UNIVERSITÀ DEGLI STUDI DI BRESCIA

DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

SSD: ING-INF/05

CICLO XXXVI

DEEP LEARNING APPROACHES TO GOAL RECOGNITION

Dottorando:
Mattia Chiari
Matricola 88898

Supervisore: Prof. Alfonso E. Gerevini
Co-Supervisore: Prof. Ivan Serina

Abstract

Recognising the goal of an agent from a trace of observations is an important task with many applications. In the literature, many approaches to goal recognition (GR) rely on the application of automated planning techniques which requires a model of the domain actions and of the initial domain state (written, e.g., in PDDL). We study three alternative approaches (GRNet, Fast and Slow Goal Recognition and a BERT-based approach) where Goal Recognition is formulated as a classification task addressed by machine learning. All these approaches are primarily aimed at solving GR instances in a given domain, which is specified by a set of propositions and a set of action names. In GRNet, the goal classification instances in the domain are solved by an LSTM network. The only information required as input of the trained network is a trace of action names, each one indicating just the name of an observed action. A run of the LSTM processes a trace of observed actions to compute how likely it is that each domain proposition is part of the agent's goal. Fast and Slow Goal Recognition, inspired by the "Thinking Fast and Slow" framework, is a dual-process model which integrates the use of the aforementioned LSTM with the automated planning techniques. This architecture can exploit both the fast, experience-based goal recognition provided by the network, and slow, deliberate analysis provided by the planning techniques. Finally, we study how a BERT model trained on plans is able to understand how a domain works, its actions and how they are related to each other. This model is then fine-tuned in order to classify goal recognition instances. Experimental analyses confirms that the presented architectures achieve good performance in terms of both goal classification accuracy and runtime, often obtaining better results w.r.t. a state-of-the-art GR system over the considered benchmarks.

Abstract

Riconoscere il goal di un agente utilizzando una traccia di osservazioni è un compito importante con diverse applicazioni. In letteratura, molti approcci di goal recognition (GR) si basano sull'applicazione di tecniche di pianificazione automatica che richiedono un modello delle azioni del dominio e dello stato iniziale del dominio (scritto, ad esempio, in PDDL). In questa tesi studiamo tre approcci alternativi (GRNet, Fast and Slow Goal Recognition e un approccio basato su BERT) in cui il goal recognition è formulato come un compito di classificazione affrontato utilizzando il machine learning. Tutti questi approcci mirano principalmente a risolvere istanze di GR in un dato dominio, specificato da un insieme di proposizioni e da un insieme di nomi di azioni. In GRNet, le istanze di classificazione del dominio sono risolte da una rete LSTM. L'unica informazione richiesta come input della rete addestrata è una traccia di nomi di azioni, ognuno dei quali indica solo il nome di un'azione osservata. Un'esecuzione della LSTM elabora una traccia di azioni osservate per calcolare la probabilità che ogni proposizione del dominio faccia parte del goal dell'agente. Fast and Slow Goal Recognition, ispirato al framework "Thinking Fast and Slow", è un modello a doppio processo che integra l'uso delle sopra-citate reti LSTM con le tecniche di pianificazione automatica. Questa architettura può sfruttare sia il riconoscimento veloce dei goal, basato sull'esperienza, fornito dalla rete, sia l'analisi lenta e deliberata fornita dalle tecniche di pianificazione. Infine, studiamo come un modello BERT addestrato sui piani sia in grado di comprendere il funzionamento di un dominio, le sue azioni e le loro relazioni reciproche. Questo modello viene poi sottoposto a fine-tuning per classificare le istanze di goal recognition. Le analisi sperimentali confermano che le architetture presentate raggiungono buone prestazioni sia in termini di accuratezza della classificazione dei goal che di tempo di esecuzione, ottenendo spesso risultati migliori rispetto a un sistema di goal recognition allo stato dell'arte sui benchmark considerati.

Contents

Chapter 1

Introduction

This thesis illustrates the study and work carried out at the Information Engineering department of Università degli Studi di Brescia during my Ph.D. in Information Engineering, curriculum Computer Science / Engineering and Control Systems, cohort XXXVI. My work was done under the supervision of Prof. Alfonso Emilio Gerevini and Prof. Ivan Serina at Università degli Studi di Brescia.

1.1 Topic of the Thesis

Suppose that there is a robot which is moving in a room. The robot can grab either a box or an apple that are both in the room. This very simple example is depicted in Figure 1.1. By solving a goal recognition task, we want to infer, by observing the robot moving, whether it is aiming for the box or for the apple. Goal Recognition, in fact, is the task of recognising the goal that an agent (the robot, in our example) is trying to achieve from observations about the agent's behaviour in the environment [62, 23]. Typically, such observations consist of a trace (sequence) of executed actions in an agent's plan to achieve the goal, or

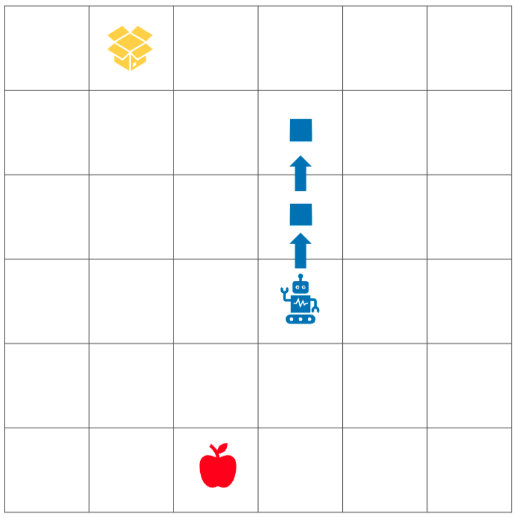


Figure 1.1: A simple goal recognition problem. We are observing a robot moving in a room and we have to recognise whether it is grabbing the yellow box or the red apple.

a trace of world states generated by the agent's actions, while an agent goal is specified by a set of propositions. In this Thesis, we assume that the agent is either unaware of being observed and it's trying to reach the goal or that it is cooperating with the observer; in other words, although the agent may not always behave in a completely rational way, it is never trying to hide his intentions from the observer. Goal recognition has been studied in AI for many years, and it is an important task in several fields, including human-computer interactions [7], computer games [46], network security [48], financial applications [11], and others.

In the literature, several systems to solve goal recognition problems have been proposed [44]. The most effective approaches are based on transforming a plan recognition problem into one or more plan generation problems solved by classical planning algorithms [55, 51, 61]. Intuitively, for each possible goal, these approaches try to complete the action sequence in order for it to reach that goal. Then they design different algorithms to select the more likely sequence and, thus, the more likely goal. In order to perform planning, this approach requires domain knowledge consisting of a model of each agent's action specified as a set of preconditions and effects, and a description of the initial state of the world, in which the agent performs the actions.

Despite the excellent performance, the computational efficiency (runtime) of these approaches largely depends on the planning algorithms, which are often quite slow. We aim to present architectures that make goal recognition faster by learning how to solve this problem in a given domain without any loss in performance. In this thesis, we investigate three alternative approaches in which the goal recognition problem is formulated as a classification task, addressed through deep learning, where each candidate goal (a set of propositions) of the problem can be seen as a value class. To design our architectures, we were inspired by state-of-the-art techniques used in Natural Language Processing (NLP) such as LSTM network and Transformer-based architectures.

The approaches we present in this thesis are: GRNet, Fast and Slow Goal Recognition and a BERT-based approach. GRNet is a new system for goal recognition based on LSTM. A run of the LSTM processes a trace of observed actions to compute how likely it is that each domain proposition is part of the agent's goal. Fast and Slow Goal Recognition is a novel approach to goal recognition inspired by the "Thinking, Fast and Slow" framework, which highlights the distinct cognitive processes involved in decision-making. Fast and Slow Goal Recognition integrates the use of the LSTM network for fast, intuitive recognition and immediate goal identification and a automated planning approach for slow, deliberate analysis and deeper understanding and inference. Finally, we present a BERT-based architecture trained on plans generated through automated planning. Our aim is to study whether this model is able to understand the actions and how they are related in a given domain. We then fine-tune this model to classify goal recognition instances.

1.2 Structure of the Thesis

In this section, we present a brief overview of the structure of the content set forth in this thesis.

In Chapter 2 we provide a brief introduction on Automated Planning. We first introduce the basics of Classical Planning (Section 2.1). Then we give a strict definition of goal and plan recognition (Section 2.2.1), explaining differences and common points between these two close research fields. Finally in Section 2.3.3, we describe the current state-of-the-art for goal and plan recognition.

Chapter 3 addresses some of the state-of-the-art techniques used in Deep-Learning, in particular those applied in the field of Natural Language Processing. In the first section of this chapter we explain the word embedding techniques, that is, how to represent words so that they can be processed by a neural network. Recurrent Neural Networks represent a well-established approach in many NLP applications; their use is shown in Section 3.2. After that, we show both the intuition and the theory behind different variants of the Attention Mechanism. In the last section, we show how the attention mechanism is used in the recent Deep-Learning architectures such as Transformers and BERT.

The original contributions of this thesis are reported in Chapter 4, 5 and 6 which report respectively the description of GRNet, Fast and Slow Goal Recognition and a BERT-based approach for goal recognition.

First, in Chapter 4 we introduce a simple goal recognition instance that will be used throughout the chapter to show how the proposed architectures work.

Then, we describe the architecture of GRNet (Section 4.2), and the combined system obtained by integrating GRNet with the state-of-the-art model-based architecture, that we call LGRNet (Section 4.3). In Section 4.4, we report the experimental setup, including a description of the automated planning domains used in our experiments, the selected data sets and evaluation measures. The experimental results are reported in Section 4.5. In this section, after reporting the hyperparameters of the networks, we compare GRNet and LGRNet with the state-of-the-art model based goal recognition system LGR on different test sets. We also provide a more in-depth analysis of our models by testing different problems with different difficulty classes and by evaluating their sensitiveness to the training data. Lastly, in Section 4.6, we propose a brief discussion on the strengths and weaknesses of the presented approaches.

In Chapter 5, we address some of the weaknesses described in Section 4.6 by introducing Fast and Slow Goal Recognition. We first describe the new architecture analysing all the different parts that compose it: S1, S2 and the Meta-Cognitive Agent. Then we describe the experimental setup (Section 5.2) and the experimental results we obtained (Section 5.3).

Finally, in Chapter 6, we describe a BERT model with a custom designed training task that we called *planning language modeling*. After describing in detail the model and how it can be trained on different tasks (Section 6.1), in Section 6.2 we introduce the four designed evaluation tasks and the data used for training the model. Then, we show the performances obtained by PlanBert on these tasks considering six well-known planning domain. In the last section, we discuss some of the pros and cons about the presented approach.

Chapter 2

Automated Planning

One of the core problems of Artificial Intelligence (AI) is the problem of intelligent behaviour, that is the capability of using one's knowledge about the world to make decisions in novel situations. In particular, we are interested in selecting the next action that an autonomous agent have to perform in an environment. In the literature, we can find three different approaches to solve these problem:

- **programming-based approach**: in this approach the programmer knows the next action to do and codes it into the agent; this solution is provided as a program or a collection of rules and behaviours;
- **learning-based approach**: in this approach the solution is induced from experience; usually the solution is provided exploiting Machine Learning techniques such as Deep Learning or Reinforcement Learning;
- **model-based approach**: in this approach the solution is provided automatically starting from a model of the actions, sensors and goals.

While *Automated Planning* is often defined as the branch of AI that deals with synthesis of plans of actions to achieve a goal, it is best conceived as the model-based approach for action selection.

Model-based approaches to the action selection problem are composed by three parts:

- **the model** that expresses the dynamics, feedback, and goals of the agent;
- **the languages** that express these models in a compact form;
- **the algorithms** that use these representations of the models to generate the solution.

In this chapter, we introduce the concept of Classical Planning, we give a strict definition of goal recognition, that is the addressed task, and highlight its commonalities and differences with plan recognition. Finally we describe the current state-of-the-art approaches in these fields.

2.1 Classical Planning

In this thesis, we will work within the branch of Automated Planning called *Classical Planning*. In Classical Planning the environment is deterministic and the initial state and goals are fully known. The model underlying classical planning can be described as a model $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where:

- S is a finite and discrete set of possible states. These states are a representation of all the possible configurations of the environment;
- $s_0 \in S$ is the initial state (i.e. the initial configuration of the environment);
- $S_G \subseteq S$ is the non-empty set of goal states;
- $A(s) \subseteq A$ is the set of actions in A that are applicable in each state $s \in S$;

- $f(a, s)$ is a deterministic transition function where $s' = f(a, s)$ is the state that follows s after applying action $a \in A(s)$;

- $c(a, s)$ is a positive cost for executing action a in the state s.

A solution or ***plan*** π is a sequence of applicable actions $\pi = a_0, a_1, \ldots, a_n$ that generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ where $a_i \in A(s_i)$, $s_i \in S$, $s_{i+1} = f(a_i, s_i)$ and s_{n+1} is a goal state (i.e. $s_{n+1} \in S_G$). The cost of the plan is defined as:

$$cost(\pi) = \sum_{i=0}^{n} c(a_i, s_i) \tag{2.1}$$

That is the sum of the action costs $c(a_i, s_i)$ with $i \in [0, n]$. A solution plan is optimal if it has minimum cost among all possible solution plans. A common cost structure consists in setting all action costs $c(a, s)$ to 1. In this case the cost of the plan is given by its length, and the optimal plans are the shortest ones.

There are two types of language for expressing classical planning models in a compact form: in the first, the state variables are all boolean while in the other they are multivalued and can be represented as a value from a finite domain [29]. The simplest classical planning language in use is STRIPS [21], a language based on boolean variables. In STRIPS, the boolean variables that compose a state can be called *facts*, *fluents* or *atoms*. A planning problem in STRIPS is represented by a tuple $P = \langle F, I, A, G \rangle$ where:

- F represents the set of atoms or propositions of interest;

- $I \subseteq F$ represents the initial state;

- A represents the set of actions;

- $G \subseteq F$ represents the goal.

```
(define (domain blocksworld)
(:requirements :strips :equality)
(:predicates (clear ?x) (on-table ?x) (arm-empty) (holding ?x) (on ?x ?y))
(:action pick-up
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
               (not (arm-empty))))
(:action put-down
  :parameters  (?ob)
  :precondition (and (holding ?ob))
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
               (not (holding ?ob))))
(:action stack
  :parameters  (?ob ?underob)
  :precondition (and  (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))
(:action unstack
  :parameters  (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
               (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty))))
)
(define (problem pb2)
   (:domain blocksworld)
   (:objects block_a block_b)
   (:init (on-table block_a) (on-table block_b)  (clear block_a)
          (clear block_b) (arm-empty))
   (:goal (and (on block_a block_b)))
)
```

Figure 2.1: An instance of the BLOCKSWORLD domain expressed in PDDL

In STRIPS, the actions $a \in A$ are represented by three sets of atoms over F called the Add ($Add(a)$), Delete ($Del(a)$) and Precondition ($Pre(a)$) lists. The Add list describes the atoms that the action a makes true, the Delete list describes the atoms that a makes false and the Precondition list describes the atoms that must be true in order for the action a to be executed.

Planning Domain Definition Language (PDDL) [42] is a language that extends the STRIPS language by adding a number of syntactic constructs, like, for instance, allowing preconditions and goals to contain negative literals. In Figure 2.1, we report the description of the well-known BLOCKSWORLD domain, in which the agent has the goal of building one or more stacks of blocks, and only one block may be moved at a time, expressed in PDDL. As we can see in Figure 2.1, PDDL problems are expressed in two parts enclosed in a “define” clause:

- the **domain** part: it represents the general domain. In this part the actions are expressed using generic atoms defined using predicate names, variables and constants;

- the **problem** part: it represents a particular domain instance. In this part, the object names that will replace the variables are declared, together with the atoms used to describe the initial state, and a formula describing the goal state.

As we can see in Figure 2.1, in the BLOCKSWORLD domain the predicates are `clear`, `on-table`, `holding`, `arm-empty` and `on`. The predicates `clear`, `on-table` and `holding` represent the property that the objects (i.e. the blocks) can have; the objects are expressed through the use of variables (e.g. `?x`). The predicate `arm-empty` is a property of the environment as it does not involve any object while the predicate on expresses the relation between two objects.
Following the predicates definition we have four actions: `Pick-Up`, `Put-Down`, `Stack` and `Unstack`. `Pick-Up` is the action of lifting a block, represented by the variable `?ob`, from a table; similarly, `Put-Down` is the action of dropping a block on the table. The action `Stack` concerns lifting a block `?ob` from the top of another block `?underob`; the opposite action of dropping `?ob` on the top of

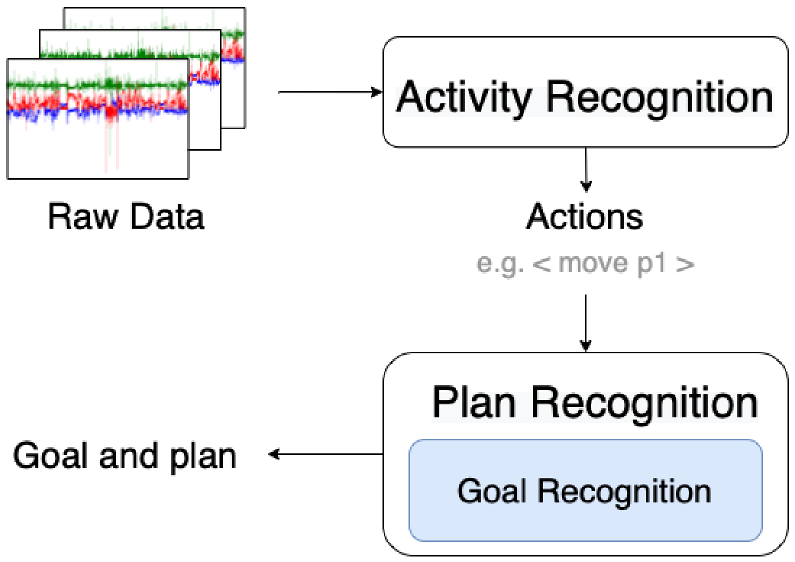


Figure 2.2: Activity, Goal and Plan Recognition

`?underob` is represented by action `Unstack`.

Finally, in the last define clause of Figure 2.1, we have the problem definition. In this very simple problem instance there are two blocks: `Block_A` and `Block_B`. In the initial state both blocks are on the table and the goal of the problem is to reach a state where `Block_A` is on top of `Block_B`.

2.2 Goal and Plan Recognition

Goal recognition, and its more general form of plan recognition, are classic problems in artificial intelligence [24]. These tasks are connected by their shared focus on recognizing patterns in action sequences, and they are widely regarded as key components of human intelligence. The capacity to interpret patterns in others' actions is critical for navigating everyday situations, such as driving a car, holding a conversation, or playing a sport. Since in the literature sometime they are not clearly distinguished, we will provide an informal definition of them following [62] and we will refer to these fields of research considering the reported definitions throughout this work of thesis. Among these definitions, we also include the Activity Recognition definition, since it's a very related task even though it is not addressed in this thesis. Figure 2.2 reports an overview of the different roles of activity, goal and plan recognition in the process of recognizing the plan and the goal of an agent from raw data.

Activity recognition concerns analyzing sequences of data generated by humans or autonomous agents acting in an environment, to identify the corresponding activity that they are performing. These data are typically low-level data, like the ones that can be collected from wearable sensors, accelerometers, or like images to recognize human activities such as running, cooking, driving, etc. [65].

Goal recognition (GR) is the problem of identifying the intention (goal) of an agent from observations about the agent behaviour in an environment. These observations can be represented in different ways such as an ordered sequence of images, each one representing a state or an ordered sequence of actions identified by activity recognition. The agent's goal can be expressed either as a set of propositions or a probability distribution over alternative sets of propositions (each one forming a distinct candidate goal).

Plan recognition is more general than GR and concerns both recognising the goal of an agent and identifying the full ordered set of actions (plan) that have been, or will be, performed by the agent in order to reach that goal; as GR, typically plan recognition takes as input a set of observed actions performed by the agent [12]. A plan recognition task includes the goal recognition task and complements it with the task of defining a set of observed and predicted actions and the relations between them that will lead to that goal.

2.2.1 Model-based and Model-free Goal Recognition

The approach to GR known as "goal recognition over a domain theory" [56, 59, 61, 62], represents the agent/environment states and the set of actions A that the agent can perform; typically it is specified by a planning language such as PDDL, as described in Section 2.1. An *instance of a GR problem* is represented by a tuple $T = \langle \Pi, I, O, \mathcal{G} \rangle$ where:

- $\Pi = \langle F, A \rangle$ is a planning domain where F specifies the set of possible fluents and A represents the set of actions
- an initial state I of the agent and environment ($I \subseteq F$);
- a sequence $O = \langle obs_1, .., obs_n \rangle$ of observations ($n \geq 1$), where each obs_i is an action in A performed by the agent;
- and a set $\mathcal{G} = \{G_1, .., G_m\}$ ($m \geq 1$) of possible goals of the agent, where each G_i is a set of fluents over F that represents a partial state.

The observations form a trace of the full sequence π of actions performed by the agent to achieve a goal G^*, i.e., a trace of the agent plan. Such a plan trace is a selection of actions in π, ordered as in π; the selected actions can be non-consecutive in π. Solving a GR instance consists of identifying the G^* in $\mathcal{G}$ that is the hidden goal of the agent.

The approach based on a model of the agent's actions and of the agent/environment states, that we call *model-based goal recognition* (MBGR), defines GR as a reasoning task addressable by automated planning techniques [29, 44].

