# GRAML: Goal Recognition As Metric Learning

**Matan Shamir**[1] , **Reuth Mirsky**[1,2]
[1]Computer Science Department, Bar-Ilan University, Israel
[2]Computer Science Department, Tufts University, MA, USA
matan.shamir@live.biu.ac.il, reuth.mirsky@tufts.edu

## Abstract

Goal Recognition (GR) is the problem of recognizing an agent's objectives based on observed actions. Recent data-driven approaches for GR alleviate the need for costly, manually crafted domain models. However, these approaches can only reason about a pre-defined set of goals, and time-consuming training is needed for new emerging goals. To keep this model-learning automated while enabling quick adaptation to new goals, this paper introduces GRAML: Goal Recognition As Metric Learning. GRAML frames GR as a deep metric learning problem, using a Siamese network composed of recurrent units to learn an embedding space where traces leading to the same goal are close, and those leading to different goals are distant. This metric is particularly effective for adapting to new goals, even when only a single example trace is available per goal. Evaluated on a versatile set of environments, GRAML shows speed, flexibility, and runtime improvements over the state-of-the-art GR while maintaining accurate recognition.

## 1 Introduction

Goal Recognition (GR) involves deducing the objective of an agent based on observed actions. It plays a crucial role in various settings, including Human-Robot Interaction [Massardi *et al.*, 2020; Shvo *et al.*, 2022] and Multi-Agent Systems [Avrahami-Zilberbrand and Kaminka, 2005; Freedman and Zilberstein, 2017; Su *et al.*, 2023]. GR problem formulations typically assume a fixed set of goals [Asai *et al.*, 2022; Mirsky *et al.*, 2021; Ramirez and Geffner, 2009; Amado *et al.*, 2023; Ramırez and Geffner, 2011]. However, consider a service robot recognizing which dish a person is making out of a recipe book with a finite set of recipes. If the robot finds that a recipe was added to the book, it will need to form a new GR problem containing the new recipe and adjust its internal state, a potentially costly process, as traditional GR methods assume a pre-defined set of goals. Adapting to the new goal and performing GR is part of a single problem named Online Dynamic Goal Recognition (ODGR). In ODGR, similar, consequent GR problems share identical or similar domain descriptions but vary in their set of goals [Shamir *et al.*, 2024; Elhadad and Mirsky, 2025]. The paper introduces a new ODGR framework called *Goal Recognition as Metric Learning (GRAML)*, which aims for a one-shot solution when a new set of goals emerges, supporting both discrete and continuous domains. In line with other learning-based GR approaches, GRAML involves a distinct *learning phase* where the domain theory is constructed and an *inference phase* where observations are given. In addition to these phases, GRAML addresses the potential shift in the set of potential goals between GR problems. It does so by incorporating an *adaptation phase*, during which the framework adjusts its internal state to align with the newly emerged goals. GRAML tackles the ODGR problem from a novel perspective. It learns a **metric** for comparing observation sequences by training a Siamese network with a Recurrent Neural Network (RNN) backbone, similar to prior work on time-series similarity [Mueller and Thyagarajan, 2016]. The network is trained to produce embeddings in which sequences leading to the same goal are mapped closer together, while those corresponding to different goals are mapped farther apart. When presented with new goals, GRAML can embed sequences associated with those goals, whether they are provided by a domain expert or self-generated, without retraining. Then, upon receiving an input sequence, its embedding is compared only to the embeddings of sequences for the currently active goals, and the closest goal is returned.

This work introduces several enhancements beyond existing GR: First, GRAML offers a novel and robust model-free, self-supervised approach across continuous and discrete environments. It is aimed to improve recognition speed and applicability in complex domains, with little to no decrease in accuracy. Second, the proposed framework addresses challenges unique to ODGR, including environments with changing goals and the recognition of suboptimal behavior, both aspects often overlooked by prior GR approaches. Second, the proposed framework addresses the unique often-overlooked challenge of dynamic goals in ODGR and the general challenge of recognizing suboptimal behavior in GR. Third, this study presents a set of ODGR benchmark environments to test its components. These environments were designed to represent as realistic and applicative scenarios as possible by generating suboptimal and diverse goal-directed observations using a variety of agents.

## 2 Preliminaries

We overview the definitions required for GR and provide background on Metric Learning and LSTM networks.

### 2.1 Goal Recognition (GR)

A GR problem consists of an *actor* and an *observer*, and it is articulated through the perspective of the observer to infer the goal of the actor. We use a common definition for goal recognition [Ramirez and Geffner, 2009; Amado *et al.*, 2023]:

**Definition 1.** *A **Goal Recognition problem** is a tuple $\langle T, G, O \rangle$, such that $T = \langle S, A \rangle$ is a domain theory where $S$ is the state space and $A$ the action space of the observed actor, $G$ is a set of goals, and $O$ is a sequence of observations, which are state-action tuples. The output of a GR problem is a goal $g \in G$ that best explains $O$.*

Notice this definition is broad to enable diverse solution approaches. While GR is often defined with $G \subseteq S$, this work makes no such assumption. Goals can belong to any computable set, such as $S$, its power set, or labels. Additionally, the actor is assumed to pursue a single goal, inferred from observed state-action pair sequences, while some methods offer flexibility by accommodating observations of either actions or states alone and using metrics to assess multiple possible goals probabilistically.

