self.obs_buf) returned by the environment compute_observations() function. Under no circumstance can you introduce
    ↪  new input variables.
    (3) Each output feature should only one a single dimension (shape: [batch]).
    (4) You should think step-by-step: first, think what is important in the task based on the task description and the
    ↪  demonstration and come up with names of the features, then, write code to calculate each feature
    (5) You should be aware that the downstream inverse reinforcement learning only creates reward functions that are linear
    ↪  function of the constructed features; thus, it is important to construct expressive features that humans do care in
    ↪  this task
    (6) Do not use unicode anywhere such as \u03c0 (pi)
```

Hyperparameters	Value
Reward learning rate α	1.0
Approximate MaxEnt-IRL number of iterations m	5
Policy training steps k	500
Number of algorithm iterations	3
Number of code generations per iteration	3 (1 for ShadowHand due to longer compile times)
Policy Neural Network Architecture	Fully-connected [32, 32] with ReLU activation

Table 10. Hyperparameters and their values

D. ELEMENTAL Hyerparameters

We tune hyperparameters via a grid search. We summarize the ELEMENTAL hyperparameters in Table 10. All other hyperparameters follow EUREKA's default setup.