An alternative approach to MBGR is *model-free goal recognition* (MFGR) [23, 11]. In this approach, GR is formulated as a classification task addressed through machine learning. The domain specification consists of a fluent set F, and a set of possible actions A, where each action $a \in A$ is specified by just a label, that is a unique identifier for each action.

A MFGR instance for a domain is specified by an observation sequence $O = \langle obs_1, .., obs_n \rangle$, formed by action labels and, as in MBGR, a goal set $\mathcal{G}$ formed by subsets of F. MFGR requires minimal information about the domain actions, and can operate without the specification of an initial state, that can be completely unknown. Moreover, since running a learned classification model is usually fast, a MFGR system is expected to run faster than a MBGR system based on planning algorithms. On the other hand, MFGR needs a data set of solved GR instances from which learning a classification model.

2.3 State of the Art

Goal and plan recognition have been extensively studied through model-based approaches exploiting planning techniques [44, 55, 61, 59, 51] or matching techniques relying on plan libraries (e.g., [47]). Among these approaches, we would like to highlight the work of Ramírez and Geffner [55], that is one of the first work that applies *classical planning techniques* to solve goal and goal recognition problems and the work of Pereira et al. [51], that is the current state-of-the-art approach for MBGR. Finally, as in this thesis we will propose approaches to solve MFGR instances, here we introduce the most relevant approaches for MFGR.

2.3.1 Plan Recognition as Planning

The work of Ramírez and Geffner [55] (PRP) presents two approaches: an exact approach and an approximate approach.

The *exact approach* computes, for all the goals $G \in \mathcal{G}$, the optimal plan from the initial state to the goal and the optimal plan that complies with the observations O from the same initial state to the same goal. The latter is computed by transforming the original goal recognition problem T. Given the original goal recognition problem $T = \langle \Pi, I, O, \mathcal{G} \rangle$, the transformed goal recognition problem is $T' = \langle \Pi', I, O', \mathcal{G}' \rangle$ where:

- $\Pi' = \langle F', A' \rangle$. F' is defined as $F' = F \cup F_o$, where $F_o = \{p_a | a \in O\}$. A' is defined as $A' = A \cup A_o$, where $A_o = \{o_a | a \in O\}$; the new action o_a has the same preconditions and effects of $a \in A$ except for the new fluent p_a that is added to $Add(o_a)$ and the fluent p_b for action b that immediately precedes a in O, if any, that is added to $Pre(o_a)$;

- O' is empty;

- $\mathcal{G}'$ contains the goal $G' = G \cup G_o \; \forall G \in \mathcal{G}$, where $G_o = F_o$.

The solution set for T is $\mathcal{G}^*_T$, while the solution set for T' is $\mathcal{G}^*_{T'}$.
The candidate goals are all the goals for which the cost of these two optimal plans is the same; formally, $G \in \mathcal{G}$ *iif* $c^*_{\Pi'}(G) = c^*_{\Pi'}(G')$ where $c^*_{\Pi'}(G)$ is the optimal cost for achieving G in Π'.
Although this approach is formally exact, its major drawback is that it requires checking all the possible goals in the hypothesis set, with several calls to the planner, which could take a long time.

The *approximate approach* tries to mitigate this drawback computing a set of "good plans" for all the goals $G \in \mathcal{G}$ using a modified version of Fast-Forward [34] instead of the optimal plans. In this case the candidate goals are the ones that contain the highest number of actions in the observations. This approach is faster than the exact one but, as the name suggests, it does not provide any formal guarantee of correctness.

A follow-up of this work [56] expands the Ramírez and Geffner [55] method using standard planners and providing a probability distribution over the candidate goals.

2.3.2 Landmark-Based Heurisitics for Goal Recogniton

The work in Pereira et al. [51] (LGR) is a state-of-the-art approach for model-based goal recognition. LGR is a planning-based system exploiting landmarks [35] (i.e., propositions that are necessarily true in every sequence of actions reaching a goal from an initial state). In this work the authors develop two heuristic methods for solving a goal recognition problem: the Goal Completion heuristic and the Uniqueness heuristic.

The *Goal Completion heuristic* (h_{gc}) aggregates the percentage of completion of each sub-goal into an overall percentage of completion for all facts of a candidate goal $G \in \mathcal{G}$. For a goal G, the heuristic h_{gc} is computed as:

$$h_{gc}(G, \mathcal{AL}_G, \mathcal{L}_G) = \frac{\sum_{f \in G} \frac{|\mathcal{AL}_f \in \mathcal{AL}_G|}{\mathcal{L}_f \in \mathcal{L}_G}}{|G|} \tag{2.2}$$

Where $\mathcal{AL}_f$ is the set of achieved landmarks from observations of every sub-goal f and $\mathcal{L}_f$ represents the set of necessary landmarks to achieve every sub-goal f. For each candidate goal G, we define the set $\mathcal{AL}_G$ as $\mathcal{AL}_G = \{\mathcal{AL}_f \mid f \in G\}$ and the set $\mathcal{L}_G$ as $\mathcal{L}_G = \{\mathcal{L}_f \mid f \in G\}$. This heuristic estimates the completion of a goal G by computing the ration between the sum of the percentage of the completion of every sub-goal and the number of sub-goals in G.

The *Uniqueness heuristic* (h_{uniq}) aims to capture how unique, and as a consequence informative, each landmark is to help to disambiguate similar goals for a set of candidate goals. To express this concept, the *landmark uniqueness* (L_{uniq}) is computed as the inverse frequency of a landmark among the landmarks found in the set of candidate goals $\mathcal{G}$:

$$L_{uniq}(\mathcal{L}_f, \mathcal{L}_\mathcal{G}) = \frac{1}{\sum_{\mathcal{L}_G \in \mathcal{L}_\mathcal{G}} |\{\mathcal{L}_f | \mathcal{L}_f \in \mathcal{L}_G\}|} \tag{2.3}$$

The value of h_{uniq} for a goal G is computed by summing the uniqueness values of the landmarks achieved in the observations. Intuitively this heuristic weights the completion value by the informative value of a landmark so that unique landmarks have the highest weight. Formally, h_{uniq} is defined as:

$$h_{uniq}(G, \mathcal{AL}_G, \mathcal{L}_G, \Upsilon_{uv}) = \frac{\sum_{\mathcal{AL}_f \in \mathcal{AL}_G} \Upsilon_{uv}(\mathcal{AL}_f)}{\sum_{\mathcal{L}_f \in \mathcal{L}_G} \Upsilon_{uv}(\mathcal{L}_f)} \tag{2.4}$$

Where Υ_{uv} contains the landmark uniqueness value (L_{uniq}) of every landmark $\mathcal{L}_f \in \mathcal{L}_G \; \forall G \in \mathcal{G}$, computed following the Equation 2.3.

LGR achieves good performance in terms of accuracy solving GR instances much faster than PRP. However, this speed is achieved by relaxing the formal guarantees provided by PRP; in fact, LGR does not provide any guarantees of correctness if the observation trace is incomplete (i.e. $O \neq \pi*$), which is the normal use-case.

2.3.3 MFGR approaches

Concerning MFGR and in particular GR systems using neural networks, some works use them for specific applications, such as game playing [46]. In order to extract useful information from image-based domains and perform goal recognition, Amado et al. [3] used a pre-trained encoder and a LSTM network, which will be described in Section 3.2.2, for representing and analysing a sequence of observed states. Amado et al. [4] trained a LSTM-based system to identify missing observations about states in order to derive a more complete sequence of states that can be analysed by a MBGR system based on classical planning techniques.

Borrajo et al. [11] investigated the use of XGBoost [13] and LSTM neural networks for goal recognition using only traces of plans, without any knowledge on the domain. They train a specific machine learning model *for each goal recognition instance* (the goal set $\mathcal{G}$ is fixed), using instance-specific datasets for training and testing. Moreover, the experimental evaluation of the networks proposed in [11] use peculiar goal recognition benchmarks with custom-made instances.

Maynard et al. [41] compared model-based techniques and approaches based on deep learning for goal recognition. As in [11], such a comparison is made using specific instances, and several kinds of neural networks are trained to directly predict the goal among a set of possible ones. This makes the trained networks in [41] specific for the considered GR instances in a domain. While in a typical goal recognition problem we can have missing observations across the entire plan of the agent(s), the work in [41] considers only observations from the start of the plan to a given percentage of it, treating every possible successive observation as missing.

Chapter 3

Deep Learning

In this thesis, as mentioned in Section 2.2.1, we tackle the problem of Goal Recognition. However, we study some novel approaches to solve it; to do so, we formulate a GR problem as a classification task and we address this task through deep learning. In this chapter we show the formalisation and the main idea behind some of the state-of-the-art approaches adopted in the field of Natural Language Processing such as Word Representation, Recurrent Neural Networks, Attention Mechanism and Transformers.

3.1 Word Representations

In order for the machine learning system to process an GR problem, we chose to represent each observation as a word. In Computer Science, a word can be represented in two ways: it can be mapped into a number, which is generally referred to as *encoding* or it can be projected into a latent space of arbitrary dimensions (i.e., it is mapped into a vector), which is called *embedding*.

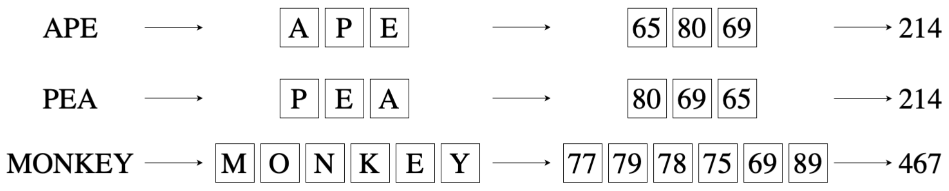
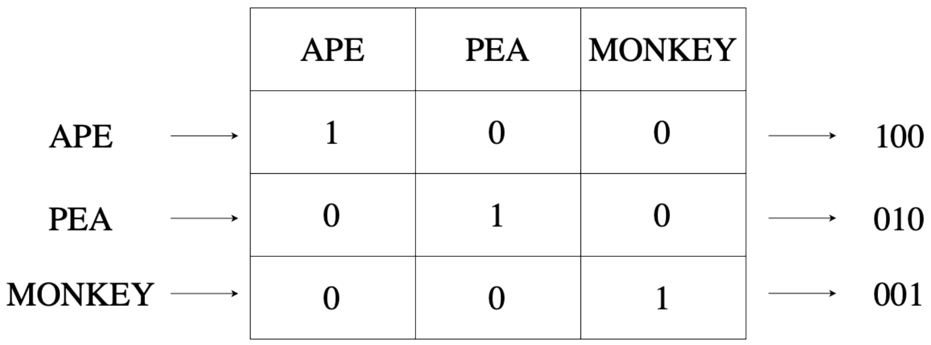


Figure 3.1: ASCII encoding

3.1.1 Encoding

Encoding involves the use of a code to change the original data into a form that can be used by an external process. We can intuitively represent an image as a colored pixel matrix, where each pixel is encoded using three numbers between 0 and 255. Similarly, we can represent a word as a sequence of characters, which are then converted into numbers using the American Standard Code of Information Interchange (ASCII). Summing these numbers is one of the simplest way of encoding of a word. In Figure 3.1 we show an example of the ASCII encoding for three words: APE, PEA, and MONKEY. Figure 3.1 also highlights the bigger problems of this approach: the encoding of the word APE is the same as that of the word PEA; in fact, for an algorithm that only processes the encodings without any knowledge of the original words, these two words are indistinguishable. This makes the ASCII encoding not suitable for machine learning approaches.

A widely used encoding method that solves this problem is the *One-Hot Encoding*. Given a set of words with size K that we want to encode, in order to obtain their one-hot encoding, we first assign a unique integer identifier from 0 to $K-1$ to each of them. The encoding of the word with $ID = j$ is a binary vector of size K with 1 in the j-th position and 0 in all the other positions. Figure 3.2, reports the one-hot encoding with $K = 3$ for the same three words: APE, PEA and MONKEY. As we can see, using this encoding, we can guarantee that each



	APE	PEA	MONKEY	
APE →	1	0	0	→ 100
PEA →	0	1	0	→ 010
MONKEY →	0	0	1	→ 001

Figure 3.2: One-Hot encoding

word has a unique representation. One-Hot Encoding is a good way to encode words, but has some limitations: First of all, a high cardinality vector is needed to represent each word; this can be a problem, as most machine learning algorithms can only handle low-dimensional numerical data as input [64]. Second, this encoding does not take into account one of the most important aspects of a word: its meaning. For example, the words APE and MONKEY are synonyms, but the fact that they are similar and can be interchanged cannot be implied in any way. In fact, when using this encoding, the word distribution sits in an orthogonal vector space (that is, $OneHot(word_1) \cdot OneHot(word_2) = \bar{0}$ for each couple of considered words). Finally, the role of the word in the sentence, its grammatical features, and its connections to other words are almost impossible to represent through this encoding.

3.1.2 Embedding

An embedding is a relatively low-dimensional space used to translate high-dimensional vectors. Embeddings are widely used in machine learning to handle large categorical inputs; in Natural Language Processing (NLP), for example, embedding is used to translate sparse vectors that represent words into a low-dimensional

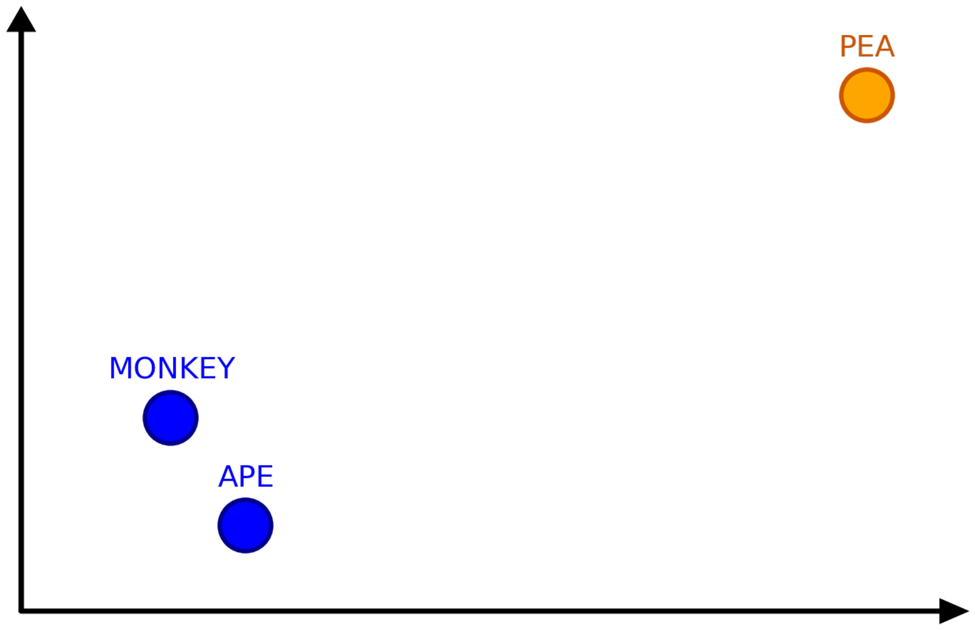


Figure 3.3: Bi-dimensional word embedding

representation [2]. The embedding provides a word representation that considers its meaning and its role in the sentence. In fact, each word is represented as a n-dimensional vector of real numbers, also called *word vector*. We can think of this representation as a single point in a vector space of size n. When creating this representation, two main factors must be considered:

- words whose meanings are closely related should be represented in the same portion of the vector space; in particular synonyms, singular and plural terms and words with different conjugations should have very close representations
- words whose meaning or context of application is very different should be far from each other in the vector space

Figure 3.3 shows, as a toy-example, a two-dimensional embedding for the three words APE, PEA and MONKEY. As we can see, in the bottom left part of the image, the synonym words APE and MONKEY have a very close representation, while the representation of the word PEA, which is not related to the other two, is on the top right side of the image, far from the other representations. In recent years, many algorithms like GloVe [50], FastText [37] and the most famous and used Word2Vec [45] have been developed to understand how language works and obtain an embedding representation of words.

However, in this work, we need a representation of tokens that are specific to our application; for this reason, we opted to compute the embeddings while training our supervised machine learning model model. Actually, this process allows us to encode the categories we predict in the classification task into embedding vectors. The embedding shown in Figure 3.3, for example, could also be obtained from a model that classifies vegetables and animals. This means that this training encodes ad hoc embedding that contains relevant features for the specific task, grouping words so that embedding vectors are words that belong to the same class.

3.2 Recurrent Neural Networks

A recurrent neural network (RNN) [19] represents one of the two primary categories within the realm of artificial neural networks, distinguished by the manner in which information flows between its layers. In contrast to the unidirectional nature of feedforward neural networks, an RNN constitutes a bidirectional artificial neural network. This means that it allows the output from some nodes to affect subsequent input from the same nodes. In particular, RNNs are implemented for their ability to use internal state (also called *memory*) to process

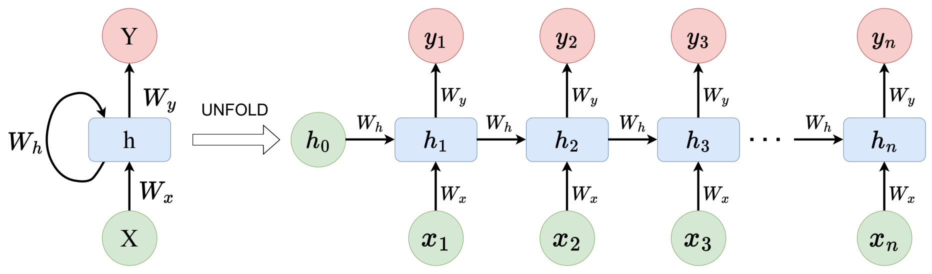


Figure 3.4: Compressed (left) and unfolded (right) representation of a Recurrent Neural Network having as input a sequence X of length n. W_x, W_h and W_y are the weight matrices computed by the network.

arbitrary sequences of inputs. In fact, RNNs analyze one element of the input sequence at a time and compute the i-th input, also considering what was provided in the previous $i-1$ inputs.

3.2.1 Standard RNNs

As an example, let us consider a sentence X of n words. Taking into account the word w_t at position t, we will call its embedding $x_t \in \mathbb{R}^k$ where k is the size of the embedding space and $t \in [1, n]$. Figure 3.4 shows the structure of an RNN both compressed (on the left) and unfolded (on the right). When unfolded, the RNN can be seen as a feedforward neural network of n layers where each layer has d neurons and d is a hyperparameter of the network. The t-th layer takes as input x_t and h_{t-1}, where the latter is the memory computed by layer $t-1$.

Considering the input embedding at time t as $x_t \in \mathbb{R}^k$ and its corresponding output as $y_t \in \mathbb{R}^m$, we can define the weight matrices $W_x \in \mathbb{R}^{d\times k}$, $W_h \in \mathbb{R}^{d\times d}$ and $W_y \in \mathbb{R}^{m\times d}$ and the bias vectors $b_h \in R^d$ and $b_y \in R^m$ so that:

$$h_t = \sigma_h(W_x x_t + W_h h_{t-1} + b_h) \tag{3.1}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{3.2}$$

Where σ_h is an activation function like *ReLU* or *tanh* and σ_y is an activation function that depends on the addressed task. Equation 3.1 shows how the RNN processes each element of the sequence considering both the current input x_t and the memory computed during the previous computation h_{t-1}, creating the new memory h_t. Equation 3.2, instead, computes the output of the RNN after processing the t-th element of the sequence.

Please note that, as shown on the left of Figure 3.4, these weight matrices (and also the bias vectors) are shared among all layers. The idea behind this implementation is that a word has the same meaning independently of its position in the sentence, and, for this reason, it should always be processed in the same way. This weight-sharing allows the use of fewer computational resources and a shorter computation time because we have to train fewer weights compared to a fully connected neural network.

However, the standard RNNs have two main issues:

- Exploding and Vanishing Gradients [49]: the exploding gradients problem refers to the large increase in the norm of the gradient during training. The vanishing gradient problem refers to the opposite behavior, when the gradients go exponentially fast to norm 0, making it impossible for the model to learn correlation between temporally distant events;

- Long-term dependency [28]: while processing a long document, for instance, the RNN has a tendency to forget the context expressed in the first sentences. When doing so, important parts of the document might be forgotten, and the RNN might not be able to process them properly.

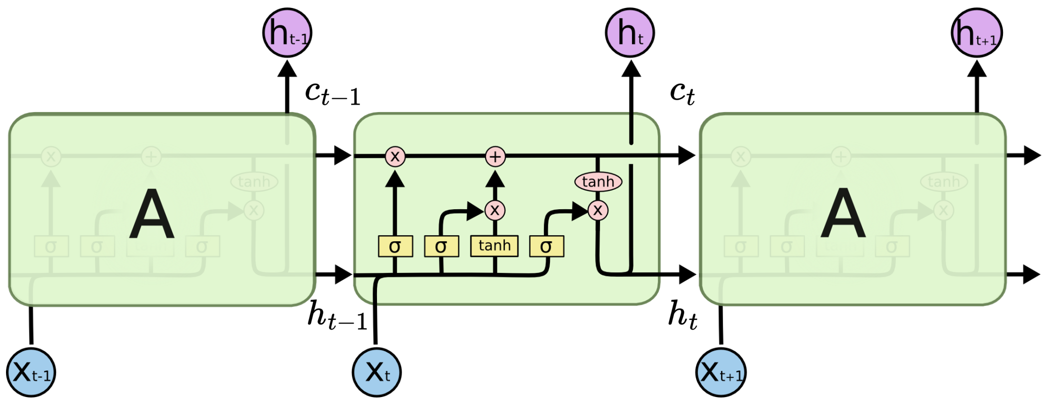


Figure 3.5: Unfolded representation of a LSTM cell. The yellow blocks represent a neural network layer with sigmoid (σ) or $tanh$ activation. Pink circles represent point-wise operations. Lines merging denote concatenation, while a line forking denote its content being copied.