In this work, a Domain Theory $T$ is modeled via an MDP:

**Definition 2.** *A Markov Decision Process $M$ is a tuple $\langle S, A, T, r, \gamma, \rho_0 \rangle$, where $S, A, \gamma$ and $\rho_0$ denote the state space, action space, discount factor, and distribution of initial states, respectively. $T : S \times A \times S \to [0, 1]$ is the transition function, and $r : S \times A \to \mathbb{R}$ is the reward function.*

In Reinforcement Learning (RL), the agent learns a stochastic **policy** $\pi : S \times A \to [0, 1]$, that is aimed at maximizing the expected discounted cumulative return

$$\mathbb{E}_{a_t \sim \pi(\cdot|s_t),\, s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} \left[ \sum_t \gamma^t r(s_t, a_t) \right]$$

This work focuses on infinite-horizon MDPs with a discount factor $0 < \gamma < 1$, where the state and action spaces can be either discrete or continuous. The RL tasks considered in this work are episodic, terminating upon task completion or after a maximum number of steps is reached. Additionally, this work utilizes Goal-conditioned RL (GCRL) [Liu *et al.*, 2022], an approach where an agent is trained towards multiple possible goals in an episode. A GCRL agent learns a **goal-conditioned policy** $\pi : G \times S \times A \to [0, 1]$, thus exhibiting different behaviors for different goals.

**Online Dynamic Goal Recognition** Shamir et al. [2024] decompose the GR process into iterated phases with distinct inputs and outputs such that each phase can occur once or more, thus shifting from a GR problem to an Online Dynamic Goal Recognition (ODGR):

**Definition 3.** *An ODGR problem is a tuple $\langle T, \langle G^i, \{O\}^i \rangle_{i \in 1..n} \rangle$, where $T = \langle S, A \rangle$ is a domain theory, $G^i$ is a set of goals, and $\{O\}^i$ represents a set of observations sequences, such that the $j$'th observation sequence given after the arrival of $G^i$ as part of $\{O\}^i$ is denoted as $O^i_j = \langle o^i_{j,1}, o^i_{j,2} \ldots \rangle = \langle \langle s^i_{j,1}, a^i_{j,1} \rangle, \langle s^i_{j,2}, a^i_{j,2} \rangle, \ldots \rangle$.*

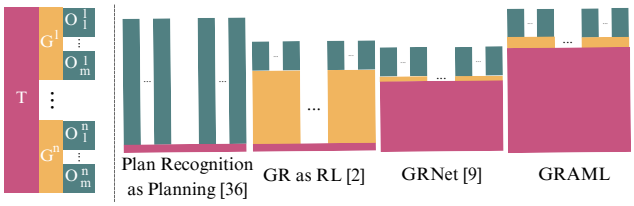


Figure 1: Depiction of ODGR intervals and the inputs at each (left) and a rough approximation of the time spent on each phase by representative GR frameworks (right).

For each $i$, the arrival of $G^{i+1}$ replaces $G^i$ as the active set of goals, where the goal sets may or may not overlap. For all $i, j$, an algorithm for solving ODGR is expected to return a goal $g \in G^i$ that best explains $O^i_j$ upon its arrival. The final output is a set of sets of goals: $G^* = \{\{g^1_1, g^1_2, ...\}, \{g^2_1, g^2_2, ...\}, ..., \{g^n_1, g^n_2, ...\}\}$ where $g^i_j$ is the most likely goal upon receiving $O^i_j$.

For each domain, several *instances* can be constructed at different times according to the active set of potential goals $G^i$ the actor may be pursuing. For each instance, a single recognition *problem* is an observation trace that needs to be explained. We can split an ODGR problem into several time intervals, depending on the reception of each input type, $T$, $G$, and $O$. Within an ODGR problem, one domain exists: as many GR instances as the number of times the goals changed and as many GR problems as the number of observations given as a query to the framework. We name the time intervals according to the reception of each input $T$, $G$, $O$:

**T Domain Learning Time** is the duration from receiving the domain theory $T$ until concluding the domain-specific processing and being prepared to receive $G$.

**G Goals Adaptation Time** is the duration from receiving $G$ until completing the inner-state changes and becoming ready to perform inference.

**O Inference Time** is the time from getting an observation sequence $O$ until the algorithm outputs the goal.

An ODGR framework can receive a new piece of input before completing the processing of the previous one. These definitions remain independent of the time steps assigned to each input. However, for simplicity, in this work, we assume that inputs must *precede* one another. Figure 1 (left) illustrates the connection between the different phases in an ODGR problem. Most of the computation time in symbolic GR approaches occurs at inference time and often depends on the number of goals $|G|$ [Meneguzzi and Fraga Pereira, 2021]. Recent learning-based approaches require less time during the inference phase by pre-processing the domain theory. However, they either necessitate pre-processing the domain for every newly emerging goal [Amado *et al.*, 2022] or result in longer domain learning times [Chiari *et al.*, 2023]. GRAML aims to reduce the goal adaptation time while maintaining a short inference time at the expense of extended domain learning time. A comparison of these approaches is shown in Figure 1 (right).

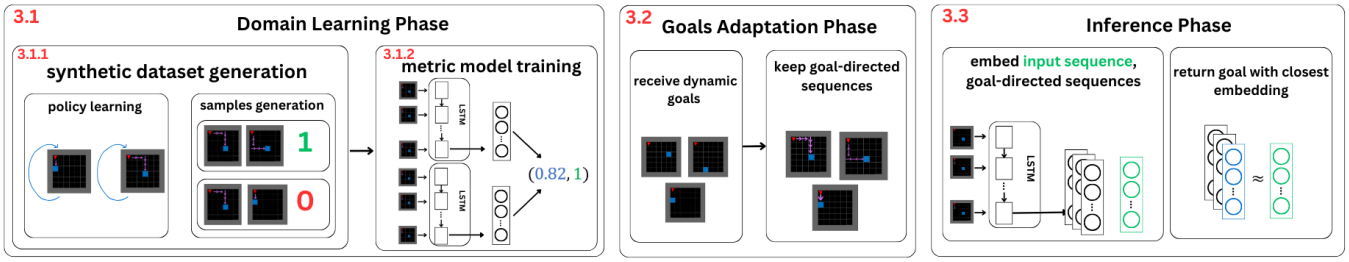


Figure 2: Implementation of GRAML for the phases of an ODGR problem. Numbers in the top-left corner of each box indicate the section discussing each component.