3.2.2 Long Short Term Memory

Long Short Term Memory networks (LSTMs) [33], are a special kind of RNN, which were designed to solve the problems of the standard RNNs. In Figure 3.5, we show the unfolded representation of an LSTM network. The new component introduced by this network is the *cell state* c_t, which works as memory buffer storing long-term information. The LSTM has the ability to remove or add information to the cell state, through structures called gates. Gates are made up of a neural network layer with a sigmoid activation function and a point-wise multiplication operation. An LSTM has three gates: the forget gate, the input gate, and the output gate.

The *forget gate* f_t, as the name suggests, decides what information in the previous cell state c_{t-1} will be forgotten considering the current input x_t and the previous output h_{t-1}. More formally, the forget gate $f_t \in (0,1)^k$ can be expressed as:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.3}$$

Where σ is the sigmoid activation function, $W_f \in \mathbb{R}^{k\times(k+n)}$ is a weight matrix, $[h_{t-1}, x_t] \in \mathbb{R}^{k+n}$ is the concatenation of the previous output $h_{t-1} \in (-1, 1)^k$ and the input vector $x_t \in \mathbb{R}^n$ and $b_f \in \mathbb{R}^k$ is the bias vector.

The *input gate* i_t decides how relevant the new information is considering the input x_t and the previous output h_{t-1}. The LSTM also creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state from the information in $[h_{t-1}, x_t]$. Using the input gate, the forget gate, and the candidate values, the LSTM can update the old cell state c_{t-1} into the new cell state c_t: multiplies the old state c_{t-1} by f_t, forgetting the things decided by the forget gate; then it adds $i_t \cdot \tilde{C}_t$ which are the new candidate values, scaled by how much the input gate decided to update each state value.
More formally, the input gate $i_t \in (0, 1)^k$ can be expressed as:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{3.4}$$

Where $W_i \in \mathbb{R}^{k\times(k+n)}$ is a weight matrix and $b_i \in \mathbb{R}^k$ is the bias vector.
The candidate values vector $\tilde{C}_t \in (-1, 1)^k$ can be formalized as:

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_i) \tag{3.5}$$

Where $tanh$ is the layer activation function, $W_c \in \mathbb{R}^{k\times(k+n)}$ is another weight matrix and $b_c \in \mathbb{R}^k$ is its bias vector.
Finally the new cell state $c_t \in \mathbb{R}^k$ can be formalized as:

$$c_t = i_t \odot \tilde{C}_t + f_t \odot c_{t-1} \tag{3.6}$$

Where $\odot$ is the element-wise product.

The *output gate* o_t, finally, decides which parts of the cell state can be provided as output considering the current input x_t and the previous output h_{t-1}. To compute the new output h_t, which stores the short-term information, the cell state is given as input to a $tanh$ function (to push the values to be between -1 and 1) and multiplied by the output gate.
More formally the output gate o_t can be expressed as:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.7}$$

Where $W_o \in \mathbb{R}^{k\times(k+n)}$ is a weight matrix and $b_o \in \mathbb{R}^k$ is its bias vector.
The output of the LSTM $h_t \in (-1, 1)^k$ can then be written as:

$$h_t = tanh(c_t) \odot o_t \tag{3.8}$$

3.2.3 Gated Recurrent Unit

Gated Recurrent Units (GRUs) [16] are another type of recurrent neural network that attempts to simplify the structure and, therefore, the training of LSTMs. In Figure 3.6, we show the structure of an unfolded GRU network. We can see that, unlike the LSTM, the GRU cell does not implement a cell state c_t; in fact, its role is replaced by output h_t. Similarly to the LSTM, the architecture of a GRU cell is also based on gates; the GRU has two gates: the reset gate and the update gate.

The *reset gate* r_t has the same function as the forget gate f_t: it decides which information from the previous output h_{t-1} will be forgotten considering also the current input x_t. It can be formalized as:

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{3.9}$$

Where $r_t \in (0, 1)^k$, σ is the sigmoid activation function, $W_r \in \mathbb{R}^{k\times(k+d)}$ is a weight matrix, $[h_{t-1}, x_t] \in \mathbb{R}^{k+n}$ is the concatenation of the previous output $h_{t-1} \in (-1, 1)^k$ and the input vector $x_t \in \mathbb{R}^n$ and $b_r \in \mathbb{R}^k$ is the bias vector.

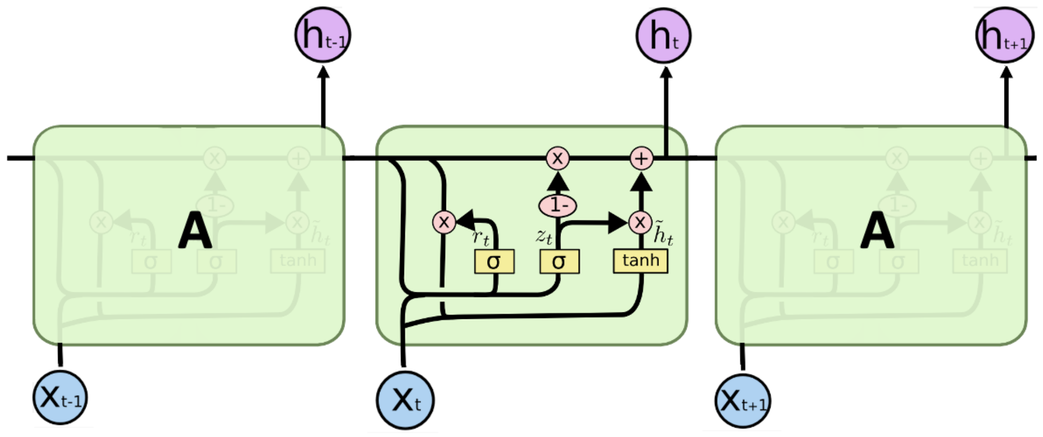


Figure 3.6: Unfolded representation of a GRU cell. The yellow blocks represent a neural network layer with sigmoid (σ) or $tanh$ activation. Pink circles represent point-wise operations. Lines merging denote concatenation, while a line forking denote its content being copied.

The *update gate* z_t, similar to the input gate i_t of the LSTM, decides how relevant the new information is, considering the previous output h_t. We can write the update gate $z_t \in (0,1)^k$ as:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{3.10}$$

Where $W_r \in \mathbb{R}^{k\times(k+d)}$ is a weight matrix and $b_z \in \mathbb{R}^k$ is its bias vector.

Unlike LSTM, the new candidate output $\tilde{h}_t$ is a vector of new candidate values that is generated from the relevant information contained in the previous output (i.e. $h_{t-1} \odot r_t$) and the current output x_t. Finally, the new output h_t will be the sum of the important information, computed by reversing the update gate z_t, contained in the previous output h_{t-1} (i.e., $(1 - z_t) \odot h_{t-1}$), and the relevant information contained in the candidate output $\tilde{h}_t$ (i.e., $z_t \odot h_t$). Please note that in the GRU cell there is no output gate.

The new candidate output $\tilde{h}_t \in (-1, 1)^k$ can be written as:

$$\tilde{h}_t = tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) \tag{3.11}$$

Where $\odot$ is the element-wise product, $tanh$ is the layer activation function, $W_h \in \mathbb{R}^{k\times(k+d)}$ is a weight matrix, $[r_t \odot h_{t-1}, x_t]$ is the concatenation of the element-wise product between $r_t \in (0, 1)^k$ and $h_{t-1} \in (-1, 1)^k$, and the current input $x_t \in \mathbb{R}^n$, and $b_h \in \mathbb{R}^k$ is the bias vector.
Finally the new output $h_t \in (-1, 1)^k$ can be expressed as:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.12}$$

3.3 Attention Mechanisms

Despite the improvement provided by the implementation of recurrent neural networks such as LSTM and GRU, their ability to connect elements in a document that are far from each other is still limited [54]. For example, suppose that we want to run a classification task on a n words document, where the word in position t is represented by its embedding w_t for $t \in [1, n]$ using an LSTM. When n is large, it can happen that the output h_n for the word w_n is not influenced at all by the first words (e.g. w_1) and it's too depending on the last ones. To solve this problem, *Attention mechanisms* [6, 54] have been developed.

3.3.1 Feed-forward Attention

Feed-forward Attention [54] applies the Attention Mechanism presented in [6], which was used in encoder-decoder systems for machine translations, to Feed-forward Neural Networks. Figure 3.7 reports a schematic of the Feed-forward Attention mechanism. Given a state sequence h of size n (i.e. $h_t \in h$ for $t \in [1, n]$),

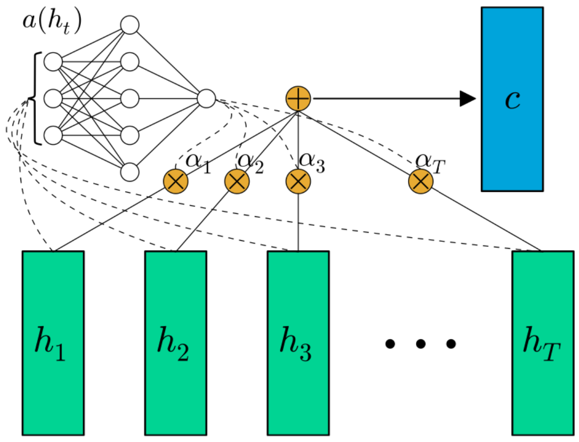


Figure 3.7: Schematic of the Feed-Forward Attention mechanism. Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α. Figure from [54].

produced by a feed-forward neural network, we want to compute a context vector c that is an embedding of the sequence h obtained by computing an adaptive weighted average of that sequence. This vector is a representation of the entire sentence that can be learned during network training to improve the quality of the sequence representation and help address memory-related problems.

The context vector c can be expressed as:

$$c = \sum_{t=0}^{n} \alpha_t h_t \tag{3.13}$$

Where α_t is the weight associated to the state h_t.
Each weight α_t can be computed as the result of the softmax function σ applied on the element $e_t \in e$:

$$\alpha_t = \sigma(e)_t = \frac{exp(e_t)}{\sum_{k=1}^{n} exp(e_k)} \tag{3.14}$$

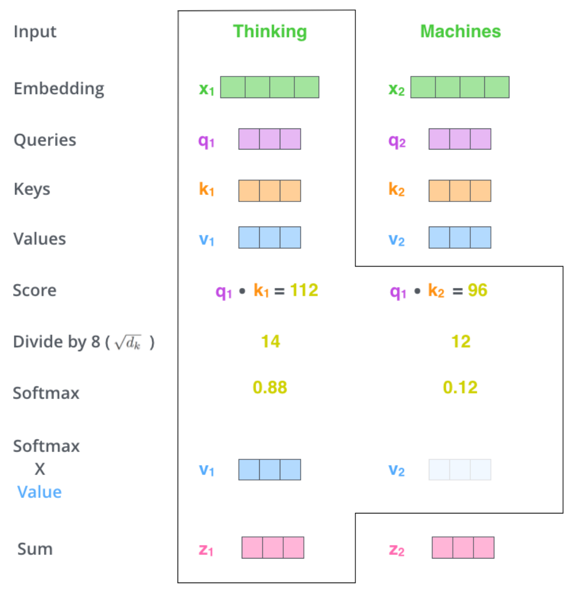


Figure 3.8: Steps of the Self-Attention procedure.

Where e_t is an integer that represents the relation between the final output and the state h_t. Formally, this relation is represented by a function, called a, which is a learnable function.

$$e_t = a(h_t) \tag{3.15}$$

Usually, the function a is computed using a feed-forward neural network.

3.3.2 Self-Attention

The Self-Attention mechanism [63] computes the interaction among all words in the considered document or sentence. The intuition behind this approach is that the RNNs are not the only way to process a sentence or a document; a sequence of attention mechanism and fully-connected layers can be used to process all the words of a document in parallel and can achieve even better results.

Considering a document of n words, the Self-Attention takes as input a word embedding $x_t \in \mathbb{R}^d$ with $t \in [1, n]$, and computes three different representations of the same word, namely *query* (q_t), *key* (k_t) and *value* (v_t), using three weights matrices $W_q, W_k, W_v \in \mathbb{R}^{b \times d}$:

$$q_t = W_q x_t,\ k_t = W_k x_t,\ v_t = W_v x_t \tag{3.16}$$

Where b represents the size of internal representation of the word, and it's an hyperparameter set by the user; usually its value is set to 512 or 768 [63].
These three representations are then used to create a final representation z_t of the word x_t, considering the relations between all the other words in the document and the word itself to better represent its meaning. The final representation z_t can be expressed as:

$$z_t = \sum_{i=1}^{n} \alpha_{it} v_i \tag{3.17}$$

Where α_{it} is a scalar value that represents the influence of word x_i with $i \in [1, n]$ on the input word x_t and v_i is the value representation of the word x_i.
The value α_{it} is computed as:

$$\alpha_{it} = \sigma\left(\frac{q_t k^T}{\sqrt{b}}\right)_i = \frac{exp\left(\frac{q_t k_i^T}{\sqrt{b}}\right)}{\sum_{j=1}^{n} exp\left(\frac{q_t k_j^T}{\sqrt{b}}\right)} \tag{3.18}$$

Where σ is the softmax function.

Taking into account a two-word toy document, *Thinking* and *Machines*, Figure 3.8 reports the step-by-step procedure to calculate the self-attention of the first word. First, we compute the key (k_1 and k_2), query (q_1 and q_2), and value (v_1 and v_2) representations of the two words (Equation 3.16). Subsequently, we compute the two scores as the scalar product of the query q_1 and the key representation of each word (k_1 and k_2). These scores are then divided by $\sqrt{b} = 8$; in this toy example b is set to 64 for simplicity's sake; usually b is set to 512 or

768. After division operations, the function $softmax$ is applied to both quotients (Equation 3.18), which are then multiplied by their relative value representations. Finally, we sum the two products to obtain the final representation z_1 for word x_1 (Equation 3.17).

Considering an input matrix $X \in \mathbb{R}^{n \times d}$ where $X[t] = x_t$ and $t \in [1, n]$, we can redefine Equation 3.16 as:

$$Q = XW_q,\ K = XW_k,\ V = XW_v \tag{3.19}$$

Where $Q, K, V \in \mathbb{R}^{n \times b}$.

We can also combine Equation 3.17 and Equation 3.18 to obtain a more compact definition of Self-Attention over Q, K, V:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{b}}\right)V \tag{3.20}$$

Where $Attention(Q, K, V) \in \mathbb{R}^{n \times b}$ is a matrix that contains the final representation of the input in each row (i.e. $Attention(Q, K, V)[t] = z_t$ for $t \in [1, n]$)

3.3.3 Multi-Headed Attention

The work in [63] further refined the Self-Attention layer by adding a mechanism called Multi-Headed Attention. This improves the performance of the Self-Attention mechanism by giving it multiple representation subspaces. In fact, with Multi-Headed Attention, we have multiple sets of query, key, value weight matrices; in [63], the authors set the cardinality of these sets, called h, to 8. Each of these sets is initialized at random. After training, each set is used to project the input vector into a different subspace of representation.

Figure 3.9 shows the Multi-Headed Attention mechanism with $h = 8$ for a sentence containing two words: THINKING and MACHINES. Given h different

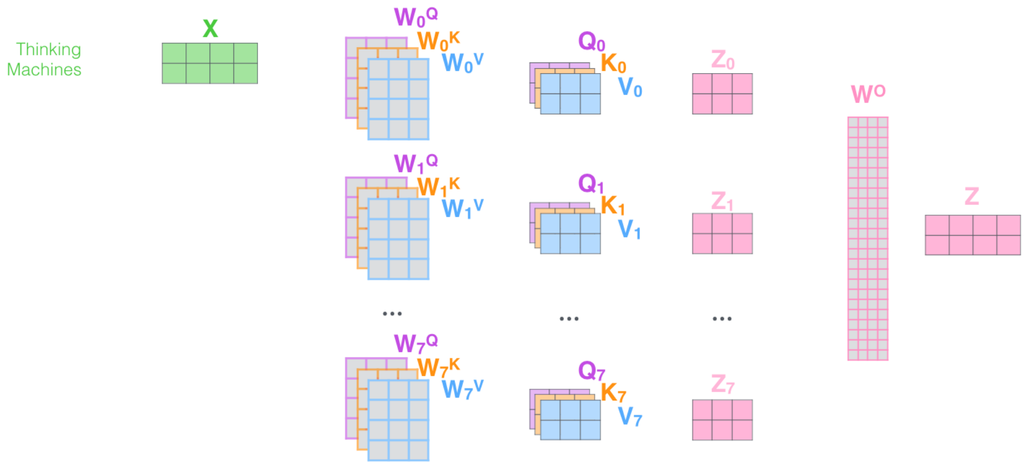


Figure 3.9: Multi-Headed Attention mechanism with $h = 8$ for a simple sentence of two words. X represents the matrix that contains the embedding of the input words. Each later $i \in [0, h-1]$ computes a separate final representation matrix Z_i, also called *head*, using the Self-Attention mechanism. These representation matrices are finally concatenated and multiplied with matrix W^O to obtain the final representation matrix Z

representations, also called *heads*, we can write the Multi-Headed Attention as:

$$MultiHead(Q, K, V) = concat(head_1, head_2, \ldots, head_h)W^O \tag{3.21}$$

Where $W^O \in \mathbb{R}^{hb \times b}$ is a new output matrix that rescales the concatenated heads vector of size $n \times hb$ to a single head shape and $head_i \in \mathbb{R}^{n \times b}$ for $i \in [1, h]$ is defined as:

$$head_i = Attention(XW_i^Q, XW_i^K, XW_i^V) \tag{3.22}$$

Where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times b}$ are the query, key and value weight matrices for head i and $X \in \mathbb{R}^{n \times d}$ is the input matrix.

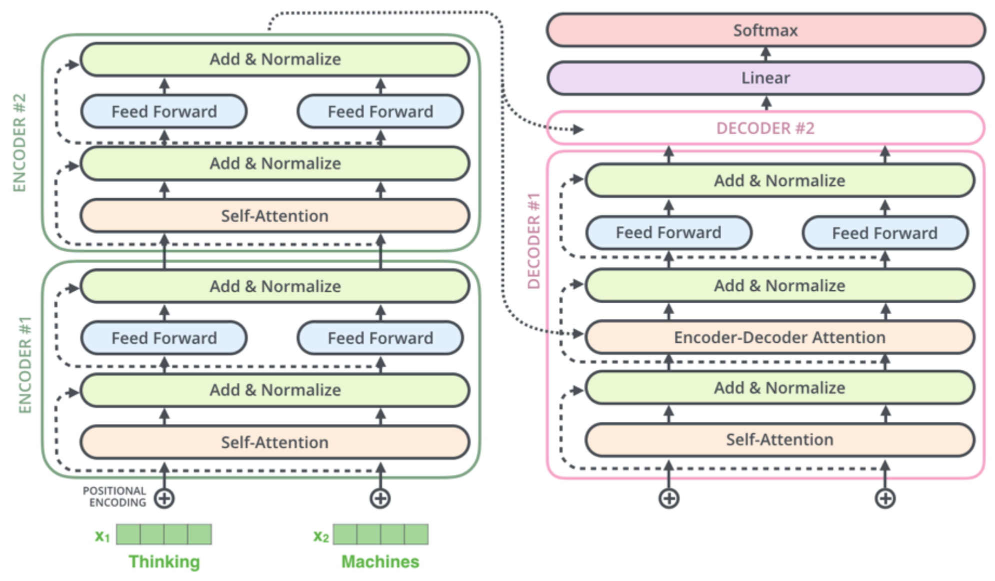


Figure 3.10: Transformer architecture with $N = 2$ stacks of encoder and decoders. The input matrix X is unfolded into its rows x_1 and x_2

3.4 Transformer

A Transformer [63] is a deep-learning architecture based on the parallel Multi-Headed Attention mechanism. This architecture was developed to reduce the computational times needed to train LSTMs and was later used to train large language models (LLMs) using large language datasets.

3.4.1 Architecture

At a high level, the Transformer architecture is composed by an encoder component and a decoder component. The encoder component is a stack of N encoders, while the decoder component is a stack of N decoders; in [63], for example, N is set to 6. In Figure 3.10, we report the model architecture with $N = 2$ stacks of encoders and decoders and a sentence of 2 words (x_1 and x_2) as input.

Embedding

As we can see in Figure 3.10, the input is a matrix $X \in \mathbb{R}^{n \times d}$ that is unfolded into its rows for representation purposes; the output matrix $Y \in \mathbb{R}^{n \times d}$ that is used as input for the decoder stack is not reported. Since the Transformer does not contain recurrence or convolution, in order for the model to make use of the order of the sequence, the Positional Encoding is summed to the matrices X and Y. The Positional Encoding injects information about the relative or absolute position of the tokens in the sequence and it can be either fixed or learned; in the work in [63], the Positional Encoding is fixed and set to:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{(2i/d)}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{(2i/d)}) \end{aligned} \tag{3.23}$$

Where $pos \in [0, n-1]$ is the position of the word in the document and $i \in [0, floor\left(\frac{d}{2}\right)]$ is the embedding dimension.

Encoders and Decoders

As previously mentioned, the transformer is made up of a stack of N identical encoders and a stack of N identical decoders. In Figure 3.10, N is set to 2.

Each encoder (left side of Figure 3.10) has two sub-layers. The first is a Multi-Headed Attention mechanism, and the second is a simple fully connected feed-forward network. A residual connection [30] is implemented around each of the two sub-layers, followed by layer normalization ("Add & Normalize" blocks). That is, the output of each sub-layer is computed as $LayerNorm(X+Sublayer(X))$, where $Sublayer(X)$ is either the $MultiHead$ function (Equation 3.21) for the first sub-layer or a feed-forward network for the second one. In [63], to facilitate the residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $b = 512$.

In addition to the two sub-layers in each encoder, the decoder (right side of Figure 3.10) inserts a third sub-layer, which performs Multi-Headed Attention over the output of the encoder stack. This layer takes as input the matrix Z^E, output of the last encoder and Z^D, output of the Multi-Headed attention sub-layer in the decoder and computes the Q, K, V matrices as:

$$Q = Z^D W^Q_{enc/dec} \; K = Z^E W^K_{enc/dec} \; V = Z^E W^V_{enc/dec} \tag{3.24}$$

Similarly to the encoder, residual connections around each of the sub-layers, followed by layer normalization, were implemented. The self-attention sub-layer in the decoder was also modified using masking to prevent a word in position i in the output from being influence by words in position $j > i$. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.4.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) [17] is a Transformer-based model designed to be pre-trained using unlabeled text. We can consider BERT as an encoder-only Transformer architecture (left part of Figure 3.10). The BERT pre-train uses two unsupervised tasks, as presented in the left part of Figure 3.11:

- **Masked Language Model (MLM) task**: given a sentence, a certain number of words (also called tokens) are *masked*, which means that they are replaced by another special token. The model is trained to predict these

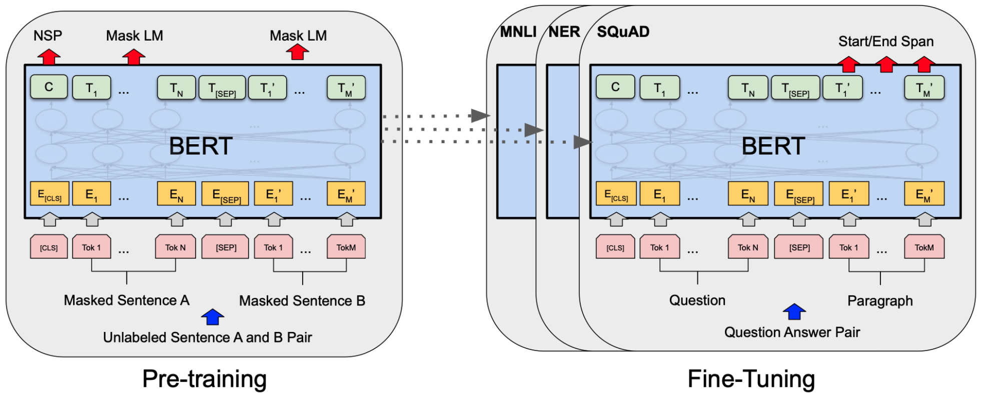


Figure 3.11: Pre-training (on the left) and Fine-Tuning (on the right) of BERT. The model is trained simultaneously on Masked Language Model and Next Sentence Prediction tasks. Once trained it can be fine-tuned on different specific tasks.

masked words using the masked sentence as input and confronting its output with the unmasked input. In [17], 15 % of the words in each sentence is masked during training. This task is used to make the model learn the context of the words in a sentence.

- **Next Sentence Prediction task**: given a couple of sentences A and B, the model is trained to predict whether B follows A in the corpus data. This task is used to make the model learn the relations between sentences, which is particularly useful in context such as Question Answering.

Please note that these two tasks are learned simultaneously, as shown on the left side of Figure 3.11.
Finally, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as Question Answering and Common Sense Inference (right part of Figure 3.11)

Chapter 4

Goal Recognition as a Deep Learning Task

In this chapter, we tackle the goal recognition task in a given domain through a LSTM Network (LSTM). A run of our LSTM processes a trace of observed actions to compute how likely it is that each domain proposition is part of the agent's goal, for the problem instance under considerations. These predictions are then aggregated through a goal selection mechanism to choose one of the candidate goals.

The proposed approach, that we call GRNet, is generally faster than the model-based approach to goal recognition based on planning, since running a trained neural network can be much faster than plan generation. Moreover, GRNet operates with minimal information, since the only information required as input for the trained LSTM is a trace of action labels (each one indicating just the name of an observed action), and the initial state can be completely unknown.

The LSTM of GRNet is trained only once for a given domain, i.e., the same trained network can be used to solve a all goal recognition instances that involve a maximum number of objects. The maximum numbers of objects are considered as hyperparameters.On the other hand, as usual in supervised learning, a (possibly large) dataset of solved goal recognition instances for the domain under consideration is needed for the training. When such data are unavailable or scarce, they can be synthetized via planning. In such a case, the resulting overall system can be seen as a combined approach (model-based for generating training data, and model-free for the goal classification task) that outperforms the pure model-based approach in terms of both classification accuracy and classification runtime. Indeed, an experimental analysis confirms that GRNET achieves good performance obtaining better results with respect to the state-of-the-art goal recognition system LGR [51] for the considered benchmark domains.

Moreover, we propose an effective method to integrate GRNET and LGR in an ensemble fashion, and we experimentally show that this system combination performs consistently better than both GRNET and LGR considered alone showing promising perspectives for integrating learning components and symbolic reasoning in goal recognition.

This chapter describes and extends the work in [15]

4.1 GR Example

As a toy example, we will use a very simple GR instance in the well-known BLOCKSWORLD domain, in which the agent has the goal of building one or more stacks of blocks, and only one block may be moved at a time. There are four types of actions: `Pick-Up` a block from the table, `Put-Down` a block on the table, `Stack` a block on top of another one, and `Unstack` a block that is on another one. In BLOCKSWORLD there are three types of facts (predicates): `On`, that has two blocks as arguments, plus `On-Table` and `Clear` that have one ar-

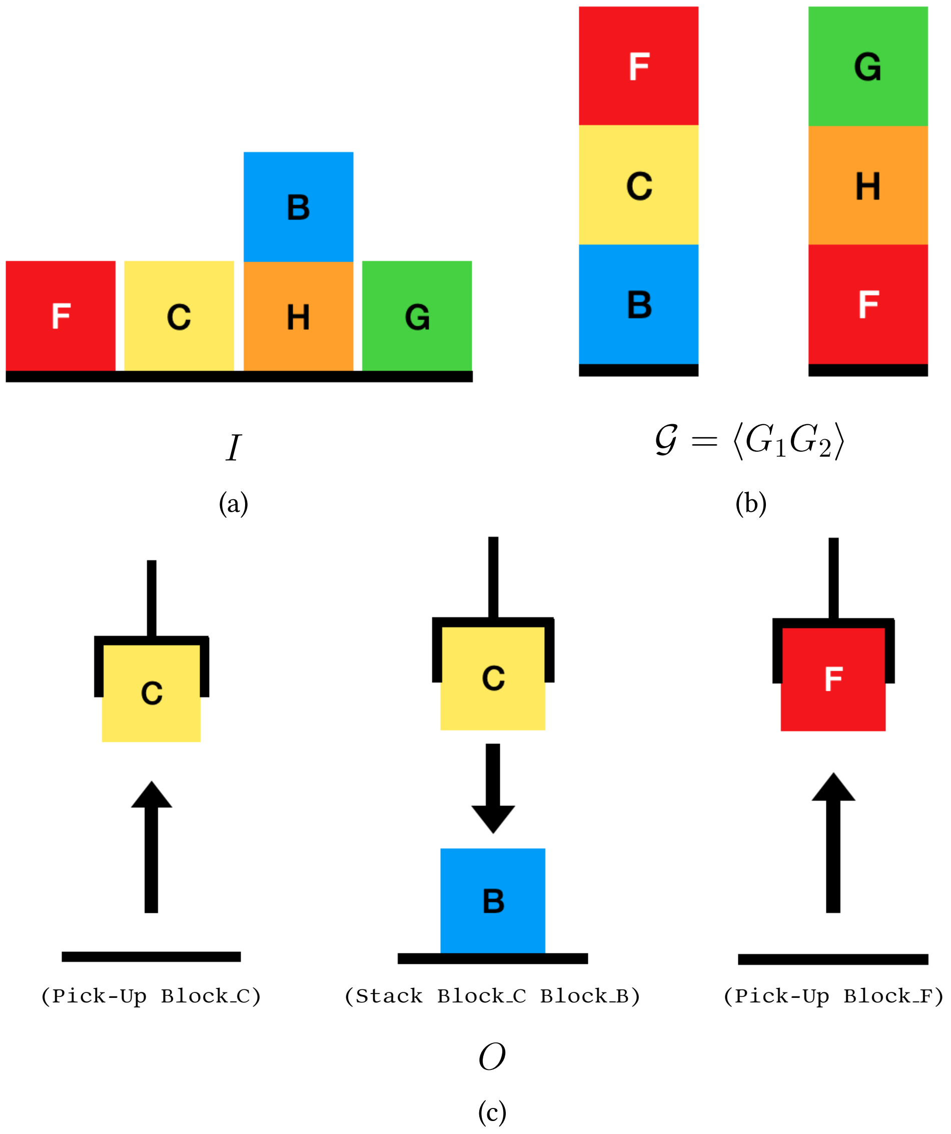


Figure 4.1: Representation of the toy goal recognition instance in the BLOCKSWORLD domain. (a) The initial state I, (b) The goal set $\mathcal{G}$ composed by G_1 (left) and G_2 (right), (c) The sequence of observations $O = \langle$`(Pick-Up Block_C)`, `(Stack Block_C Block_B)`, `(Pick-Up Block_F)`$\rangle$.

gument. A goal recognition instance in the presented domain, composed by an initial state I, an observation sequence O and a goal set $\mathcal{G}$, is illustrated in Figure 4.1. The initial state, reported in Figure 4.1a, is $I = \langle$`(On-Table Block_F)`, `(On-Table Block_C)`, `(On-Table Block_H)`, `(On-Table Block_G)`, `(On Block_B, Block_H)`, `(Clear Block_F)`, `(Clear Block_C)`, `(Clear Block_B)`, `(Clear Block_G)`$\rangle$. In Figure 4.1c, we report the observation sequence $O = \langle$`(Pick-Up Block_C)`, `(Stack Block_C Block_B)`, `(Pick-Up Block_F)`$\rangle$. Finally, the goal set $\mathcal{G}$ of the instance example (Figure 4.1b) consists of the two goals $G_1 = \{$`(On Block_F Block_C)`, `(On Block_C Block_B)`$\}$ and $G_2 = \{$`(On Block_G Block_H)`, `(On Block_H Block_F)`$\}$;

4.2 GRNET

The approach we present to solve goal recognition instances is an alternative approach in which the goal recognition problem is formulated as a classification task, addressed through machine learning, where each candidate goal (a set of propositions) of the problem can be seen as a value class. Our primary aim is making goal recognition more accurate as well as faster by learning how to solve it in a given domain. This approach to goal recognition, called GRNET, is depicted in Figure 4.2. It consists of two main components. The first component takes as input the observations of the GR instance to solve, and gives as output a score (between 0 and 1) for each proposition in the domain proposition set F. This component, called *Environment Component*, is general in the sense that it can be used for every GR instance over F (training is performed once for each domain). The second component, called *Instance Component*, takes as input the proposition ranks generated by the environment component for a GR instance, and uses them

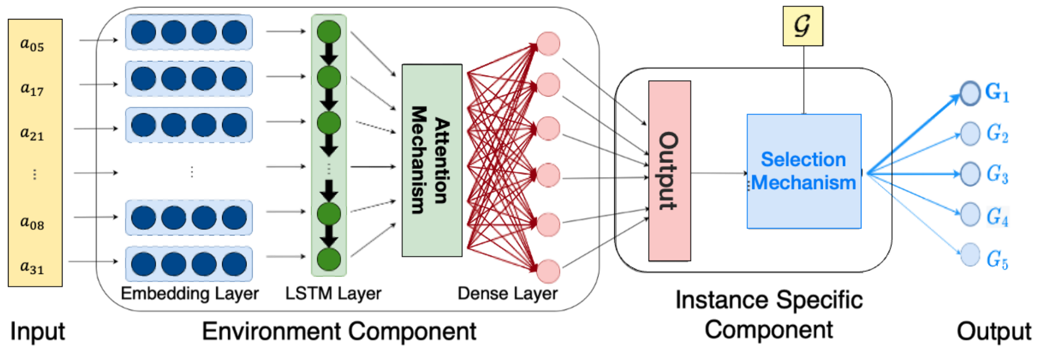


Figure 4.2: Architecture of GRNET. The input observations are encoded by embedding vectors and then fed to a LSTM neural network. After that the attention mechanism computes the context vector, which is used by a feed-forward layer to define the corresponding output values. This layer is composed by $|F|$ neurons, each one representing a possible fluent in the domain. The output of the neural network is then used by the instance component for selecting the goal with the highest score (G_1 in the example of the figure). The observed actions a_{05}, a_{17}, ..., a_{31} are ordered from top to bottom according to their execution order.

to select a goal from the candidate goal set $\mathcal{G}$.

GRNET can be used alone or it can be combined with the MBGR system LGR [51], as shown in the last part of this section.

4.2.1 The Environment Component of GRNet

Given a sequence of observations, represented on the left side of Figure 4.2, each action a_i corresponding to an observation is encoded as a vector e_i of real numbers by an embedding layer [8] [1]. In Figure 4.2, the observed actions are displayed from top to bottom in the order in which they are executed by the agent. The embedding layer is initialised with random weights, and trained at the same time

[1]https://keras.io/api/layers/corelayers/embedding/

with the rest of the environment component.

The index of each observed action is simply the result of an arbitrary order of the actions that is computed in the pre-processing phase, only once for the domain under consideration. Note that two consecutively observed actions a_i and a_j may not be consecutive in the full plan of the agent, which may contain any number of actions in between a_i and a_j.

The Environment Component is based on a Long Short-Term Memory network (LSTM) (see Section 3.2.2), which is especially suitable for processing sequential data like signals or text documents that, in our case, is the sequence of observed actions. The cells that compose the LSTM process each action a in the input sequence O of observed actions. The output of each cell is processed by an Attention Mechanism (see Section 3.3), in particular, the implementation of the Feed-Forward Attention proposed by Yang et al. [67], which computes the weights representing the contribution of each element of the sequence, and generates a unique representation (also called the *context vector*) of the entire plan trace. The context vector is then passed to a feed-forward layer, which has N output neurons with *sigmoid* activation function. N is the number of the domain fluents (propositions) that can appear in any goal of $\mathcal{G}$ for any GR instance in the domain; in this work, N was set to the size of the domain fluent set F, i.e., $N = |F|$. The output of the i-th neuron $\overline{o}_i$ corresponds to the i-th fluent f_i where fluents are lexically ordered, and the activation value of $\overline{o}_i$ gives a rank for f_i being true in the agent's goal (with rank equal to one meaning that f_i is true in the goal). In other words, our network is trained as a multi-label classification problem, where each domain fluent can be considered as a different binary class; each class (fluent) has value 1 is the fluent is present in the goal and 0 otherwise. As loss function, we used standard binary crossentropy.

As shown in Figure 4.2, the dimensions of the input and output of our neural

Hyperparameter	Range
$\|E\|$	[50, 200]
$\|LSTM\|$	[150, 512]
Use Dropout	{True, False}
Dropout	[0, 0.5]
Use Rec. Dropout	{True, False}
Rec. Dropout	[0, 0.5]

Table 4.1: Value ranges of the hyperparameters for the Bayesian-optimisation done by the Optuna framework. $|E|$ represents the dimension of the embedding vectors, $|LSTM|$ is the number of neurons in the LSTM layer. Interval [x, y] indicates a range of integer values from x to y, while set $\{x_1, ..x_n\}$ enumerates all possible values the hyperparameter can assume.

networks depend only on the selected domain and some basic information, such as the maximum number of possible output facts that we want to consider. The dimension of the embedding vectors, the dimension of the LSTM layer and other hyperaparameters of the networks are selected using the Bayesian-optimisation approach provided by the Optuna framework, [1], with a validation set formed by 20% of the training set, while the remaining 80% is used for training the network. More details about the hyperparameters are given in Table 4.1.

4.2.2 The Instance Specific Component of GRNet

After the training and optimisation phases of the environment component, the resulting network can be used to solve any goal recognition instance in the environment through the instance-specific component of our system (right part of

Figure 4.2). Such component performs an evaluation of the candidate goals in $\mathcal{G}$ of the GR instance, using the output of the environment component fed by the observations of the GR instance. To choose the most probable goal in $\mathcal{G}$ (solving the multi-class classification task corresponding to the GR instance), we designed a simple score function that indicates how likely it is that G is the correct goal, according to the neural network of the environment component. This score is defined as

$$S_{\text{GRNET}}(G) = \sum_{f \in G} \overline{o}_f \tag{4.1}$$

where $\overline{o}_f$ is the network output for fact f of the current GR instance. For each candidate goal $G \in \mathcal{G}$, we consider only the output neurons that have associated facts in G. By summing only these predicted values, we derive an overall score for G being the correct goal. The element with the highest score is the most probable goal in $\mathcal{G}$.

4.2.3 Running Example

Considering the goal recognition example reported in Section 4.1, we will show how GRNET processes a goal recognition instance.
Let's assume that a GR instance in this domain involves at most 22 blocks. As a consequence, we have that the action set A is formed by 22 `Pick-Up` actions, 22 `Put-Down` actions, $22 * 21 = 462$ `Stack` actions and 462 `Unstack` actions, for a total of 968 different actions in the domain (i.e. = $|A| = 968$). Considering three types of predicates, `On`, that involves two blocks and `On-Table` and `Clear` that involve one block, we have that the fluent set F consists of $22 \times 21 + 22 + 22 = 506$ propositions. We can suppose that the three observed actions `(Pick-Up Block_C)`, `(Stack Block_C Block_B)` and `(Pick-Up Block_F)` forming the observation sequence O of Figure 4.1 have ids corresponding to indices

5, 17 and 21, respectively. In the Environment Component of GRNet, after being processed by the embedding layer, the input O is represented by the sequence of vectors e_{05}, e_{17} and e_{21} which is fed to the following network layer (see Figure 4.2). Then this sequence is fed to the LSTM layer and subsequently to the attention mechanism, producing a context vector c representing the entire plan trace formed by the observed actions. Finally, the vector c is processed by a final feed-forward layer made of $|F| = 506$ output neurons. In this representation, each neuron corresponds to a distinct proposition in F. Considering two possible goals, G_1, made by `(On Block_F Block_C)` and `(On Block_C Block_B)`, and G_2 made by `(On Block_G Block_H)` and `(On Block_H Block_F)`, if the network has to predict G_1, the neurons associated to the different propositions should have value 1, while the neurons of the propositions in G_2 should have value 0.

Therefore, in the Instance Component of GRNet, the prediction values of G_1 and G_2 is the sum of the predictions for the neurons representing their facts. Suppose that $\overline{o}_{\texttt{(On Block\_F Block\_C)}} = 0.017$, $\overline{o}_{\texttt{(On Block\_C Block\_B)}} = 1.000$, $\overline{o}_{\texttt{(On Block\_G Block\_H)}} = 0.000$, $\overline{o}_{\texttt{(On Block\_H Block\_F)}} = 0.003$, we have that the final score of G_1 is 1.017, while the final score of G_2 is 0.003.

The goal with the highest score (G_1) is selected as the most probable goal solving the GR instance according to the scores of the neural network.

4.3 Integrating GRNet and LGR: LGRNet

GRNet can be integrated with an approach based on plan generation such as the state-of-the-art system LGR, which is described in Section 2.3.2. GRNet and LGR focus on different aspects of the goal recognition problem. While GRNet has the capability of learning from experience the relations among actions and fluents

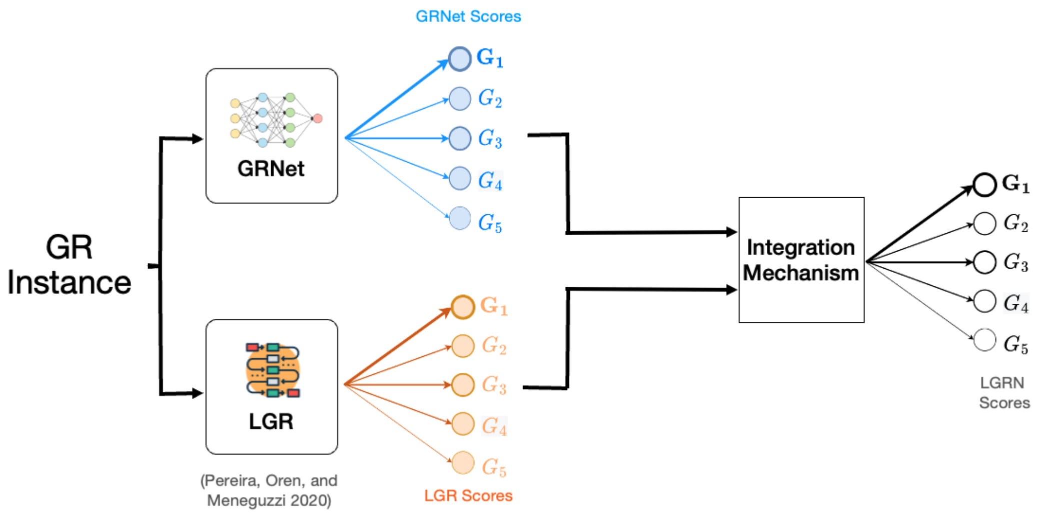


Figure 4.3: Architecture of LGRNet. The input Goal Recognition instance is processed by both GRNet (top) and LGR (bottom). The numerical scores for each goal provided by these systems are combined using the Integration Mechanism, which outputs the aggregated score.

belonging to the goal, LGR exploits domain knowledge and automated reasoning for selecting the most probable goal. Combining these two ways of addressing goal recognition can lead to better accuracy results, overcoming the limits of the automated reasoning approach, especially in the presence of incomplete plan traces, and improving GRNet when the learned experience is inadequate to solve the task. Therefore, we have created an integrated system, called LGRNet, that combines LGR with GRNet.