## 2.2 Metric Learning

The primary purpose of metric learning is to learn a new metric where the distances between samples of the same class are expected to be small, and distances between samples of different classes are expected to be larger. This work aims to learn such a metric where sequences of observations from distinct goal-oriented agents are distant from each other but close if the sequences lead to the same goal. Similarity metrics that do not use Deep Neural Networks have limited capabilities due to their inability to represent non-linear structures, such as state-action observation sequences, and thus they cannot be utilized for GRAML [Duan *et al.*, 2018; Duan *et al.*, 2020; Zheng *et al.*, 2019]. One common approach for representing such observation sequences for different possible goals (a multivariate time series) is Long Short-Term Memory (LSTM) for modeling time series similarities [Mueller and Thyagarajan, 2016; Rakthanmanon *et al.*, 2012]. In these models, the last hidden layer represents the input sequence, and the L1 or L2 distance between the representations is the global distance. It has been shown that when using DNNs, a metric learned in a set of classes can be generalized to new classes. This work builds upon these generalization abilities when new goals are introduced.

# 3 Goal Recognition as Metric Learning

GRAML employs a metric-learning model trained on sequences that converge to prototype goals, forming a representation where sequences from agents with distinct goals are widely separated, while those aiming for the same goal are closely clustered. Figure 2 outlines the three phases of GRAML: In the domain learning phase (Section 3.1), GRAML undergoes self-supervised training using agents trained in the GR domain to generate labeled samples. In the goal adaptation phase (Section 3.2), it receives or generates sequences for each newly introduced goal. Finally, in the inference phase (Section 3.3), given a new trace which forms a GR problem, GRAML embeds the goal-directed traces to map each dynamic goal sequence into the embedding space, capitalizing on the generalization abilities of the metric model. Then it identifies the goal whose embedding lies closest to that of the input trace.

We adopt two distinct strategies for GRAML: the *base-goals* approach, referenced to as **BG-GRAML** and the *goal-conditioned RL* approach, referenced to as **GC-GRAML**. As the names imply, GC-GRAML utilizes a single GCRL agent that learns a goal-conditioned policy during domain learning and uses the same policy to generate traces in the goal adaptation phase. In contrast, BG-GRAML involves training separate goal-directed agents and using expert-generated samples or planners in the goal adaptation phase. In the following sections, we outline each framework phase and describe how GRAML operates under both strategies.

## 3.1 Domain Learning Phase

In the domain learning phase, GRAML receives a domain theory $T = \langle S, A \rangle$ and performs a self-supervised training process split into two: dataset generation and model training.

### Dataset Generation

The dataset generation involves training agents on selected goals in the environment, stochastically producing partial sequences from their policies, and labeling sequence pairs as 0 or 1 based on whether they share the same goal.

In BG-GRAML, we first select $m$ base goals $\bar{G} = \langle \bar{g}^1, ..., \bar{g}^m \rangle$ which are ideally chosen such that a metric model trained on sequences directed toward them can generalize well and distinguish sequences aimed at different goals within the potential goal space. These goals are not a part of the ODGR problem formulation but a distinct set chosen by GRAML prior to receiving the first set of possible goals for recognition, $G^0$. We train individual goal-directed RL agents for each $g \in \bar{G}$, obtaining a set of policies $\bar{\Pi} = \langle \bar{\pi}^1, ..., \bar{\pi}^m \rangle$. Then, observation sequences are generated by randomly pairing agents trained towards goals $\bar{g}^i, \bar{g}^j \in \bar{G}$ and have them produce traces $O_i, O_j$ that end at the assigned goal, creating a labeled sample $\langle O_i, O_j, y \rangle$. If $i$ equals $j$, then $y$ equals 1. Otherwise, $y$ equals 0, since these traces lead to separate goals. For more robust training, parts of the sequences are randomly removed, creating both consecutive and non-consecutive traces at varying lengths to associate all kinds of traces that might be encountered in the inference phase with the same goal. Section 4 details the generation of these input variations as used in our empirical work.

GC-GRAML follows a similar process, but relies on a single GCRL agent trained over the set $\bar{G}$. Sample generation is likewise performed through this agent, using a single goal-conditioned policy to produce the traces. Unlike in BG-GRAML, the GCRL agent and the associated policy $\bar{\pi}^*$ are retained for use during the goal adaptation phase.

**Model Training**

Given an existing dataset of $n$ labeled samples $D = \{\langle O^1, O^2, y\rangle_1, ..., \langle O^1, O^2, y\rangle_n\}$, the model training involves training a metric model $M$, structured as a Siamese network and comprised of 2 LSTM branches with shared weights. The input to each LSTM unit has a size of $s + a$, where $s$ and $a$ represent the sizes of the state and action representations in $T$, respectively. For every sample $\langle O^1, O^2, y\rangle_i$, the model processes $O^1, O^2$ through branches 1 and 2 to obtain 2 embeddings $v^1, v^2 \in \mathbb{R}^k$. The model then calculates an exponential transformation on the Manhattan distance metric between $v^1, v^2$, to produce a similarity score:

$$\hat{y} = \text{similarity}(v^1, v^2) = e^{-\sum_{i=1}^{k} |v^1[i] - v^2[i]|} \quad (1)$$

where brackets denote specific elements of the vector. The closer the corresponding embedding elements are in value, the closer the similarity score is to 1. The model is penalized using the binary cross entropy loss function:

$$L = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right) \quad (2)$$

### 3.2 Goal Adaptation Phase

The goal adaptation phase starts upon receiving the $i$-th set of active goals, $G^i = \{g_1, ..., g_{|G^i|}\}$. The main challenge GRAML faces is obtaining a set of libraries of goal-directed sequences $O^* = \langle \{O^1, ..., O^s\}_{g_1}, ..., \{O^1, ..., O^s\}_{g_{|G|}} \rangle$, where $s$ denotes the library size and is a hyperparameter that can be tuned. If a domain expert provides $O^*$, the adaptation is immediate (disregarding the manual effort involved). Otherwise, GRAML supports two alternative strategies: In BG-GRAML, an off-the-shelf MDP planner such as Monte Carlo Tree Search (MCTS), is used to produce $O^*$, generating one or more goal-directed sequences for each $g \in G^i$. In GC-GRAML, the goal-conditioned policy $\bar{\pi}^*$ generates the sequences using stochastic action predictions to encourage diversity, relying on its generalization capacity.