The integration is simple and effective. Both GRNet and LGR provide a numerical score for each goal in $\mathcal{G}$, and each system considers the goal with the highest score as the most probable one. Therefore, we can combine them by using an aggregated score defined as the normalized sum of the scores of the two individual systems. More formally, given a GR instance, for each candidate goal

$G \in \mathcal{G}$, the score $S_{\text{LGRNet}}(G)$ computed by LGRNet for G is:

$$\sigma([S_{\text{LGR}}(G_i)|G_i \in \mathcal{G}])_G + \sigma([S_{\text{GRNet}}(G_i)|G_i \in \mathcal{G}])_G \tag{4.2}$$

where $S_{\text{LGR}}(G)$ is the score of LGR for G, and $\sigma([\cdot])_G$ is the output for G of the softmax function applied to the input score vector $[\cdot]$. The softmax normalisation is chosen in order to have the same value distribution space for the two scores, i.e., values in the range $[0, 1]$ with sum 1. Furthermore, the softmax function tends to flatten the score values, which helps to avoid cases where the sum is extremely biased towards one approach or the other.

4.3.1 Running Example

Carrying on the goal recognition example shown in Sections 4.1, we will show how LGRNet processes a goal recognition instance. In the first step, the goal recognition instance is solved by both LGR and GRNet. For the overall working of LGR, please see Section 2.3.2.

The output of LGR containing the computed landmarks, their uniqueness scores ($\mathcal{L}_{uniq}$) and the heuristic values (h_{uniq}) for each goal are reported in Figure 4.4. As we can see, there are 6 landmarks achieved in the observations over a total of 7 landmarks for G_1 and 2 achieved landmarks over a total of 6 for G_2. Among the total landmarks there is only one that belongs both to G_1 and to G_2; this landmark obtains a uniqueness score of 0.5 while all the others get a uniqueness score of 1. As a consequence, the heuristic value (i.e. the final score) of G_1 is 0.85 while the heuristic value of G_2 is 0.28. As described in Section 4.2.3, the final scores computed by GRNet when processing this simple goal recognition example are 1.017 for G_1 and 0.003 for G_2.

Therefore, the Integration Mechanism of LGRNet combines these two scores by computing the sum on the normalized scores. The normalization process is

```
Computing achieved landmarks
- Goal: (and (on f c)  (on c b) )
  Ordered Landmarks:
  * on f c -> [[arm-empty, on-table f, clear f], [holding f, clear c], [on f c]]
  * on c b -> [[on-table c, arm-empty, clear c], [on h b, clear h, arm-empty],
              [clear b, holding c], [on c b]]
  Total number of Landmarks: 7
  Achieved Landmarks in Observations [6]:
    [[on-table c, arm-empty, clear c], [holding f, clear c], [on c b],
    [arm-empty, on-table f, clear f], [on h b, clear h, arm-empty],
    [clear b, holding c]]
- Goal: (and (on h f)  (on g h) )
  Ordered Landmarks:
  * on h f -> [[on h b, clear h, arm-empty], [holding h, clear f], [on h f]]
  * on g h -> [[arm-empty, clear g, on-table g], [holding g, clear h], [on g h]]
  Total number of Landmarks: 6
  Achieved Landmarks in Observations [2]:
    [[arm-empty, clear g, on-table g], [on h b, clear h, arm-empty]]
Landmark Uniqueness Heuristic
- Goal: (and (on f c)  (on c b) )
    * [on-table c, arm-empty, clear c] = (1.0)
    * [holding f, clear c] = (1.0)
    * [on c b] = (1.0)
    * [arm-empty, on-table f, clear f] = (1.0)
    * [on h b, clear h, arm-empty] = (0.5)
    * [clear b, holding c] = (1.0)
  Heuristic Value = 5.5 / 6.5 = 0.85
- Goal: (and (on h f)  (on g h) )
  * [arm-empty, clear g, on-table g] = (1.0)
  * [on h b, clear h, arm-empty] = (0.5)
  Heuristic Value = 1.5 / 5.5 = 0.28
```

Figure 4.4: Solution provided by LGR with h_{uniq} heuristic for solving the toy goal recognition instance reported in Section 4.1.

obtained by applying the softmax function on the original scores. Hence, given that the normalized scores of LGR are 0.639 for G_1 and 0.361 for G_2 and the nomalized scores for GRNet are 0.734 for G_1 and 0.266 for G_2, we have that the final score for G_1, following Equation 4.2, is 1.373 while the final score G_2 is 0.627.

The goal with the highest combined score (G_1) is selected as the most probable goal solving the GR instance accordind to the scores of both LGR and GRNet.

4.4 Benchmark Suite and Data Sets

4.4.1 Domains

We consider six well-known benchmark domains [43, 40]:

- **BLOCKSWORLD**. The domain consists of an hand robot that has to stack or unstack blocks, picking up them one at a time, in order to obtain a desired configuration of an available set of blocks.

- **DEPOTS**. The domain consists of actions for loading and unloading packages into trucks through hoists, and moving them between depots. The goals concern having the packages at certain depots.

- **DRIVELOG**. In this domain there are drivers (trucks) that can walk (drive) between locations. Walking and driving require traversing different paths. Packages can be loaded into or unloaded from trucks, that can be moved by drivers. The goal is to deliver (move) all packages (drivers) to their destinations.

- **LOGISTICS**. In this domain there are aircrafts that can fly between cities, trucks that can move between locations within a city, and packages that can be loaded into/unloaded from trucks and aircrafts. The goal is to deliver a set of packages to their delivery locations.

- **SATELLITE**. This is a scheduling domain in which one or more satellites can make certain observations, collect data, and download the collected data to a ground station. The goals concern having observation data at a ground station.

- **ZENOTRAVEL**. This is another variant of a transportation domain where passengers have to be embarked and disembarked into aircrafts, that can fly between cities at two possible speeds. The goals concern transporting (move) all passengers (aircrafts) to their required destinations.

Of course GRNET can be trained and tested also using other domains.

4.4.2 Training sets

In order to create the (solved) GR instances for the training and test sets in the considered domains, we used automated planning techniques. Concerning the training set, for each domain, we randomly generated a large collection of (solvable) plan generation problems of different size. We considered the same ranges of the numbers of involved objects as in the experiments of Pereira et al. [51]. For each of these problems, we computed up to four (sub-optimal) plans solving them. As planner we used LPG [26, 27], which allows to specify the number of requested different solutions for the planning problem it solves. From the generated plans, we derived the observation sequences for the training samples by randomly selecting actions from the plans (preserving their relative order). The selected actions are between 30% and 70% of the plan actions. However, the test sets described next also include GR instances with lower and higher percentages of observed actions.

The generated training set consists of pairs (O, G^*) where O is a sequence of observed actions obtained by sampling a plan π, and G^* is the hidden goal corresponding to the goal of the planning problem solved by π. For each considered domain, we created a training set with 55000 pairs.

4.4.3 Test sets

Domain	A	F	G_i		$\mathcal{G}$		*Object*			
			min	max	min	max	Name	min	max	Total
BLOCKSWORLD	968	506	4	16	19	21	block	7	17	22
DEPOTS	13050	150	2	8	7	10	depot	1	3	3
							distributor	1	3	3
							truck	2	3	3
							pallet	2	6	6
							crate	2	10	10
							hoist	2	6	6
DRIVELOG	4860	156	4	11	6	10	driver	2	3	3
							truck	2	3	3
							package	2	7	7
							locations$_s$	3	12	12
							locations$_p$	2	25	41
LOGISTICS	15154	154	2	4	10	12	airplane	1	8	8
							airport	2	8	8
							location	6	11	11
							city	2	6	6
							truck	2	5	5

Domain	A	F	G_i		$\mathcal{G}$		*Object*			
			min	max	min	max	Name	min	max	Total
							package	2	14	14
SATELLITE	33225	629	4	9	6	8	satellite	1	5	5
							instrument	1	11	11
							mode	3	5	12
							direction	7	17	37
ZENOTRAVEL	23724	66	5	9	6	11	aircraft	2	3	3
							person	5	8	8
							city	3	6	6
							flevel	7	7	7

Table 4.2: Size of A, F, $G_i \in \mathcal{G}$ and $\mathcal{G}$ and number of objects involved in the considered GR instances for each considered domain. min and max columns indicate a range of values from min to max. Column Total represents the total number of objects of that type in the domain.

For evaluating GRNET, we generated two test sets formed by GR instances *not seen at training time*: $\text{TS}_{\text{LGR}\mathit{Gen}}$ and TS_{Rec}. Such test instances were generated as for the training instances, except that the observation sequences were derived from plans computed by LAMA [57], while for the training instances we used plans computed by LPG; this change was made in order to make the testing more robust.

$\text{TS}_{\text{LGR}\mathit{Gen}}$ is a generalisation and extension of the test set used in [51] for the same domains that we consider. We indicate the test set provided by Pereira et al. [51] with TS_{LGR}. $\text{TS}_{\text{LGR}\mathit{Gen}}$ includes all TS_{LGR} instances; moreover, the goal sets ($\mathcal{G}$) of TS_{LGR} and $\text{TS}_{\text{LGR}\mathit{Gen}}$ are the same. The additional instances in $\text{TS}_{\text{LGR}\mathit{Gen}}$ are motivated by the limited number and particular structure of those in TS_{LGR}. The structure is limited because the observations in the instances of TS_{LGR} are created from plans for the goals in $\mathcal{G}$ that are all generated from the *same* initial state. In $\text{TS}_{\text{LGR}\mathit{Gen}}$, the GR instances are created combining different initial states with the candidate goal sets, obtaining a richer diversification of the observation traces and a larger number of test instances. In particular, for each of DEPOTS, DRIVELOG, SATELLITE and ZENOTRAVEL, TS_{LGR} contains only 84 instances, while $\text{TS}_{\text{LGR}\mathit{Gen}}$ contains 1000 instances for each domain.

For each plan generated for being sampled, we randomly derived five different action traces formed by 10%, 30%, 50%, 70% and 100% of the plan actions, respectively. This gives five groups of test instances, for each considered domain, allowing to evaluate the performance of GRNET also in terms of different amounts of available observations.

Table 4.2 gives information about the size of the GR instances in our test and training sets for each domain, in terms of number of possible actions ($|A|$), facts ($|F|$), min/max size of a goal ($|G_i|$) in a goal set $\mathcal{G}$, min/max size of a goal set ($|\mathcal{G}|$), the names of the objects involved in the GR instances, the min/max number of objects of each type that are involved in an instance and the total number of objects of each type among all the instances (*Object* column). Please note that column *Total* reports the number of all possible objects of a certain type that can be used when defining a GR instance while column Object max indicates the maximum number of objects of a certain type that are used in a GR instance.

Test set TS_{Rec} was created to evaluate how well the compared systems behave on GR instances of different difficulty. We focus this analysis on a specific domain (ZENOTRAVEL). In TS_{Rec}, the generated GR instances are grouped into several classes according to their difficulty. As difficulty measure, we used the notion of *recognizability of the hidden goal*, which is inspired by the notion of the Uniqueness Value (Equation 2.3). Specifically, the recognizability $R(G)$ of a goal $G \in \mathcal{G}$ is defined as

$$R(G) = \sum_{f \in G} \frac{1}{|\{G' \mid G' \in \mathcal{G} \wedge f \in G'\}|} \tag{4.3}$$

The lower $R(G)$ is, the more difficult recognising G is; vice versa, the higher $R(G)$ is, the more discernible G is. We normalize $R(G)$ as a value between 0 and 1, denoted $R_Z(G)$. E.g., if $\mathcal{G} = \{G_1, G_2, G_3\}$, with $G^* = G_1 = \{a, b, c\}$, $G_2 = \{a, e, f\}$ and

$G_3 = \{g, h, i\}$, then $R(G^*) = \frac{1}{2} + 1 + 1 = \frac{5}{2}$ and $R_Z(G^*) = 0.75$ (high recognizability). If $\mathcal{G} = \{G_1, G_2, G_3\}$ with $G^* = G_1 = \{a, b, c\}$, $G_2 = \{a, b, x\}$ and $G_3 = \{a, b, y\}$, then $R(G^*) = \frac{1}{3} + \frac{1}{3} + 1 = \frac{5}{3}$, and so $R_Z(G^*) = 0.33$ (low recognizability).

Using different values for $R_Z(G^*)$, we generated nine classes of GR instances, denoted $C_1, ..., C_9$. For each GR instance in class C_i, we have $0.1 \cdot i \leq R_Z(G^*) < 0.1 \cdot (i + 1)$, for $i = 1...9$. Each class consists of 100 GR instances.

4.4.4 Evaluation measures

We use the GR *accuracy* for a set of test instances as the main evaluation criteria, which is defined as the percentage of instances whose goals are correctly identified over the total number of instances in the test set. If for an instance the evaluated system provides k different goals with the same highest score, then, in the overall count of the solved instances, this instance has value $\frac{1}{k}$ if the true goal is one of these k goals, 0 otherwise.

Following the methodology in [51], we also analyze the GR *θ-accuracy*. This measure assumes that the scores assigned to the goals by the GR system are values between 0 and 1, and uses a θ threshold to select the set of candidate goals whose score is greater than or equal to the highest assigned score minus θ. If the true goal belongs to the set of selected goals, the instance is considered correctly identified, and this instance obtains θ-accuracy 1 (otherwise θ-accuracy is 0). For evaluating the θ-accuracy for GRNet and LGRNet, their prediction scores (positive numbers) are scaled using the min-max normalization which sets each score to a value in the $[0, 1]$ range. For $\theta = 0$ the system selects the $k \geq 1$ goals with the highest score, and if the true goal belongs to this set, the instance is considered solved obtaining value 1, instead of $\frac{1}{k}$ as in the standard accuracy metric. The θ-accuracy is analysed together with the *Spread* measure, i.e., the average number of predicted goals for an instance according to the used θ threshold.

Domain	$\|E\|$	$\|LSTM\|$	Dropout	Recurrent Dropout
BLOCKSWORLD	119	354	0.00	0.00
DEPOTS	200	450	0.15	0.23
DRIVELOG	183	473	0.00	0.00
LOGISTICS	85	446	0.12	0.01
SATELLITE	117	496	0.04	0.00
ZENOTRAVEL	83	350	0.00	0.00

Table 4.3: Hyperparameters of the neural networks used in our experiments. $|E|$ is the size of the vector in output from the embedding layer, $|LSTM|$ is the number of neurons in the LSTM layer.

4.5 Experimental Results

We experimentally evaluate GRNET and LGRNET, and we compare them with the state-of-the-art system LGR. For the goal selection in LGR we used heuristic h_{uniq} because the authors stated that it performs better than heuristic h_{gc}.

4.5.1 Hyperparameters of the Neural Networks

We used the Optuna framework [1] to find the best hyperparameters for out networks. For each benchmark domain, we used a separate process of optimization (study) which executes 30 objective function evaluations (trials). We used a sampler that implements the Tree-structured Parzen Estimator algorithm.

Table 4.3 reports the hyperparameters of the neural networks in our experiments. For all the experiments we selected a batch size of 64 elements; we used Adam as optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

4.5.2 Accuracy Results for $\text{TS}_{\text{LGR}Gen}$

Table 4.4 summarizes the performance results of LGR, GRNet and LGRNet in terms of GR accuracy using $\text{TS}_{\text{LGR}Gen}$. Bold results for GRNet and LGRNet indicate better performance w.r.t. to LGR. GRNet and LGRNet perform generally well, and they improve their performances with the increase of the percentage of the observed actions. With 30% of the actions, the accuracy of GRNet improves w.r.t. 10% of the action in every domain by more than 20 points. For instance, in drivelog, GRNet improves from 39.8 to 65.4. Similar improvements can be observed considering 50%, 70% and 100% of the actions. E.g., with 70% of the observed actions, the accuracy of GRNet is higher than 96 in satellite and zenotravel. GRNet always obtains higher accuracy w.r.t. LGR except in depots with 100% of the observations, and in many cases the performance improvement is of several points (e.g., more than 12 points for blocksworld, drivelog, logistics and zenotravel with 30% of the actions).

Regarding the accuracy of LGRNet, it always performs better than both LGR and GRNet. Moreover, in several domains, we can see a remarkable improvement w.r.t. both LGR and GRNet, especially with 30% and 50% of the actions. For instance, in depots the accuracy of LGRNet for 30% of actions is almost 72, 24 points better than LGR and 12 points better than GRNet. With 100% of the actions, the accuracy scores of GRNet and LGRNet are higher than or equal to 90%, and still generally better than (especially for LGRNet) the accuracy scores of LGR.

Moreover, GRNet's performance does not seem to be affected by the diversity of the domains indicated by the parameters of Table 4.2. While the remarkable performance obtained for zenotravel might be correlated with the fact that in this domain the test instances have only 66 facts (see column $|F|$ of Table 4.2), the results for satellite are not so distant even if the instances in this domain have 629 facts. Analysing the experimental results, it seems that also the number of the actions has no significant impact on the performance. In fact, while blocksworld has only 968 actions, the other domains have more than 15000 actions, and GRNet obtains better results for them. This

Plan %	*Models*	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
	LGR	20.39	25.42	25.43	27.20	41.26	28.37
10	GRNET	**22.85**	**32.95**	**39.80**	**39.15**	**45.50**	**48.00**
	LGRNET	**30.80**	**40.30**	**44.50**	**43.60**	**58.70**	**58.00**
	LGR	38.92	47.52	41.27	55.50	73.95	49.70
30	GRNET	**52.35**	**59.80**	**65.40**	**68.80**	**75.10**	**76.90**
	LGRNET	**63.70**	**71.90**	**72.70**	**77.60**	**85.50**	**85.70**
	LGR	53.02	65.88	59.07	73.93	84.35	69.90
50	GRNET	**71.10**	**74.70**	**78.40**	**80.40**	**88.30**	**89.20**
	LGRNET	**79.30**	**87.60**	**82.90**	**90.90**	**94.50**	**94.80**
	LGR	72.25	78.78	76.32	85.06	92.34	88.18
70	GRNET	**84.90**	**84.90**	**86.20**	**89.20**	**96.10**	**96.80**
	LGRNET	**90.60**	**93.60**	**91.10**	**97.10**	**97.80**	**98.80**
	LGR	88.62	93.17	89.94	90.14	96.44	98.71
100	GRNET	**92.02**	91.95	**90.78**	**93.94**	**98.73**	**98.73**
	LGRNET	**96.81**	**97.25**	**94.28**	**99.23**	**99.36**	**99.58**

Table 4.4: Goal recognition accuracy (% of GR instances correctly predicted) by LGR, GRNET and LGRNET with test set $\mathrm{TS}_{\mathrm{LGR}Gen}$. Results for GRNET and LGRNET are in bold when they are better than the corresponding results for LGR.

Domain	plan %	LGR						GRNet						LGRNet					
		θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)			θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)			θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)		
		0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2
	10	21.7	43.8	64.2	1.07	3.65	7.94	**23.0**	**56.9**	**65.9**	**1.00**	**3.39**	**4.46**	**31.2**	**53.4**	**68.3**	**1.00**	**2.49**	**4.51**
	30	40.4	63.3	77.9	1.07	3.07	6.43	**52.5**	**79.5**	**86.5**	**1.01**	**2.26**	**3.00**	**65.8**	**78.6**	**87.5**	**1.00**	**1.56**	**2.28**
BLOCKS	50	56.1	77.3	88.7	1.10	2.69	5.60	**71.3**	**90.3**	**94.3**	**1.00**	**1.89**	**2.55**	**80.2**	**88.3**	**93.3**	**1.00**	**1.28**	**1.66**
(1000)	70	75.9	89.8	96.2	1.11	2.14	4.40	**85.2**	**96.2**	**98.3**	**1.01**	**1.65**	**2.24**	**91.4**	**95.7**	**98.3**	**1.01**	**1.15**	**1.33**
	100	98.5	100.0	100.0	1.25	1.66	3.19	92.2	99.9	99.9	**1.00**	**1.46**	**1.96**	**98.6**	99.5	100.0	**1.00**	**1.05**	**1.13**
	10	27.2	47.3	69.6	1.15	2.49	4.56	**33.0**	**50.5**	56.4	**1.00**	**1.78**	**2.19**	**40.9**	**57.4**	**73.3**	**1.00**	**1.87**	**3.07**
	30	49.3	67.0	82.3	1.10	2.10	3.79	**59.8**	**72.6**	79.0	**1.00**	**1.45**	**1.73**	**72.8**	**81.4**	**86.9**	**1.00**	**1.34**	**1.72**
DEPOTS	50	68.0	80.8	92.1	1.07	1.78	3.08	**74.7**	**87.4**	90.9	**1.00**	**1.36**	**1.54**	**88.9**	**92.3**	**94.7**	**1.00**	**1.12**	**1.28**
(1000)	70	80.8	90.9	97.2	1.05	1.52	2.43	**84.9**	**92.4**	95.0	**1.00**	**1.25**	**1.38**	**94.2**	**96.4**	**97.7**	**1.00**	**1.05**	**1.13**
	100	96.2	99.2	99.8	1.06	1.30	1.63	91.9	96.6	98.2	**1.00**	**1.16**	**1.24**	**97.7**	98.1	98.5	**1.00**	**1.01**	**1.04**
	10	26.9	46.1	67.6	1.06	2.12	3.61	**39.8**	**58.4**	**71.4**	**1.00**	**1.72**	**2.40**	**45.6**	**54.2**	65.6	**1.00**	**1.38**	**1.90**
	30	43.1	64.1	80.8	1.07	1.97	3.23	**65.4**	**82.5**	**89.8**	**1.00**	**1.61**	**2.23**	**73.8**	**79.8**	**85.4**	**1.00**	**1.18**	**1.45**
DRIVELOG	50	61.9	76.8	91.3	1.09	1.77	2.77	**78.4**	**90.5**	**96.4**	**1.00**	**1.45**	**1.96**	**84.3**	**88.6**	**91.6**	**1.00**	**1.11**	**1.24**
(1000)	70	80.3	90.3	96.3	1.11	1.58	2.30	**86.2**	**95.2**	**98.4**	**1.00**	**1.37**	**1.75**	**92.3**	**94.9**	**97.1**	**1.00**	**1.06**	**1.17**
	100	97.2	99.0	99.6	1.18	1.32	1.78	90.8	98.0	99.3	**1.00**	**1.27**	**1.57**	95.7	97.4	98.4	**1.00**	**1.04**	**1.10**
	10	29.0	47.3	65.6	1.10	2.44	4.58	**39.2**	**57.1**	63.8	**1.00**	**1.66**	**2.06**	**44.2**	**64.1**	**79.7**	**1.00**	**2.02**	**3.70**
	30	57.4	71.5	82.3	1.07	1.71	2.82	**68.8**	**81.6**	**85.3**	**1.00**	**1.54**	**1.75**	**77.7**	**87.9**	**93.9**	**1.00**	**1.31**	**1.78**
LOGISTICS	50	75.8	84.5	92.2	1.06	1.41	1.94	**80.4**	**91.0**	**92.6**	**1.00**	1.43	**1.58**	**90.9**	**95.3**	**97.9**	**1.00**	**1.14**	**1.33**
(1000)	70	89.0	93.7	98.0	1.11	1.25	1.50	**89.2**	**96.5**	98.0	**1.00**	1.32	**1.45**	**97.3**	**98.8**	**99.9**	**1.00**	**1.05**	**1.11**
	100	99.6	100.0	100.0	1.22	1.24	1.31	93.9	99.8	100.0	**1.00**	**1.18**	**1.24**	99.2	100.0	100.0	**1.00**	**1.01**	**1.02**
	10	47.4	84.5	97.7	1.27	3.25	5.27	45.5	68.9	81.1	**1.00**	**1.92**	**2.52**	**60.1**	83.5	94.2	**1.11**	**2.14**	**3.18**
	30	78.4	92.2	98.4	1.20	1.96	3.27	75.1	91.4	95.8	**1.00**	**1.56**	**1.82**	**87.1**	**94.0**	97.6	**1.11**	**1.34**	**1.61**
SATELLITE	50	87.9	95.7	98.8	1.16	1.52	2.20	**88.3**	**97.0**	98.6	**1.00**	**1.26**	**1.40**	**95.5**	**97.5**	**98.9**	**1.11**	**1.18**	**1.26**
(1000)	70	95.3	98.4	99.7	1.14	1.29	1.59	**96.1**	**99.4**	99.7	**1.00**	**1.13**	**1.23**	**97.9**	**99.0**	99.4	**1.11**	**1.14**	**1.17**
	100	98.5	99.6	99.8	1.12	1.19	1.37	**98.7**	99.6	99.8	**1.00**	**1.08**	**1.19**	**99.7**	**99.8**	99.8	**1.12**	**1.12**	**1.13**
	10	29.6	46.1	63.1	1.05	1.89	3.04	**48.0**	**71.7**	**89.6**	**1.00**	1.97	**2.98**	**59.4**	**70.0**	**80.8**	**1.00**	**1.43**	**2.00**
	30	50.5	67.4	80.0	1.03	1.74	2.65	**76.9**	**88.0**	**94.7**	**1.00**	**1.32**	**1.65**	**87.5**	**91.0**	**94.3**	**1.00**	**1.12**	**1.28**
ZENO	50	70.5	83.9	92.1	1.02	1.53	2.24	**89.2**	**94.7**	**97.7**	**1.00**	**1.15**	**1.27**	**95.7**	**97.4**	**98.6**	**1.00**	**1.04**	**1.09**
(1000)	70	88.6	94.4	98.8	1.01	1.27	1.64	**96.8**	**99.0**	**99.6**	**1.00**	**1.05**	**1.10**	**99.4**	**99.5**	**99.8**	**1.00**	**1.00**	**1.02**
	100	99.8	99.9	100.0	1.02	1.04	1.15	98.7	99.4	99.8	**1.00**	**1.01**	**1.04**	99.8	99.8	99.9	**1.00**	**1.00**	**1.01**

Table 4.5: θ-accuracy and spread of LGR, GRNet and LGRNet for test set $\text{TS}_{\text{LGR}Gen}$. Results for GRNet and LGRNet are in bold when they are better than the corresponding results for LGR.

is probably due to the embedding layer that is able to learn a compact and informative representation even with a large vocabulary of actions. Overall, GRNet exhibits good robustness with respect to the size of the space of actions and the number of facts in the domains (the output of the network).

4.5.3 θ-accuracy results for $\text{TS}_{\text{LGR}\textit{Gen}}$

Table 4.5 compares GRNet and LGRNet with LGR in terms of θ-accuracy and the corresponding spread in $\mathcal{G}$. Considering θ equal to either 0 or 0.1 and partial plan traces (from 10% to 70% of observed actions), GRNet obtains a better θ-accuracy w.r.t. LGR in 44 out of 48 configurations, while LGRNet has better performance in 47 out of 48 configurations. In several cases, the improvement is by several points, such as in DRIVELOG with 10% of the actions. With $\theta = 0.2$, overall GRNet and LGRNet perform better than LGR. There are two notable exceptions, DEPOTS and SATELLITE. In DEPOTS, LGR obtains a higher θ-accuracy w.r.t. GRNet but not w.r.t. LGRNet. However, this result should be analysed also in terms of spread in $\mathcal{G}$. LGR has a considerably higher spread than GRNet, especially considering low percentages of actions. According to the definition of θ-accuracy, an instance is considered correctly solved if the true goal belongs to the selected *set* of goals, and so a higher spread can lead to a higher θ-accuracy. The same can be said for SATELLITE with $\theta = 0.2$, in which GRNet and LGRNet have considerably lower spreads and worse θ-accuracy (but in most cases performing similarly).

We can observe that GRNet and LGRNet have lower spreads in all but three considered configurations. In particular, with $\theta = 0$ the spread of GRNet is always 1. With $\theta > 0$, especially when considering low percentages of actions, we have a remarkable improvement in terms of spread w.r.t. LGR alongside an improvement in terms of θ-accuracy (see, e.g., DRIVELOG and ZENOTRAVEL with 30% of the actions).

Concerning instances with complete plan traces (100% of the actions), in terms of θ-accuracy, all three systems obtain very good results, in many cases close to 100%. Although for these cases LGR often has better θ-accuracy, considering also the lower spreads of GRNet and LGRNet, we think that overall the results for the full plan traces are comparable.

In term of CPU time to solve (classify) a GR instance, GRNet is generally much faster than LGR. The average execution time of LGR is 1.158 seconds with a standard deviation of 0.87 seconds, while GRNet runs on average in 0.06 seconds with a standard deviation of 0.04 seconds.

4.5.4 Results for TS_{LGR}

While we consider the evaluation using the extended set $\text{TS}_{\text{LGR}Gen}$ more significant and informative than using the restricted test set TS_{LGR}, we compared LGR, GRNet and LGRNet also on TS_{LGR}. As shown in Table 4.6 and Table 4.7, in terms of accuracy, θ-accuracy and spread, the results are still in favor of GRNet compared with LGR, and substantially better for LGRNet compared with LGR. However, this is not the case for domain satellite with test instances that have 70% of the actions. In this case LGR reaches accuracy 93.4, while GRNet and LGRNet have accuracy 84.5 and 88.1, respectively. Most of the errors made by GRNet are due to the restricted and particular set of instances in TS_{LGR}, which has instances with very similar goals in $\mathcal{G}$. These are not clearly distinguished by GRNet, making LGRNet less effective in such cases.

4.5.5 Results for TS_{Rec} and sensitiveness to the training set size

Figure 4.5 shows the accuracy of the two compared systems considering different classes of test sets with decreasing difficulty measured using R_Z. We focus this analysis on instances with 30-50-70% of observed actions. As expected, the accuracy of GRNet depends on the difficulty of the problem, since there is an increasing trend in terms of accuracy for each observation percentage. This trend is more evident when we have 30% of the actions and becomes less marked as the number of observations grows. LGR

Plan %	*Models*	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
10	LGR	30.08	32.14	34.23	43.25	42.06	34.52
	GRNET	18.90	**41.67**	**42.86**	**46.41**	**42.86**	**44.05**
	LGRNET	26.83	29.76	**46.43**	**50.98**	**44.05**	**53.57**
30	LGR	49.59	45.63	42.26	70.92	66.27	60.71
	GRNET	48.58	**66.67**	**63.10**	**75.16**	64.29	**67.86**
	LGRNET	**58.13**	**67.86**	**65.48**	**75.16**	**72.62**	**73.81**
50	LGR	52.91	71.43	62.30	84.64	79.76	76.19
	GRNET	**69.92**	**83.33**	**77.38**	**88.89**	79.76	**84.52**
	LGRNET	**71.54**	**88.10**	**78.57**	**92.81**	**85.71**	**86.90**
70	LGR	72.09	83.93	83.93	94.44	93.45	90.48
	GRNET	**86.59**	**84.52**	**84.52**	**96.08**	84.52	**97.62**
	LGRNET	**88.62**	**91.67**	**86.90**	**95.42**	88.10	**98.81**
100	LGR	85.96	98.21	92.26	100.00	96.43	100.00
	GRNET	**92.93**	92.86	**92.86**	95.08	96.43	100.00
	LGRNET	**95.65**	**100.00**	**92.86**	100.00	96.43	100.00

Table 4.6: Goal recognition accuracy (% of GR instances correctly predicted) by LGR, GRNET and LGRNET with test set TS_{LGR}. Results for GRNET and LGRNET are in bold when they are better than the corresponding results for LGR.

Domain	plan %	LGR						GRNet						LGRNet					
		θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)			θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)			θ-Acc (↑)			Spread in $\mathcal{G}$ (↓)		
		0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2	0	0.1	0.2
	10	30.9	52.4	67.5	1.04	2.77	6.12	19.5	50.8	58.5	**1.02**	2.78	**3.75**	25.6	43.5	60.6	**1.01**	**2.11**	**3.62**
	30	51.2	65.9	77.6	1.07	2.50	5.02	49.2	**76.0**	**85.8**	**1.03**	**2.10**	**2.73**	**61.8**	**72.8**	**79.7**	**1.01**	**1.46**	**1.98**
BLOCKS	50	56.1	73.6	84.6	1.10	2.31	4.51	**70.7**	**88.2**	**93.1**	**1.02**	**1.83**	**2.55**	**71.5**	**80.9**	**85.4**	**1.02**	**1.24**	**1.68**
(246)	70	77.2	89.0	95.9	1.15	2.00	3.53	**87.8**	**96.7**	**98.4**	**1.02**	**1.65**	**2.10**	**89.8**	**93.9**	95.9	**1.02**	**1.17**	**1.35**
	100	100.0	100.0	100.0	1.38	1.60	2.61	94.6	100.0	100.0	**1.03**	**1.45**	**1.76**	98.9	100.0	100.0	**1.03**	**1.10**	**1.20**
	10	32.1	51.2	75.0	1.10	2.74	5.42	**41.7**	**56.0**	63.1	**1.00**	**1.73**	**2.24**	28.6	48.8	**76.2**	**1.00**	**2.14**	**3.68**
	30	47.6	70.2	94.0	1.08	2.49	4.83	**66.7**	**79.8**	89.3	**1.00**	**1.51**	**1.81**	**64.3**	**78.6**	88.1	**1.00**	**1.52**	**1.98**
DEPOTS	50	72.6	84.5	96.4	1.04	2.11	3.67	**83.3**	**95.2**	95.2	**1.00**	**1.39**	**1.56**	**88.1**	**94.0**	94.0	**1.00**	**1.21**	**1.38**
(84)	70	84.5	95.2	96.4	1.01	1.60	2.82	84.5	94.0	96.4	**1.00**	**1.21**	**1.39**	**94.0**	**96.4**	**97.6**	**1.00**	**1.06**	**1.19**
	100	100.0	100.0	100.0	1.04	1.29	1.75	92.9	100.0	100.0	**1.00**	**1.18**	**1.18**	100.0	100.0	100.0	**1.00**	**1.00**	**1.04**
	10	36.9	56.0	79.8	1.11	2.35	4.29	**42.9**	**64.3**	73.8	**1.00**	**1.99**	**2.52**	**46.4**	54.8	65.5	**1.00**	**1.30**	**1.90**
	30	45.2	69.0	83.3	1.08	2.01	3.40	**63.1**	**81.0**	**86.9**	**1.00**	**1.73**	**2.43**	**64.3**	**72.6**	78.6	**1.00**	**1.26**	**1.58**
DRIVELOG	50	67.9	82.1	94.0	1.13	1.75	2.68	**77.4**	**88.1**	**96.4**	**1.00**	**1.38**	**1.86**	**77.4**	81.0	85.7	**1.00**	**1.06**	**1.15**
(84)	70	91.7	96.4	100.0	1.18	1.60	2.17	84.5	95.2	96.4	**1.00**	**1.35**	**1.79**	88.1	95.2	97.6	**1.00**	**1.08**	**1.17**
	100	100.0	100.0	100.0	1.18	1.29	1.46	92.9	96.4	100.0	**1.00**	**1.21**	1.54	92.9	100.0	100.0	**1.00**	**1.07**	**1.11**
	10	49.0	78.4	94.1	1.22	3.61	6.66	46.4	59.5	64.7	**1.00**	**1.48**	**1.74**	**50.3**	**84.3**	**96.7**	**1.00**	**2.87**	**5.58**
	30	73.9	85.0	98.0	1.10	1.84	3.14	**75.2**	84.3	88.2	**1.00**	**1.47**	**1.66**	**74.5**	**89.5**	**98.7**	**1.00**	**1.56**	**2.27**
LOGISTICS	50	86.3	94.1	100.0	1.05	1.46	2.01	**88.9**	**94.8**	95.4	**1.00**	**1.29**	**1.47**	**92.8**	**96.1**	99.3	**1.00**	**1.22**	**1.51**
(153)	70	95.4	96.1	100.0	1.02	1.12	1.41	**96.1**	**99.3**	99.3	**1.00**	1.23	**1.39**	95.4	**96.7**	100.0	**1.00**	**1.05**	**1.20**
	100	100.0	100.0	100.0	1.00	1.00	1.08	95.1	96.7	100.0	1.00	1.08	1.21	100.0	100.0	100.0	1.00	1.00	**1.00**
	10	47.6	81.0	96.4	1.21	3.01	4.90	42.9	60.7	71.4	**1.00**	**1.68**	**2.19**	42.9	76.2	85.7	**1.00**	**2.27**	**3.13**
	30	70.2	89.3	97.6	1.14	2.04	3.55	64.3	85.7	89.3	**1.00**	**1.71**	**1.98**	**72.6**	88.1	97.6	**1.00**	**1.62**	**2.14**
SATELLITE	50	84.5	92.9	98.8	1.12	1.58	2.32	79.8	89.3	94.0	**1.00**	**1.32**	**1.60**	**86.9**	90.5	95.2	**1.00**	**1.25**	**1.40**
(84)	70	95.2	98.8	100.0	1.04	1.35	1.88	84.5	94.0	97.6	**1.00**	**1.25**	**1.44**	88.1	92.9	95.2	**1.00**	**1.13**	**1.20**
	100	100.0	100.0	100.0	1.07	1.18	1.50	96.4	96.4	100.0	**1.00**	**1.14**	**1.29**	96.4	96.4	96.4	**1.00**	**1.04**	**1.07**
	10	36.9	48.8	73.8	1.05	1.90	3.14	**44.0**	**66.7**	**84.5**	**1.00**	**1.75**	**2.70**	**53.6**	**67.9**	**78.6**	**1.00**	**1.51**	**2.02**
	30	61.9	79.8	90.5	1.02	2.06	3.12	**67.9**	78.6	89.3	**1.00**	**1.39**	**1.76**	**76.2**	**84.5**	**91.7**	**1.00**	**1.27**	**1.57**
ZENO	50	76.2	88.1	95.2	1.00	1.57	2.17	**84.5**	**92.9**	**97.6**	1.00	**1.18**	**1.33**	**85.7**	**90.5**	94.0	1.00	**1.08**	**1.17**
(84)	70	90.5	95.2	100.0	1.00	1.30	1.57	**97.6**	**100.0**	100.0	1.00	**1.05**	**1.12**	**98.8**	**98.8**	98.8	1.00	**1.00**	**1.04**
	100	100.0	100.0	100.0	1.00	1.07	1.11	100.0	100.0	100.0	1.00	**1.00**	**1.00**	100.0	100.0	100.0	1.00	**1.00**	**1.00**

Table 4.7: θ-accuracy and spread of LGR, GRNet and LGRNet for test set TS_{LGR}. Results for GRNet and LGRNet are in bold when they are better than the corresponding results for LGR.

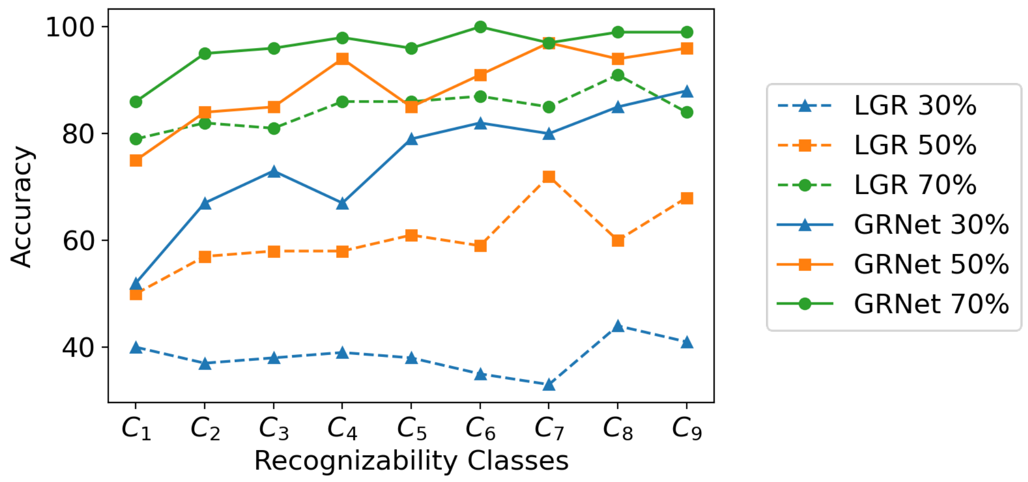


Figure 4.5: Accuracy results of LGR and GRNet on GR instances grouped into classes of decreasing difficulty with test set TS_{Rec}. C_1 is the most difficult class; C_9 is the easiest one.

appears to be more stable over the recognizability classes than GRNet. However, GRNet always performs significantly better than LGR regardless the value of R_Z.

Since the predictive performance of a machine learning system can be deeply influenced by the number of training instances, we experimentally investigated how much GRNet is sensible to this issue. We focus the analysis on the domain SATELLITE, training several neural networks with different fractions of our training set: 20%, 40%, 60% and 80%. Figure 4.6 shows how accuracy increases for $\text{TS}_{\text{LGR}Gen}$ when the training set size increases. In particular, for $\text{TS}_{\text{LGR}Gen}$ we can observe that using only 20% of the training instances gives accuracy lower than 40 in all three cases (30-50-70% of observed actions), but accuracy rapidly improves reaching more than 60 using 60% of the training instances.

We evaluated GRNet also for larger training sets, up to twice the number of instances in the original training set. As it can be seen in Figure 4.6, the enlarged training set for $\text{TS}_{\text{LGR}Gen}$ produces only a small improvement in accuracy.