### 3.3 Inference Phase

This phase starts with an active set of goals $G^i$, and a single observation $O^i_j = \langle o^i_{j,1}, o^i_{j,2} \ldots \rangle$. The objective of this phase is to return the goal $g \in G^i$ to which $O^i_j$ most likely leads. Using the LSTM from the learning phase and the sequences for the active set of goals, $O^*$, the goal whose embedding is closest to the embedding of the input sequence $O$ is returned.

Using $O^*$, GRAML produces embeddings libraries $\langle \{v^1, ..., v^s\}_{g_1}, ..., \{v^1, ..., v^s\}_{g_{|G^i|}} \rangle$ corresponding to every goal-directed library $\{O^1, ..., O^s\} \in O^*$ . These embeddings could potentially be generated during the goal adaptation phase to decrease online inference runtime. However, our experiments showed improved accuracy when the full sequences $O^*$ were truncated to the length of $O^i_j$ before computing embeddings. It then generates an embedding $v$ for $O^i_j$ and returns the goal whose sequence embeddings yield the highest average similarity with $v$:

$$g = \arg\max_{g \in G^i} \frac{1}{s} \sum_{j=1}^{s} \text{similarity}(v^j_g, v) \quad (3)$$

Where $v^j_g$ denotes embedding number $j$ in the library of a goal $g$. This averaging method improves recognition accuracy by accounting for different trajectories for the same goal. However, it introduces a trade-off: as the value of $s$ increases, the runtime also increases. Constructing larger libraries is likely to improve GRAML's accuracy, but as demonstrated in the empirical evaluation, the library generation time grows linearly with $s$ for both BG-GRAML and GC-GRAML.

## 4 Evaluation

We implemented GRAML using Python, employing PyTorch for the LSTM-based metric learning components.[1] All experiments were conducted on a commodity Intel i-7 pro.

**Variables** We vary our experiments across the following dimensions to test GRAML under different settings:

- **Discrete or Continuous state/action environments**. In our experiments, we have two environments with discrete state and action spaces (Minigrid-SimpleCrossing and Minigrid LavaCrossing) and four environments with continuous ones (Parking, PointMaze-Obstacle, PointMaze-FourRooms and Panda-Gym Reach).
- **Consecutive and non-consecutive partial observations**. A *consecutive* sequence is a partial sequence of observations from a starting state to the goal, where the missing parts are from a certain observation until the goal. A *non-consecutive* sequence is a partial sequence of observations from a starting state to the goal, where the missing parts are randomly chosen from the sequence. For each environment, we tested observation sequences that are $30\%, 50\%, 70\%$, and $100\%$ of the full sequence, both consecutively and non-consecutively.
- **Suboptimal Observations**. At inference, a stochastic policy generates sub-optimal observation sequences.
- **Initial States**. Initial steps are chosen randomly to encourage diverse plans.

For each ODGR problem generated, there is a single goal adaptation phase followed by several inference phases where each goal in the active set is considered the true goal with every observability configuration. This process yields 5 (average number of active goals $|G^0|$) $\times$ 8 (observability $\%$ and consecutive/non-consecutive) $= 40$ unique inference phases. Since there are five separate ODGR problems for every environment, there are five goal adaptation phases and 200 GR problems for every one of the six environments.

**Data Sets**

Building on the GCRL survey and the benchmark environments suggested at ApexRL [2], we form a collection of GR problems from several sets of environments that adhere to the Gymnasium API [3], with detailed descriptions of each in Appendix **??**. We consider two custom Minigrid environments from the minigrid package [Chevalier-Boisvert *et al.*, 2023], two custom PointMaze environments from the

[1] https://github.com/MatanShamir1/Grlib

[2] https://github.com/apexrl/

[3] https://gymnasium.farama.org/

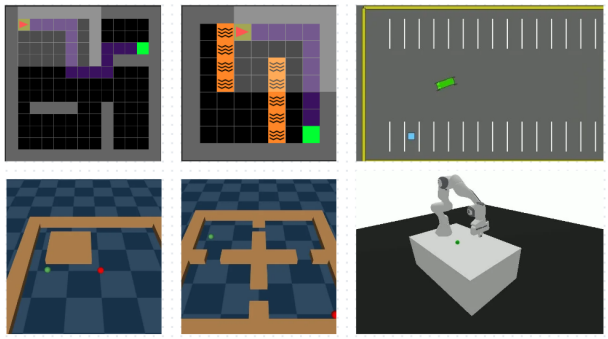

Figure 3: The evaluation environments from top-left to bottom-right: Minigrid-SimpleCrossing, Minigrid-LavaCrossing, Parking, PointMaze-Obstacle, PointMaze-FourRooms and Panda-Gym.

Gymnasium-Robotics package [Fu *et al.*, 2020], the Parking environment from the highway-env package [Leurent, 2018], and the Reach environment from PandaGym [Gallouédec *et al.*, 2021]. Figure 3 illustrates these environments.