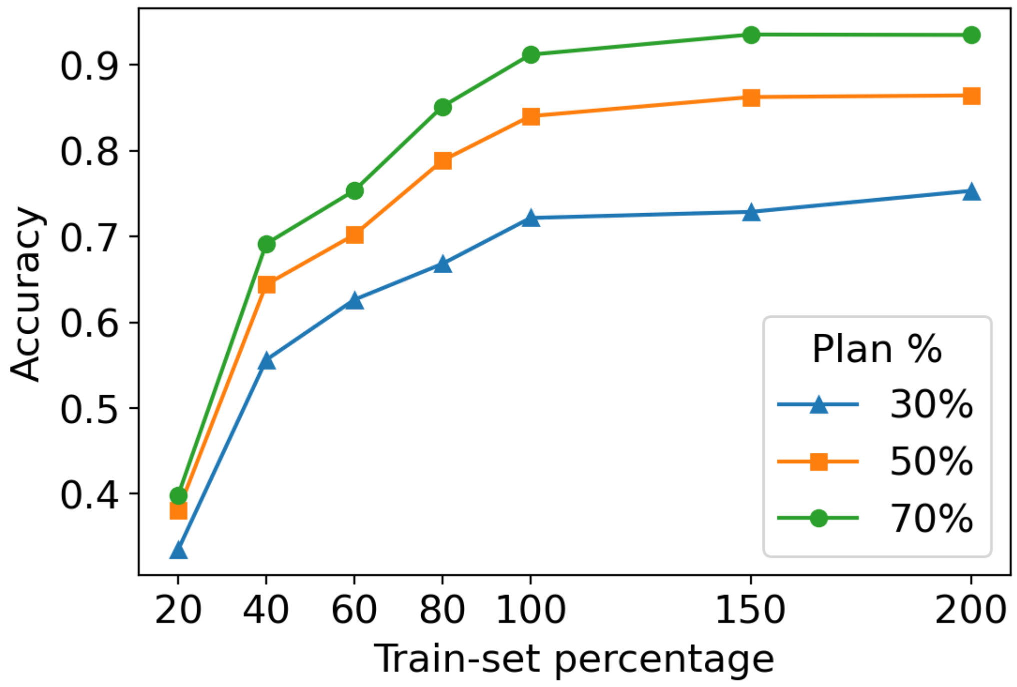


Figure 4.6: Accuracy of GRNet trained using data sets of different sizes (% of the original train set) using test set $\text{TS}_{\text{LGR}\mathit{Gen}}$ in domain SATELLITE.

4.6 Discussion

The proposed architecture, i.e. GRNet, is an approach to address goal recognition as a deep learning task. Our system learns to solve goal recognition tasks as classification problems learning from past experience in a given domain. The learning process consists in training only one neural network for the considered domain, allowing to solve a large collection of GR instances in the domain by the same trained network. Moreover, GRNet can be effectively integrated with the state-of-the-art model-based system LGR. An experimental analysis shows that GRNet and LGRNet, our system integrating GRNet and LGR, perform generally well for the considered benchmark domains, in terms of accuracy, θ-accuracy and spread.

Differently from LGR, the GR instances addressable by GRNet are limited to those involving subsets of fluents and actions that were used in the training phase. If the GR instance to solve involves a new fluent, clearly such a fluent cannot be predicted in GRNet; if the instance involves a new action, such an action cannot be part of the input observed actions for GRNet. Another possible drawback of the proposed approach is that it is important to have a large dataset for training the net, which in some contexts may not be available. However, this problem can be mitigated by exploiting data-augmentation techniques, or by dataset generation assisted by automated planning, as we did for our experiments. Finally, we can notice how the network, unlike some model based approaches like PRP, does not provide any formal guarantees of correctness; in other words, there is no way to tell whether we can trust the network prediction or not.

We try to address some of these problems by adopting a new architecture that will be presented in the following chapter, Chapter 5.

Chapter 5

Fast and Slow Goal Recognition

Advancements in algorithms, techniques, computational power, and specialized hardware have made automated reasoning tools more efficient and reliable. However, they often still lack some desired properties that are common in human intelligence, such as generalizability, robustness, and abstraction. To address these limitations, a growing number of AI experts are striving to develop systems that possess more human-like properties.

One of the main strategies is to create cognitive architectures that utilize a combination of neural networks and symbolic/logic-based AI. This chapter explores multi-agent planning for goal recognition within the context of one such architecture [10], inspired by dual process theories, like the one described in Kahneman's "Thinking, Fast and Slow" [38]. According to this theory, human reasoning is divided into two distinct systems: System 1 (fast thinking) and System 2 (slow thinking), that handle decision-making in different ways. System 1 makes intuitive and imprecise decisions, while System 2 deals with complex, logical, and rational decision-making. The division of responsibilities between the two systems is based on the difficulty of the problem and the experience gained in solving it. Over time, System 2 accumulates examples that System 1 can later use with ease. The division between the two systems enables humans to consider different levels

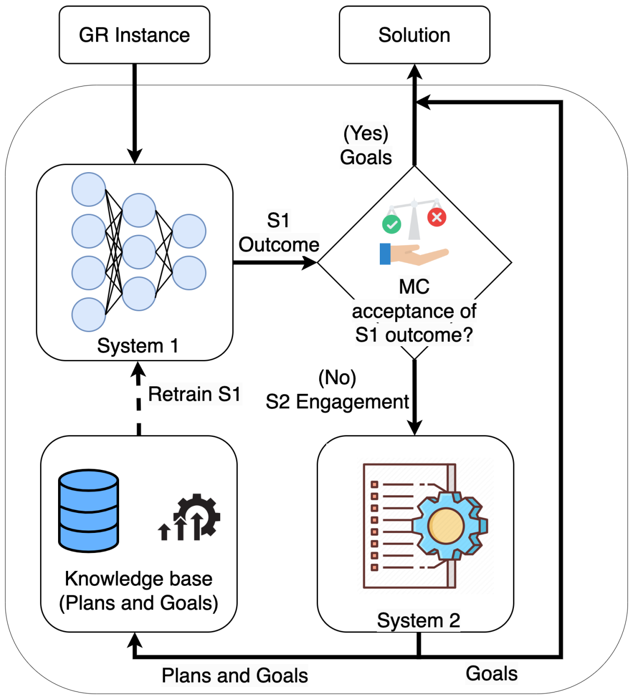


Figure 5.1: Architecture of FSGR. A GR instance is processed by System 1 (S1) and its solution is evaluated by a meta-cognitive agent which decides whether to trust it or not. In the latter case, System 2 (S2) is engaged. The plans computed by System 2 are stored in the knowledge base and used for training System 1.

of abstraction, adapt and generalize their experiences, and multi-task. In this chapter, we present an architecture based on the Fast and Slow principles to solve goal recogniton, called Fast and Slow Goal Recognition (FSGR). FSGR can exploit both the fast, experience-based goal recognition provided by a neural network (i.e. GRNet), and slow, deliberate analysis provided by the planning techniques (i.e. PRP). This is a novel way of combining planning based and data-driven approaches, allowing FSGR to provide both guarantees of correctness, provided by the planner and a certain degree of trust over the prediction provided by the network.

This chapter describes and extends the work presented in [14]

5.1 Fast and Slow Architecture for Goal Recognition

In this section, we describe the architecture of the implemented system, called Fast and Slow Goal Recognition (FSGR), based on the "Fast and Slow" framework. To effectively orchestrate this dual process framework, we developed a meta-cognitive agent, as suggested in the existing literature [22]. In particular, we show how System 1 (also S1) and System 2 (also S2) are realized and how the meta-cognitive agent (also MC) orchestrates between the two systems to provide the final prediction.

The architecture is depicted in Figure 5.1: given a GR instance, System 1 computes a solution based on past experience in the domain. The solution proposed by S1 is then evaluated by the meta-cognitive agent that decides whether to accept it or engage System 2 for a better solution. In this case, the solution of the instances computed by System 2 is also added to the knowledge base for possible retraining of System 1. We assume that the agent is fully rational and follows optimal plans to achieve its goals, as described in Section 4.5. Moreover, we assume partial observability of the agent's actions (i.e. only a percentage of the actions can be seen) and absence of noise in such observations.

5.1.1 System 1 and System 2

System 1 is developed with an MFGR system based on an LSTM neural network, as proposed in GRNet, presented in Section 4.2; in particular, we only implement the Environment Component of GRNet. The structure of S1 is shown in Figure 5.2. The input of the network is the sequence of action O in the GR instance. The output of the Environment Component of GRNet is a score in $[0, 1]$ for each proposition in F. The output of the i-th neuron $\overline{o}_i$ corresponds to the i-th fluent f_i (fluents are lexically ordered), and the activation value of $\overline{o}_i$ gives a rank for f_i being true in the agent's goal (with a rank greater than 0.5 meaning that f_i is true in the goal). For each fact f_i, we evaluate network performance using common machine learning metrics, such as precision, recall, or

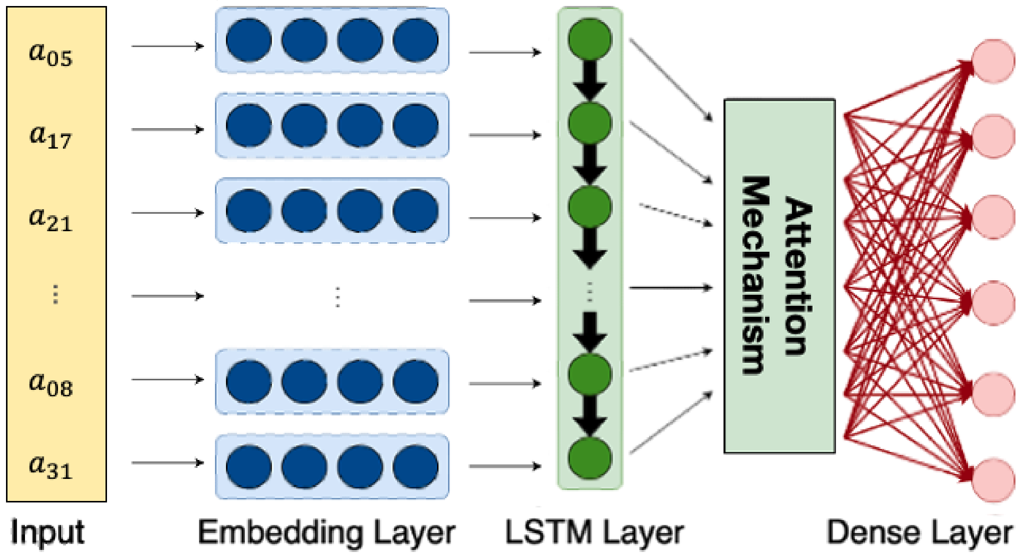


Figure 5.2: Architecture of System 1 composed by the Environment Component of GRNet.

F-Score on a separate validation set, made of instances that the network did not see at training time. In particular, we calculate the precision score as:

$$\textit{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

Where TP represents the fluents correctly identified by our neural network while $TP + FP$ represents all those fluents predicted by the network as true. We use this score to measure how the neural network performs when it makes a specific prediction.

System 2 is based on the work of Ramírez and Geffner [55] (i.e. PRP), described in Section 2.3.1. Given an instance of a goal recognition problem $T = \langle \Pi, I, O, \mathcal{G} \rangle$, we want to compute $\mathcal{G}^*_T$, the exact solution to the problem. $\mathcal{G}^*_T$ is a goal set that contains all goals $G \in \mathcal{G}$ such that some optimal plan for a classical planning problem $P = \langle \Pi, I, G \rangle$ satisfies the observation sequence O. To obtain this goal set, we run an optimal planner twice for each goal in $\mathcal{G}$. First, we compute an optimal solution, if it exists, to the planning problem $P_G = \langle \Pi, I, G \rangle$ for all $G \in \mathcal{G}$; then again, for all $G \in \mathcal{G}$, we compute an optimal solution to a transformed planning problem P'_G whose solution is a solution for P_G that satisfies the observations in O. In our implementation, we compile the problems using PAC-C [9], this is possible by converting P'_G into a PAC problem $\langle \Pi, I, G, C \rangle$, where the set C contains the constraints to encode the observed actions $a_1 \dots a_k$. In PAC, these constraints are expressed through the so-called $pattern\ a_1 \dots a_k$ constraint which can be handled by PAC-C with the addition of the extra effects of the actions in A' related to the observed actions off-the-shelf.

5.1.2 Meta Cognitive Agent

Algorithm 1 MC Algorithm

Input:

- $T = \langle \Pi, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$: the goal recognition instance
- y_i: output of S1 for the current instance
- p: vector of S1 precision scores for each fact on a validation set
- τ_1, τ_2: Meta Classifier thresholds

Output:

A set of predicted goals

```
1: $\hat{G} \leftarrow \arg\max_{G \in \mathcal{G}} \sum_{f \in G} (y_i[f])$
2: $\hat{G}_2 \leftarrow \arg\max_{G \in \mathcal{G} - \{\hat{G}\}} \sum_{f \in G} (y_i[f])$
3: $confidence \leftarrow \sum_{f \in \hat{G}} (y_i[f]) - \sum_{f \in \hat{G}_2} (y_i[f])$
4: $count \leftarrow 0$
5: $sum \leftarrow 0$
6: for $f \in \hat{G}$ do
7:     if $y_i[f] > 0.5$ then
8:         $sum \leftarrow sum + p[f]$
9:         $count \leftarrow count + 1$
```

```
10:     end if
11: end for
12: $experience \leftarrow sum/(count + \epsilon)$
13: if $confidence > \tau_1|\ \ \&\ \ experience > \tau_2$ then
14:     return $\{\ \hat{G}\ \}$
15: else if $experience > \tau_2$ then
16:     $goalset \leftarrow \{G \in \mathcal{G} | 0 \leq \sum_{f \in \hat{G}} y_i[f] - \sum_{f \in G} y_i[f] \leq \tau_1\}$
17: else
18:     $goalset \leftarrow \mathcal{G}$
19: end if
20: $results \leftarrow S2(\langle \Pi, \mathcal{I}, \mathcal{O}, goalset \rangle)$
21: if $results == \{\}$ then
22:     return $\hat{G}$
23: else
24:     return $result$
25: end if
```

Inspired by Ganapini et al. [22], in our system, the meta-cognitive agent (MC) is in charge of deciding whether to accept the solution proposed by S1 or, instead, engage S2 to evaluate a better solution. Intuitively, the meta-cognitive assessment is twofold. On the one hand, it considers S1 confidence in its answer. But due to the fact that S1 is a machine learning approach that may have great confidence in the solution even if it is utterly wrong, the meta-cognitive agent also evaluates the level of correctness of System 1 in similar tasks.

The pseudocode of the meta-cognitive process is reported in Algorithm 1. It takes in input a goal recognition instance T, the solution y_i proposed by S1, the vector p of S1 precision scores (i.e., the precision metric calculated for each fact in F on a validation set), and two thresholds τ_1, τ_2. The solution proposed by S1 is a vector of real numbers that represents a score in $[0, 1]$ for each proposition in F. First, MC calculates a score for each goal $G_i \in \mathcal{G}$. This score is obtained as the sum of the GRNet output for all fluents belonging to G_i. The candidate goal $\hat{G}$ is the one with the highest score (line 1). MC also calculates the second-best candidate, $\hat{G_2}$ (line 2), this is used to define the confidence of S1 for the proposed solution. This is computed as the difference between the score of the candidate goal and the second-best candidate (line 3). We call this metric $confidence$. The intuition behind this metric is that if there is a large difference between these two goals, the neural network has identified more fluents belonging to $\hat{G}$ than belonging to $\hat{G_2}$, and therefore $\hat{G}$ is the most probable goal according to GRNet.

In order to evaluate the quality of the prediction provided by the network, we compute a metric, called $experience$. This metric indicates the network performance on problems that cover similar goals, in particular, it evaluates how often it predicted a corrected output in the past (line 12, where ϵ is a small value used to avoid division by zero). In particular for each fluent f in the candidate goal $\hat{G}$ (line 6), we calculate the average precision over only the ones with a score greater than a threshold (in our case 0.5), meaning that f is true (lines 4-12). Intuitively, this indicates how the network performs when it chooses to predict those fluents, and therefore if its prediction is reliable or not.

The confidence and experience metrics are compared with two thresholds (τ_1 divided by the number of fluents in the candidate goal and τ_2) and, if both exceed those thresholds, MC trusts the prediction made by S1, returning $\hat{G}$ as the candidate goal predicted by the whole system (lines 13-14). In our experiments, after a grid search optimization phase, we set $\tau_1 = 0.08$ and $\tau_2 = 0.8$. Otherwise, if the network has enough experience with those fluents but predicted two or more goals with similar scores, MC selects all the goals for which the scores provided by the network are in the range $[\sum_{f \in \hat{G}} y_i[f] - \tau_1, \sum_{f \in \hat{G}} y_i[f]]$ (lines 15-16), these goals are then examined by S2 (line 20). Finally, if the network does not perform sufficiently on the fluents in $\hat{G}$, MC chooses to discard its predictions and all the goals in $\mathcal{G}$ are processed by S2 (lines 17-20). If S2 does not return any solution, the solution of S1 is returned instead (lines 21-25).

5.1.3 Updating System 1 using System 2

An important desideratum consists in improving the performance of S1 the more the system is used, as the experience on the domain increases. To do that, we fine-tuned S1 using the optimal plans that are generated by S2 every time this is adopted to solve a goal recognition instance: when a new goal recognition instance is solved by S2, we memorize each generated plan together with the candidate goal that this plan is achieving in a *temporary* memory buffer of predefined size L. Once the memory buffer is full, actions are randomly selected (preserving their sequential order) from the stored plans in order to create new observation sequences. These observations, along with their corresponding goals, are then used for continuing the training of S1, with a fine-tuning procedure that starts from the weights previously learned.

Following the completion of the fine-tuning procedure, some plans are randomly selected to be transferred from the temporary memory buffer to a *permanent* memory buffer with a capacity of $2L$, after which the temporary memory buffer is cleared. As the permanent memory buffer fills up, old plans are randomly replaced with new ones, thus maintaining a balanced number of samples among all the temporary memory buffers used. If observations derived from the plans in the temporary memory contain new actions or achieve unseen fluents, the permanent and the temporary memory buffers are used to train a new model from scratch. In our experiments, we set $L = 960$ plans.

5.1.4 Running Example

As a simple example of the overall working of our FSGR system, we will use the simple example reported in Section 4.1. We use the same assumption applied to the Running Example of GRNet (Section 4.2.3): the GR instances involve at most 22 blocks, a total of 968 actions (i.e. $|A| = 968$) and a total of 506 possible output fluents (i.e. $|F| = 506$). We also assume that the three observed actions `(Pick-Up Block_C)`, `(Stack Block_C Block_B)` and `(Pick-Up Block_F)` have ids corresponding to indices 5, 17 and 21 respectively, as in the previous example. The two possible goals in $\mathcal{G}$, as reported in Figure 4.1, are $G_1 = \{\texttt{(On Block\_F Block\_C) (On Block\_C Block\_B)}\}$ and $G_2 = \{\texttt{(On Block\_G Block\_H)}, \texttt{(On Block\_H Block\_F)}\}$

Initially, the observation sequence is computed by S1, which produces a score between 0 and 1 for all 506 propositions. However, in our specific GR instance, we consider only the facts belonging to one of the goals in $\mathcal{G}$. For these facts we obtain $\overline{o}_{\texttt{(On Block\_F Block\_C)}} = 0.017$, $\overline{o}_{\texttt{(On Block\_C Block\_B)}} = 0.810$, $\overline{o}_{\texttt{(On Block\_G Block\_H)}} = 0.000$, $\overline{o}_{\texttt{(On Block\_H Block\_F)}} = 0.003$. These values are then passed to the MC for computing the confidence and experience metrics. Given that the final scores are 0.827 for G_1 and 0.003 for G_2, the MC selects G_1 as the goal with the highest score ($\hat{G}$ in Algorithm 1) and G_2 as the second best choice ($\hat{G}_2$ in the algorithm). Please note that, in this simple example G_2 is the only possible candidate for $\hat{G}_2$; however in GR instances with more than two candidate goals (i.e. $|\mathcal{G}| > 2$), $\hat{G}_2$ is the candidate goal with the second highest final score. The *confidence* metric is computed as the difference of the two final scores: $1.017 - 0.003 = 1.014$. Supposing of having a threshold of $\tau_1 = 0.08$, we can see that the confidence is way over the threshold τ_1, and therefore we consider that S1 is certain enough of its prediction.

Then, we evaluate the *experience* metric, i.e. the average precision of S1 on those fluents predicted as true. The only fluent predicted as true (i.e. with a score higher than 0.5) is $\texttt{(On Block\_C Block\_B)}$, which has a score of 0.810. Suppose that the precision of S1, calculated on a validation set, for that fluent is 0.82 and the threshold for the experience metric is $\tau_2 = 0.8$. Given that the precision value is greater than τ_2, the Meta Cognitive Agent trusts S1 and returns G_1 as the predicted goal. Otherwise, suppose that the precision of S1, calculated on a validation set, for $\texttt{(On Block\_C Block\_B)}$ is 0.19, lower than τ_2. In this case the Meta Cognitive Agent, recognising that S1 doesn't have enough experience to provide a trustworthy solution, calls S2. If S2 is able to provide a solution within a predefined amount of time (e.g. 15 minutes), the MC Agent returns it as the final solution to the problem. However, if S2 is not able to provide any solution within the given time, the MC Agent returns the goal identified by S1.

5.2 Benchmark Suite and Datasets

We experimentally evaluate our system and compare it with the state-of-the-art model for goal recognition by observing optimal plans, PRP. We consider four well-known benchmark domains, introduced in Section 4.4: BLOCKSWORLD, DEPOTS, LOGISTICS and ZENOTRAVEL; of course, as for GRNET, FSGR can be trained and tested using other domains. In the domains considered, we use automated planning techniques to create the solved GR instances for the training and test sets. Concerning the training set, for each domain, we randomly generated a large collection of plan generation problems of different sizes. We use the same number of actions ($|A|$), fluents ($|F|$), fluents for each goal ($|G_i|$), number of candidate goals ($|\mathcal{G}_i|$) and objects used in the experimental setup of GRNET, reported in Table 4.2. For each problem, we computed an optimal plan for solving it, in this case, we adopted the Big Joint Optimal Landmarks Planner [18] to compute the solution. For evaluation, we generated a test set made of 600 GR instances (not seen at training time) of different sizes. The procedure for obtaining problems and plans is akin to the one we described for the training set. Optimal solutions for the planning problems compiled with PAC-C are computed with the A* search guided by the h_{LM-Cut} heuristic that supports conditional effects [58], implemented in FastDownward [31]. For each optimal plan π_i, we derived three different observations by subsampling 30%, 50%, and 70% of each π_i actions. For each considered domain, this results in three groups of test instances, allowing us to evaluate the performance of FSGR in terms of different amounts of available observations.