**Algorithms** The primary evaluation focuses on the BG-GRAML and GC-GRAML implementations. In the goal adaptation phase of BG-GRAML, a custom implementation of MCTS was used to generate plans in discrete environments. For continuous environments, expert-provided examples were used exclusively, with the RL policy used for generating these examples differing from the one used during input generation in the inference phase.

We compare BG-GRAML and GC-GRAML to the GRAQL [Amado *et al.*, 2022] algorithm in discrete domains and DRACO [Nageris *et al.*, 2024] in continuous domains. These baseline methods require training a dedicated goal-conditioned agent for each candidate goal. The resulting policies—while dependent on the underlying algorithm—are generally assumed to be near-optimal. When the observed behavior at inference time is close to optimal, these methods can achieve high accuracy, as they do not depend on the generalization capacity of a learned metric model. However, this advantage comes at the expense of significant computational overhead, which may render them impractical for real-time or large-scale deployments. Another variation, named GC-DRACO, learns a goal-conditioned policy once during the learning phase. It is used in our experiments to compare against GC-GRAML. Deep RL algorithms from Stable Baselines3 [Raffin *et al.*, 2021] are used to train agents in continuous environments, while tabular Q-Learning, as originally applied in GRAQL, is used in discrete settings across all phases of the framework. Experimental diversity is further promoted by varying the properties of the environments and using different RL algorithms across phases. The algorithms used include PPO, SAC, and TD3, rotated to ensure that sequence generation relies on distinct policies.

Next, we showcase how learned embeddings enable measuring goal distances (Figure 4), continue with the trade-off between BG-GRAML and GC-GRAML (Figure 5), and then present GRAML performance under varying observability (Table 1) and discuss runtime. Lastly, we examine how recognition accuracy is affected by the size of the base goal set $\bar{G}$ and the active goal set $G^i$ (Figure 6).

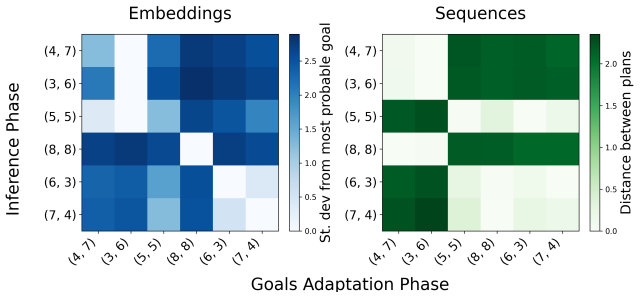


Figure 4: Confusion matrices for plan similarity and recognition confidence in the inference phase for GRAML across multiple tasks in the PointMaze environment where the goals differ from the set of base goals from the domain learning phase.

## 4.1 Embedding Similarity

Before evaluating the performance of GRAML for GR, we verify that the learned model embeds sequences leading to the same goal closer together. This is especially important when execution can vary, and different distinct sequences can lead to the same goal. Figure 4 shows that BG-GRAML is able to create an embedding in which different sequences to the same goal get closer embeddings. The right matrix shows similarity scores between sequences, where each cell $(i, j)$ indicates the element-wise numerical difference between the input sequence reaching goal $i$ from the inference phase and the sequence reaching goal $j$ from the latest goal adaptation phase. The left matrix follows the same structure but reflects differences between the corresponding embeddings. For example, despite the lack of similarity between the sequences leading to the goal (8,8) in the goal adaptation phase (Figure 4 (right)), GRAML (Figure 4 (left)) learns an embedding in which these sequences are close to one another.

## 4.2 GRAML Strategies

Next, we compare the accuracy of BG-GRAML and GC-GRAML given sequences of changing observability when we aim to adapt quickly to new goals while minimizing the dependence on a domain expert. To do so, both BG-GRAML and GC-GRAML are fed with a single sequence per goal in $(s = 1)$. Each single-goal agent was trained for $300,000$ timesteps, and the goal-conditioned agent was trained for $1$ million timesteps. GC-GRAML is trained using sequences generated by a single goal-conditioned policy $\pi^*$, which was trained on a wide range of goals in $\bar{G}$. This allows the dataset to include more diverse observation sequences, as the same policy produces behavior across many different goals. In contrast, BG-GRAML relies on five separate goal-directed policies, each trained on a single goal, limiting the diversity of the training data available to its LSTM. Since the domain learning time of GC-GRAML is not affected by the number of base goals, $\bar{G}$ was set to 20, while BG-GRAML used only 5. As shown in Figure 5, GC-GRAML benefits from this sequence diversity and consistently generalizes better, recognizing unseen observation traces more effectively than BG-GRAML.

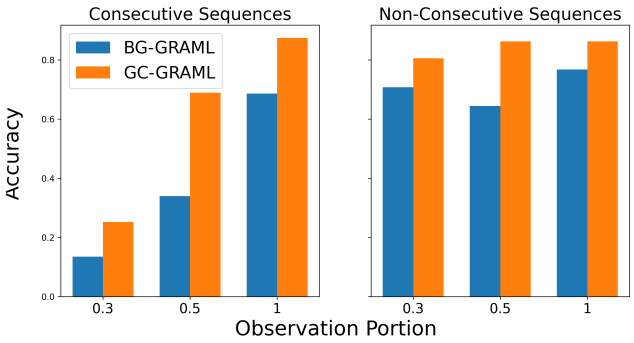


Figure 5: BG-GRAML and GC-GRAML accuracy in Parking.

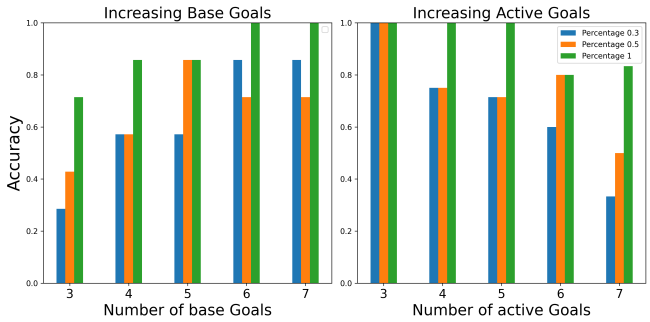


Figure 6: Recognition accuracy in BG-GRAML increases with more base goals and decreases with more newly introduced goals.

### 4.3 Inference Performance

Given a GR problem, the performance of GRAML is compared with that of the selected GR as RL algorithms. In each scenario presented in Section 4, a set of base goals $\bar{G}$ was chosen, along with sets of 3-9 active goals $G^0$ for the sole goal adaptation phase in the experiment. Every $g \in G^0$ was once chosen as the true goal, yielding a set of 200 GR problems per scenario on average, depending on the size of $G^0$.

**GR accuracy** Table 1 summarizes the execution results in terms of accuracy. Notice that due to the ambiguity of traces (especially with consecutive observations and low observability), even a perfect recognizer might not be able to reach perfect recognition. The performance of the GRAML variants consistently demonstrated strong results, comparing to or surpassing the GR as RL versions in many scenarios despite its decreased adaptation time. Notably, GRAML benefits from increased observability levels compared to algorithms that were trained directly on the current goal set $G^0$.

**GR runtime** Recall that GRAML aims to optimize goal adaptation and inference times. We compare the computation time during goal adaptation in GRAML to the goal learning in GRAQL and DRACO, as they need to learn these goals for every GR instance. We compared all supported domains, averaging results over 200 instances and selecting the best-performing variant from each algorithm. Recall that G and O denote goal adaptation and inference times, respectively. Algorithms exceeding two hours for goal adaptation were deemed to have timed out. In the Minigrid domain, GRAQL timed out during G, with O averaging 0.04 seconds, while BG-GRAML incurred $G = 0$ due to the absence of computations and achieved an average O of 0.03 seconds. In the PointMaze domain, DRACO timed out during G and had an average O of 2.56 seconds, whereas BG-GRAML incurred $G = 0$ and achieved an average O of 1.47 seconds. In the Parking domain, GC-DRACO timed out during G and averaged $O = 2.49$ seconds, while GC-GRAML required 36 seconds for G and averaged $O = 2.17$ seconds. In the Panda domain, GC-DRACO timed out during G and averaged $O = 2.24$ seconds, while GC-GRAML required 87 seconds for G and 5.29 seconds for O. BG-GRAML achieves $G = 0$ by leveraging expert sequences. For reference, in the Parking domain with $|G_i| = 5$, DRACO required ten hours for goal adaptation.

Additional times are the acquisition of the base goals sequences during the self-supervised learning and potential plan time in the goals adaptation of BG-GRAML. Running BG-GRAML with MCTS in discrete environments for planning instead of receiving a sample from an expert yields a planning time of 75 seconds per goal on average. Considering the time it takes to train agents towards every active goal as the timeout, creating a library of sequences per goal in a problem with five goals on average takes 18 minutes, about 4 times faster than training agents for every goal in the active set.

### 4.4 Goal Set Size

Lastly, we report the influence of the goal set size ($|\bar{G}|$) for the learning phase of GRAML and the active goal set size ($|G_i|$) on the performance of BG-GRAML performance in the Parking environment. Figure 6 (left) shows that increasing the number of base goals leads to improved accuracy for the algorithm during recognition. This improvement is due to the wider variety of goal sequences to train on, as well as the sheer number of diverse goals, which, in turn, leads to higher generalization abilities. Figure 6 (right) shows an opposite trend as the number of active goals increases. This is unsurprising; as more goals are introduced, the possible similarity between sequences leading to them increases, making the goals harder to disambiguate.

There are a couple of instances where, on average, lower observability leads to better recognition than higher observability. We attribute this behavior to the random way the sequences were generated for each experiment (recall that the policy used for this generation process is stochastic). Still, the results show a mostly consistent behavior.

## 5 Related Work

This section provides an overview of recent work, emphasizing the strengths and weaknesses of each methodology under different assumptions regarding input timing and the potential changes in goals presented by ODGR problems. We split existing work into Model-Based GR (MBGR) and Model-Free GR (MFGR) [Geffner, 2018]:

**Model-Based GR (MBGR)** In MBGR, the recognition process entails utilizing a pre-defined and clearly stated model that encompasses the characteristics of an environment along with the actions that can be executed within that environment [Meneguzzi and Fraga Pereira, 2021; Mirsky *et al.*,

| Env. | Consecutive | | | | | | Non-consecutive | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 30% | | 50% | | 70% | | 30% | | 50% | | 70% | | 100% | |
| | BG-G | GRAQL | BG-G | GRAQL | BG-G | GRAQL | BG-G | GRAQL | BG-G | GRAQL | BG-G | GRAQL | BG-G | GRAQL |
| Minigrid | 0.35(0.18) | **0.40**(0.19) | 0.44(0.20) | **0.51**(0.22) | 0.55(0.19) | **0.58**(0.21) | 0.63(0.24) | **0.66**(0.22) | 0.77(0.17) | **0.81**(0.13) | **0.83**(0.17) | 0.84(0.20) | **0.82**(0.18) | 0.90(0.15) |
| | BG-G | DRACO | BG-G | DRACO | BG-G | DRACO | BG-G | DRACO | BG-G | DRACO | BG-G | DRACO | BG-G | DRACO |
| PointMaze | **0.51**(0.20) | 0.50(0.23) | 0.63(0.21) | **0.71**(0.20) | **0.83**(0.17) | 0.82(0.18) | 0.75(0.30) | **0.81**(0.21) | 0.79(0.29) | **0.84**(0.20) | 0.78(0.27) | **0.84**(0.16) | **0.80**(0.24) | 0.84(0.16) |
| | GC-G | GC-D | GC-G | GC-D | GC-G | GC-D | GC-G | GC-D | GC-G | GC-D | GC-G | GC-D | GC-G | GC-D |
| Parking | **0.43**(0.26) | 0.42(0.22) | 0.45(0.24) | **0.49**(0.20) | **0.76**(0.20) | 0.54(0.19) | **0.84**(0.16) | 0.55(0.19) | **0.88**(0.19) | 0.56(0.19) | **0.88**(0.17) | 0.66(0.18) | **0.89**(0.16) | 0.71(0.18) |
| Panda | **0.45**(0.23) | 0.35(0.16) | **0.74**(0.17) | 0.56(0.20) | **0.82**(0.16) | 0.79(0.03) | **0.83**(0.20) | 0.82(0.20) | **0.91**(0.15) | 0.90(0.11) | 0.92(0.13) | **0.94**(0.10) | 0.92(0.14) | **0.95**(0.08) |

Table 1: Accuracy for Different Domains, Sequence Lengths, and Algorithms, averaged over 200 instances (with std) for configurations varying by observation type and observability %. Minigrid is further averaged on Minigrid-SimpleCrossing and Minigrid-LavaCrossing, and PointMaze on PointMaze-Obstacle and PointMaze-FourRooms. BG-G is BG-GRAML, GC-G is GC-GRAML, and GC-D is GC-DRACO.

2021]. Traditional MBGR usually exploits planning and parsing techniques [Baker *et al.*, 2009; Geib and Goldman, 2009; Ramirez and Geffner, 2009; Ramırez and Geffner, 2011]. Although the domain learning for these algorithms is nonexistent in terms of compute, it requires a domain expert to manually construct these models.

**Model-Free GR (MFGR)** In MFGR, the recognizer lacks access to the environment model that describes its properties and dynamics [Geffner, 2018]. Some approaches learn the model first then apply MBGR [Asai *et al.*, 2022; Geib and Kantharaju, 2018; Su *et al.*, 2023], while others perform end-to-end model-free GR without explicit model learning [Amado *et al.*, 2022; Borrajo *et al.*, 2020; Chiari *et al.*, 2023; Fang *et al.*, 2023; Min *et al.*, 2014]. These methods differ in the representation of the environment and in the assumptions about existence and timing of the training inputs. GRAML belongs to MFGR but stands out by not requiring solved task datasets and by addressing goal generalization, which most methods overlook.

The approach most similar to GRAML in underlying model is GR as RL [Amado *et al.*, 2022], which also employs an MDP representation. It trains a separate policy for each goal, resulting in high computational cost and inability to directly handle ODGR. GRAQL, a discrete domain algorithm derived from GR as RL, has been extended to continuous domains with deep RL by DRACO [Nageris *et al.*, 2024], but it inherits the limitations of GRAQL. A variant of DRACO using GCRL offers fast adaptation to new goals but still depends on the narrow notion of optimal behavior. Most approaches mentioned rely on a single plan or policy, which may misclassify observations that diverse plans would recognize, with few exceptions addressing this implicitly [Chiari *et al.*, 2023] or explicitly [Sohrabi *et al.*, 2016].

While most methods do not explicitly address ODGR, Chiari et al. [Chiari *et al.*, 2023] propose GRNet, a framework that generalizes to any goal set without further learning, providing a model-free solution for dynamic environments. GRNet uses RNNs to process observation sequences and requires a dedicated learning phase on a dataset of solved GR problems with domain theory known beforehand, leading to long domain learning times. Despite its promise, GRNet's reliance on fluent enumeration remains critical, and extending it to continuous domains requires further study. In contrast, GRAML handles continuous domains by having the LSTM output operate in embedding space rather than state space.

# 6 Conclusions and Future Work

This paper introduces GRAML, a new algorithm for GR and ODGR using Metric Learning. GRAML was designed to handle both discrete and continuous domains, prioritizing the **speed** of adaptation to emerging goals and the promptness of inference upon receiving a sequence of observations $O$. Aiming at robust applicability, GRAML also targets accurate recognition of **suboptimal** and **highly-dissimilar** observation sequences that lead to a certain goal. We achieve diversity across (1) input sequences to the same goals, used to distinguish the acting agent and the observing agents' underlying policies, and (2) inference phase and goal adaptation phase sequences, used to generate more challenging and nontrivial recognition of new goals for GRAML.

This paper further analyzes how different variables of an environment, such as the domain characteristics, the variability in sequence generation, and the observation sequence format, can influence learning and inference performance for ODGR algorithms and GRAML in particular. These domain-agnostic attributes are crucial when evaluating an ODGR framework, and researchers are welcome to build upon this methodology when evaluating new ODGR approaches.

We propose two GRAML variants. BG-GRAML may be time-consuming due to reliance on expert samples or planners but does not depend on the generalization of a GCRL agent. GC-GRAML, when feasible, requires no extra information during goal adaptation and benefits from a more diverse training dataset. Its superior performance over BG-GRAML is mainly due to handling more base goals during training. Since GC-GRAML uses a general-purpose policy instead of goal-specific ones, it is not expected to outperform BG-GRAML when both use the same number of goals.

The selection of base goals for learning is critical but was not addressed here. Without adequate coverage of the goal space during domain learning, the model may fail to recognize goals in underrepresented regions. Training a GCRL agent on a wide range of goals proved effective, but more sophisticated goal selection methods could improve coverage and sample efficiency. Optimizations during goal adaptation, such as outlier removal or clustering over the sequence libraries, could also be effective.

Future work could extend GRAML to sequences from multiple MDPs with varying dynamics, addressing a more general GR paradigm where both goals and environment dynamics may change.

## References


[Amado *et al.*, 2022] Leonardo Amado, Reuth Mirsky, and Felipe Meneguzzi. Goal recognition as reinforcement learning. In *The AAAI Conference on Artificial Intelligence*, volume 36, pages 9644–9651, 2022.

[Amado *et al.*, 2023] Leonardo Amado, Ramon Fraga Pereira, and Felipe Meneguzzi. Robust neuro-symbolic goal and plan recognition. *The AAAI Conference on Artificial Intelligence*, 37(10):11937–11944, Jun. 2023.

[Asai *et al.*, 2022] Masataro Asai, Hiroshi Kajino, Alex Fukunaga, and Christian Muise. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74:1599–1686, 2022.

[Avrahami-Zilberbrand and Kaminka, 2005] Dorit Avrahami-Zilberbrand and Gal A Kaminka. Fast and complete symbolic plan recognition. In *IJCAI*, pages 653–658, 2005.

[Baker *et al.*, 2009] Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[Borrajo *et al.*, 2020] Daniel Borrajo, Sriram Gopalakrishnan, and Vamsi K Potluru. Goal recognition via model-based and model-free techniques. *FinPlan 2020*, page 10, 2020.

[Chevalier-Boisvert *et al.*, 2023] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

[Chiari *et al.*, 2023] Mattia Chiari, Alfonso Emilio Gerevini, Francesco Percassi, Luca Putelli, Ivan Serina, and Matteo Olivato. Goal recognition as a deep learning task: The grnet approach. In *The International Conference on Automated Planning and Scheduling*, volume 33, pages 560–568, 2023.

[Duan *et al.*, 2018] Yueqi Duan, Jiwen Lu, Jianjiang Feng, and Jie Zhou. Deep localized metric learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2644–2656, 2018.

[Duan *et al.*, 2020] Yueqi Duan, Jiwen Lu, Wenzhao Zheng, and Jie Zhou. Deep adversarial metric learning. *IEEE Transactions on Image Processing*, 29:2037–2051, 2020.

[Elhadad and Mirsky, 2025] Osher Elhadad and Reuth Mirsky. General dynamic goal recognition. *arXiv preprint arXiv:2505.09737*, 2025.

[Fang *et al.*, 2023] Zihao Fang, Dejun Chen, Yunxiu Zeng, Tao Wang, and Kai Xu. Real-time online goal recognition in continuous domains via deep reinforcement learning. *Entropy*, 25(10):1415, 2023.

[Freedman and Zilberstein, 2017] Richard Freedman and Shlomo Zilberstein. Integration of planning with recognition for responsive interaction using classical planners. In *The AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[Fu *et al.*, 2020] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[Gallouédec *et al.*, 2021] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[Geffner, 2018] Hector Geffner. Model-free, model-based, and general intelligence. In *International Joint Conference on Artificial Intelligence*, 2018.

[Geib and Goldman, 2009] Christopher W Geib and Robert P Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173(11):1101–1132, 2009.

[Geib and Kantharaju, 2018] Christopher Geib and Pavan Kantharaju. Learning combinatory categorial grammars for plan recognition. In *The AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Leurent, 2018] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[Liu *et al.*, 2022] Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

[Massardi *et al.*, 2020] Jean Massardi, Mathieu Gravel, and Éric Beaudry. Parc: A plan and activity recognition component for assistive robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3025–3031. IEEE, 2020.

[Meneguzzi and Fraga Pereira, 2021] Felipe Meneguzzi and Ramon Fraga Pereira. A survey on goal recognition as planning. In *International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4524–4532, 8 2021.

[Min *et al.*, 2014] Wookhee Min, Eun Ha, Jonathan Rowe, Bradford Mott, and James Lester. Deep learning-based goal recognition in open-ended digital games. In *The AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, pages 37–43, 2014.

[Mirsky *et al.*, 2021] Reuth Mirsky, Sarah Keren, and Christopher Geib. *Introduction to Symbolic Plan and Goal Recognition*, pages 1–120. Number 1 in Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 1 edition, January 2021.

[Mueller and Thyagarajan, 2016] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. *The AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.

[Nageris *et al.*, 2024] Ben Nageris, Felipe Meneguzzi, and Reuth Mirsky. Goal recognition using actor-critic optimization. *arXiv preprint arXiv:2501.01463*, 2024.

[Raffin *et al.*, 2021] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah

Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[Rakthanmanon *et al.*, 2012] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Al Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. *KDD : proceedings. International Conference on Knowledge Discovery & Data Mining*, 2012:262 – 270, 2012.

[Ramirez and Geffner, 2009] M. Ramirez and H. Geffner. Plan recognition as planning. In *International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[Ramırez and Geffner, 2011] Miquel Ramırez and Hector Geffner. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *IJCAI*, pages 2009–2014. IJCAI/AAAI, 2011.

[Shamir *et al.*, 2024] Matan Shamir, Osher Elhadad, and Reuth Mirsky. Odgr: Online dynamic goal recognition. In *The Workshops of The RLC Conference*, 2024.

[Shvo *et al.*, 2022] Maayan Shvo, Ruthrash Hari, Ziggy O'Reilly, Sophia Abolore, Sze-Yuh Nina Wang, and Sheila A McIlraith. Proactive robotic assistance via theory of mind. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 9148–9155. IEEE, 2022.

[Sohrabi *et al.*, 2016] Shirin Sohrabi, Anton V Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *IJCAI*, pages 3258–3264. New York, NY, 2016.

[Su *et al.*, 2023] Zihang Su, Artem Polyvyanyy, Nir Lipovetzky, Sebastian Sardiña, and Nick van Beest. Fast and accurate data-driven goal recognition using process mining techniques. *Artificial Intelligence*, 323:103973, 2023.

[Zheng *et al.*, 2019] Wenzhao Zheng, Zhaodong Chen, Jiwen Lu, and Jie Zhou. Hardness-aware deep metric learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 72–81, 2019.