The experiments were conducted on an Intel Xeon Gold 6140M 2.3 GHz processor. For calculating optimal solutions, runtime and memory were constrained to 1800 seconds and 8GB, respectively. When training the models, memory utilization was limited to 40GB. The system is evaluated in terms of two different aspects:we evaluate the GR accuracy, and we also keep track of the time taken by the systems to find a solution. For a set of test instances, the accuracy is defined as the percentage of instances whose goals are correctly identified (predicted) over the total number of instances in the test set. If,

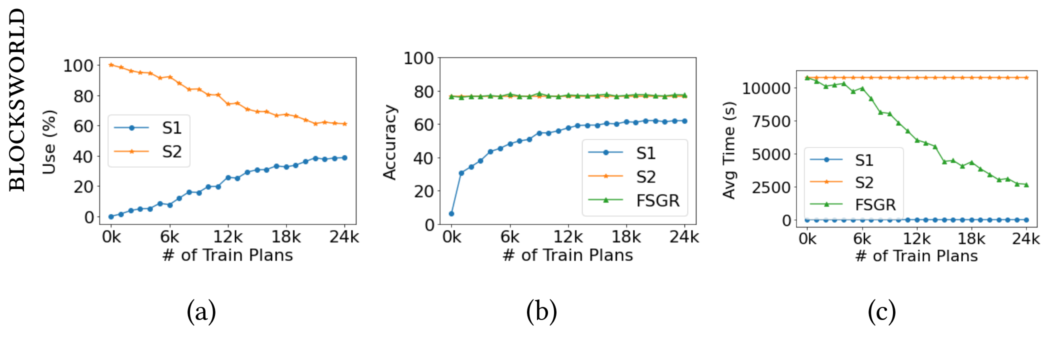


Figure 5.3: Average performance on BLOCKSWORLD domain considering the whole test set: (a) Use of S1 and S2; (b) Accuracy of S1, S2, and FSGR; (c) Time for S1, S2, and FSGR.

for a problem instance, the evaluated system provides k different goals with the same highest score, then, in the overall count of the solved instances, this instance has value $\frac{1}{k}$ if the true goal is one of these k goals, 0 otherwise.

5.3 Experimental Results

We analyzed the performance of FSGR on the test set as well as on the three subgroups of instances based on the percentage of plan observations. The performances on the three different subsets of instances of the test set in terms of GR accuracy and time are reported in Table 5.1. For all the considered domains, we can see that the accuracy of System 1 increases with both the number of plans used for training GRNet and the percentage of observed actions of the plan. This is consistent with the fact that the more information is used to train System 1, the better it performs. However, it is interesting to notice that the system is able to exploit the available information providing a gain either in accuracy or in time. For instance, considering 50% of the plan, the accuracy of GRNet in the DEPOTS domain changes from 24.8%, when $6k$ plans are used, to 77.1% with $24k$ plans. This improvement leads the meta-cognitive system to trust System 1 more, and thus using it

Domain	Train plans	30% of the plan					50% of the plan					70% of the plan				
		A_{S1}	A_{S2}	A_{FSGR}	T_{S2}	T_{FSGR}	A_{S1}	A_{S2}	A_{FSGR}	T_{S2}	T_{FSGR}	A_{S1}	A_{S2}	A_{FSGR}	T_{S2}	T_{FSGR}
	-	-	62.5	-	9108	-	-	79.6	-	10958	-	-	88.1	-	12124	-
	0	4.7		62.5		9108	7.2		79.6		10958	7.0		88.1		12124
BLOCKSWORLD	6k	35.1		65.3		8364	49.5		80.6		9818	59.9		88.3		11677
(600)	12k	42.8		66.4		5506	57.9		78.1		5811	72.3		87.9		6778
	18k	46.5		66.9		4504	62.4		77.8		4263	75.3		86.7		4320
	24k	46.3		67.7		3298	62.9		77.8		2507	77.0		87.0		2214
	-	-	85.4	-	4148	-	-	94.2	-	4330	-	-	97.3	-	4487	-
DEPOTS	0	19.2		85.4		4148	16.2		94.2		4330	16.6		97.3		4487
(600)	6k	22.3		85.4		4148	24.8		94.2		4330	25.9		97.3		4487
	12k	45.5		81.0		2930	60.6		87.8		2580	69.3		90.2		2312
	18k	57.5		79.8		1172	74.6		89.1		378	86.3		93.0		251
	24k	58.7		82.4		837	77.1		92.3		182	87.4		95.5		73
	-	-	62.4	-	10629	-	-	66.5	-	10467	-	-	67.9	-	10388	-
	0	9.7		62.4		10629	10.7		66.5		10467	9.1		67.9		10388
LOGISTICS	6k	45.1		62.8		10375	55.8		66.6		10129	55.3		68.1		10225
(600)	12k	52.0		69.1		5736	62.2		70.5		6164	70.5		70.8		7117
	18k	54.8		72.7		4854	65.3		75.4		4486	77.8		76.3		4658
	24k	59.1		74.5		4112	71.2		78.7		3004	81.8		79.8		3018
	-	-	92.7	-	4998	-	-	97.2	-	5591	-	-	98.9	-	6008	-
	0	16.0		92.7		4998	17.1		97.2		5591	18.5		98.9		6008
ZENOTRAVEL	6k	75.9		86.3		478	87.8		92.8		54	95.3		96.9		2
(600)	12k	74.5		88.1		581	87.3		91.9		51	95.1		97.9		10
	18k	75.2		88.3		597	87.8		92.5		66	95.5		96.6		4
	24k	75.5		88.5		590	87.3		93.4		69	94.8		97.6		14

Table 5.1: Performance of FSGR in terms of accuracy (in %) and Time (in seconds) considering goal recognition problems into which the actions observed are the 30%, 50%, or 70% of the entire plan. In the A_{S1} columns, we report the accuracy of System 1 (i.e., GRNet), in the A_{S2} columns, we report the accuracy of System 2 (i.e., PRP), in the A_{FSGR} columns, we report the accuracy of FSGR. In the T_{FSGR} column, we report the average time taken by FSGR to find a solution. For each domain, the first row reports the performance of System 2, and from the second row on, we report different stages of the incremental training of System 1.

Domain	Train plans	5 minutes					10 minutes					15 minutes				
		A$_{S1}$	A$_{S2}$	A$_{FSGR}$	T$_{S2}$	T$_{FSGR}$	A$_{S1}$	A$_{S2}$	A$_{FSGR}$	T$_{S2}$	T$_{FSGR}$	A$_{S1}$	A$_{S2}$	A$_{FSGR}$	T$_{S2}$	T$_{FSGR}$
	-	-	69.9	-	3337	-	-	73.6	-	5258	-	-	75.3	-	6864	-
	0k	6.3		69.9		3337	6.3		73.6		5258	6.3		75.3		6864
BLOCKSWORLD	6k	48.2		71.4		3041	48.2		75.1		4834	48.2		76.7		6335
(600)	12k	57.7		72.1		1868	57.7		75.2		2955	57.7		76.4		3866
	18k	61.4		72.8		1398	61.4		75.4		2176	61.4		76.4		2824
	24k	62.1		73.8		917	62.1		76.0		1383	62.1		76.9		1771
	-	-	90.6	-	1682	-	-	92.2	-	3033	-	-	92.3	-	4265	-
	0k	17.3		90.6		1682	17.3		92.2		3033	17.3		92.3		4265
DEPOTS	6k	24.3		90.6		1682	24.3		92.2		3033	24.3		92.3		4265
(600)	12k	58.5		85.2		1023	58.5		86.2		1840	58.5		86.3		2583
	18k	72.8		86.6		249	72.8		87.1		435	72.8		87.3		599
	24k	74.4		89.4		159	74.4		89.9		269	74.4		90.1		364
	-	-	60.8	-	2135	-	-	62.3	-	3969	-	-	63.4	-	5703	-
	0k	9.8		60.8		2135	9.8		62.3		3969	9.8		63.4		5703
LOGISTICS	6k	52.1		61.3		2082	52.1		62.8		3872	52.1		63.7		5566
(600)	12k	61.6		66.7		1281	61.6		67.8		2388	61.6		68.5		3437
	18k	66.0		71.5		942	66.0		72.3		1754	66.0		73.1		2526
	24k	70.7		75.1		689	70.7		75.9		1278	70.7		76.5		1836
	-	-	85.4	-	1398	-	-	93.1	-	2407	-	-	95.0	-	3288	-
	0k	68.2		84.1		1195	68.2		91.3		2056	68.2		93.3		2808
ZENOTRAVEL	6k	86.3		91.5		50	86.3		91.9		82	86.3		92.0		109
(600)	12k	85.6		92.2		58	85.6		92.6		97	85.6		92.6		131
	18k	86.2		92.0		61	86.2		92.4		102	86.2		92.5		136
	24k	85.9		92.7		62	85.9		93.1		102	85.9		93.2		137

Table 5.2: Performance of our system in terms of accuracy (A) and Time (T) considering goal recognition problems considering time limits of 5, 10 and 15 minutes for solving each planning problem computed by S2. In the A_1 and columns, we report the accuracy of System 1 (i.e., GRNet), in the A_{FSGR} columns, we report the accuracy of the integrated system. In the T_{FSGR} column, we report the average time the integrated system needs to find a solution. For each benchmark domain, the first row reports the performance of System 2 (i.e., PRP), from the second row instead we report different stages of the incremental training of System 1.

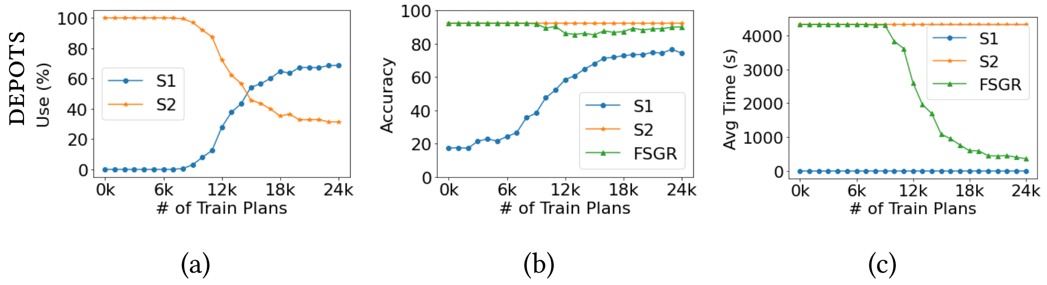


Figure 5.4: Average performance on DEPOTS domain considering the whole test set: (a) Use of S1 and S2; (b) Accuracy of S1, S2, and FSGR; (c) Time for S1, S2, and FSGR.

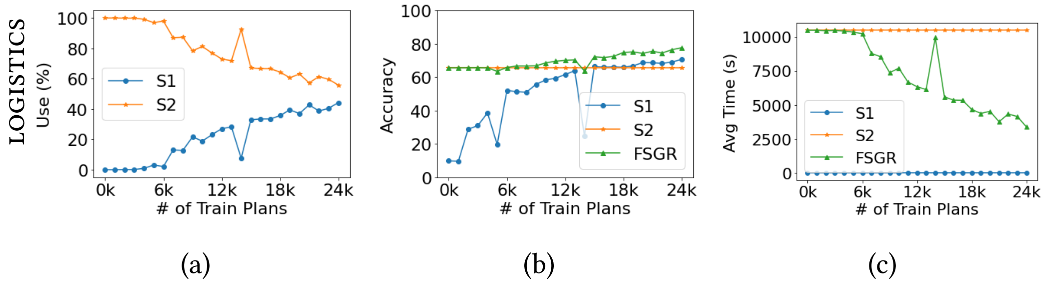
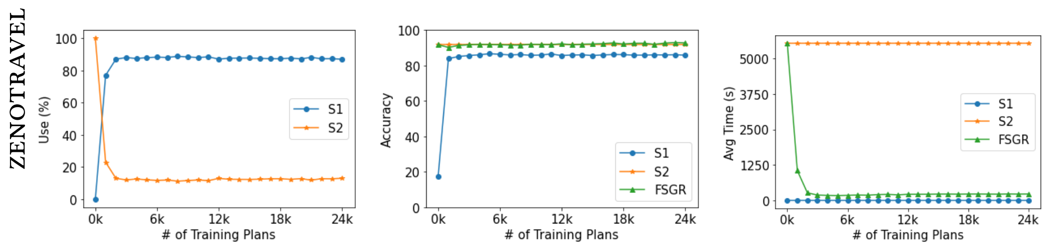


Figure 5.5: Average performance on LOGISTICS domain considering the whole test set: (a) Use of S1 and S2; (b) Accuracy of S1, S2, and FSGR; (c) Time for S1, S2, and FSGR.

more times without exploiting System 2, which is slower and more resource-demanding. The effect of this process can be seen in the T_{FSGR} column reporting the average time to compute a solution by the integrated system: the value of T_{FSGR} decreases drastically as the number of train plans increases. For instance, in DEPOTS, the time required by FSGR to find a solution changes from 4330 seconds to 182 seconds, more than twenty times less. Despite this time reduction, the system maintains a high accuracy, which is always over 87%. Notice that with 70% of the observed plan, the system obtains 95.5% (just 1.8 points less than PRP) with an average time of only 73 seconds compared to the 4430



(a) (b) (c)

Figure 5.6: Average performance on ZENOTRAVEL domain considering the whole test set: (a) Use of S1 and S2; (b) Accuracy of S1, S2, and FSGR; (c) Time for S1, S2, and FSGR.

required by PRP.

An interesting result of this approach is reported in the LOGISTICS domain. For all the considered cases, it can be noticed that trusting System 1 increases the overall performance of the integrated system, which obtains an accuracy even higher than the one reached by System 2, that has guarantees of correctness. For instance, with 50% of the plan, System 2 has an accuracy of 66.5%, while the integrated system, with a fully trained GRNet, reaches 78.7%. These results are due to the fact that System 1 can provide a solution in less time than System 2, overcoming the time limit of the system. Regarding the results reported in Table 5.1, we set the limit to 30 minutes for computing the solution of a single planning instance. Although in several cases this time limit is not enough for S2, the system is able to compute a solution even for these cases by exploiting System 1 capabilities.

The results in Table 5.1 allow us to compare our system with two baselines: a system that randomly predicts a goal (in Table 5.1, this corresponds to the results on rows with 0 train plans, i.e. a neural network not trained which acts as a random classifier) and the state-of-the-art PRP (A_{S2} and T_{S2} columns in Table 5.1 respectively). We can see that when System 1 performs very poorly due to insufficient training (as in the 0k rows of Table 5.1 and, for instance, in the 6k row for the LOGISTICS domain), the integrated system behaves almost as PRP in terms of average accuracy and time. On the contrary, with a properly trained System 1, the behavior of the integrated system shows a neglectable decrease of just a few points in terms of accuracy with respect to PRP but with a considerable gain in terms of time. In fact, in some cases such as in BLOCKSWORLD with 24k training plans with 70%, we can even reduce the time necessary to compute a solution by more than 5 times. This shows the capability of our meta-cognitive system to work remarkably well in both cases, taking the best of System 1 and System 2.

In Figures 5.3, 5.4, 5.5 and 5.6 we graphically show the behavior of our system respectively for BLOCKSWORLD, DEPOTS, LOGISTICS, and ZENOTRAVEL. In each figure, we report three different plots showing three different aspects and how they vary during the incremental training of GRNet: in the first, we report the use of Systems 1 and System 2 (i.e., the number of times the meta-cognitive system adopts one of the two systems, in percentage). It can be noticed that, in each considered domain, S1 is more used when it is trained with a larger quantity of plans. For instance, in DEPOTS (Figure 5.4a), S1 is used for 68.6% of the test instances when it is trained with 24k plans. This result is expected: in fact, a larger training set should lead to better performance for S1 and, therefore, the Meta Cognitive System should be more inclined to trust its predictions. In the second plot we report the accuracy of S1, S2, and the overall accuracy of FSGR. In the third plot, we represent the average time required to find a solution to a GR instance. It is worth noting that, while Table 5.1 presents results categorized according to various percentages of observations, the presented figures display the average results, considering all percentages collectively, which means that the reported performance refers to the whole test set. In BLOCKSWORLD, it can be noticed that the increase in the use of System 1 (see Figure 5.3a) corresponds to a negligible variation in accuracy (see Figure 5.3b), but the time taken to compute a solution decreases (see Figure 5.3c). A similar but more determined behavior can be seen in ZENOTRAVEL. In fact, in this domain, even a limited number of plans allows GRNet to obtain remarkably good performance reaching 80% of use very rapidly (see Figure 5.6a). Once again, this causes a very small variation in terms of accuracy (see Figure 5.6b) but reduces the time by more than 10 times (see Figure

5.6c). The performance in DEPOTS is similar in principle to that in ZENOTRAVEL, except for the fact that S1 needs more plans to obtain good accuracy. For LOGISTICS, Figure 5.4 illustrates a crucial advantage of S1 over S2, as it demonstrates that S1's quicker computation of solutions allows the entire system to produce outputs even for instances where S2 would exceed its time constraints. This results in higher overall accuracy (see Figure 5.5b).

We also evaluated how sensible our system is to different threshold values of the confidence (τ_1) and the experience (τ_2) metrics. In Table 5.3 we report the performance of the FSGR system for the considered domains. Considering three different values of τ_1 and τ_2, in the considered domains, we can see that when the neural network is not properly trained, in particular when using only 0 or $6k$ training problems, S1 does not have enough confidence or experience in its predictions, and therefore S2 is called most of the time; for instance, in DEPOTS the average time for resolving a problem instance is 4322 seconds. When the performance of System 1 increases with $12k$ and $18k$ training plans, we can notice some differences, especially when varying τ_1. In fact, in BLOCKSWORLD, LOGISTICS and ZENOTRAVEL setting an higher threshold for the confidence metric and, therefore, requesting that only a single goal is selected by the neural network, increases the accuracy by a few points but at the expenses of the computation time. For instance, in BLOCKSWORLD, with $18k$ training plans and $\tau_2 = 0.8$, the accuracy goes from 75.4 to 78.2 while the time to compute a solution goes from 6578 to 6689. In these three domains the experience threshold (i.e. τ_2) doesn't seem to affect much the accuracy or the computation time; this is due to the fact that the networks are very confident of the predicted outputs, with precision values greater than 0.8. However, the higher the experience threshold is, the more correct prediction computed by System 1 will be rejected by the MC agent, leading to higher computation times. Nevertheless, in DEPOTS, where the neural network precision on the validation set is not always high, the experience threshold has an higher impact on the performance of the system. For example, with $18k$ training plans and $\tau_1 = 0.16$, the accuracy goes from 88.7 to 89.9 while the training time goes from 631 to 1501, meaning that some unreliable predictions of System 1, that were in fact wrong, were correctly solved by System 2. A smaller impact can be seen when the neural network is fully trained ($24k$ training plans). Although increasing the confidence and experience thresholds generally leads to a longer computation time without remarkable improvements in terms of accuracy, we can see that in general the S1 is generally precise enough that even setting a low τ_2 does not lead to a loss of accuracy.

Finally, Table 5.2 reports the performance considering different time limits (5, 10 and 15 minutes), to solve each planning problem generated by S2. For all the considered domains, we can see that the accuracy of S1 (column A_{S1}) increases as the number of training plans used increases according to what is reported in Table 5.1. We can also see that, as the time limit grows, both the accuracy of System 2 (columns A_{S2}) and the average time S2 needs to find a solution (columns T_{S2}) increases for almost all the considered domains. However, this is not the case for LOGISTICS, where the average time decreases as the time limit increases. In this domain, FSGR maintains the same properties seen in Table 5.1. In fact, by exploiting the new information collected through training, the meta-cognitive agent is able to choose S1 more often, leading to a gain in either performance or time (columns A_{FSGR} and T_{FSGR}). This is particularly noticeable when S2, due to the time limit, does not perform well; for instance, in BLOCKSWORLD with 5 minutes time limit, starting from $6k$ training plan, FSGR always performs better than S2 both in terms of GR Accuracy and average time. In fact, often S2 is not able to provide any solution within this limited amount of time as demonstrated by its low performance (i.e. $A_{\text{S2}} = 62.5$). As shown in Algorithm 1, when no solution is provided by System 2, MC agent returns the solution of S1, improving the performance of FSGR scenarios where the available time is limited.

Domain	Train plans	$\tau_1 = 0.04$						$\tau_1 = 0.08$						$\tau_1 = 0.16$					
		$\tau_2 = 0.4$		$\tau_2 = 0.8$		$\tau_2 = 09$		$\tau_2 = 0.4$		$\tau_2 = 0.8$		$\tau_2 = 09$		$\tau_2 = 0.4$		$\tau_2 = 0.8$		$\tau_2 = 09$	
		A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}	A_{FSGR}	T_{FSGR}
	0	76.7	10730	76.7	10730	76.7	10730	76.7	10730	76.7	10730	76.7	10730	76.7	10730	76.7	10730	76.7	10730
	6k	75.2	9369	76.6	10000	76.6	10259	76.9	9408	77.6	10012	77.2	10270	77.6	9438	78.0	10025	77.5	10276
BLOCKSWORLD	12k	74.1	4317	75.5	7307	76.8	9460	77.4	4452	77.2	7376	77.1	9485	78.9	4551	78.4	7436	77.6	9507
(600)	18k	73.0	2008	75.4	6578	75.9	9042	76.5	2118	77.1	6630	76.8	9071	78.6	2221	78.2	6689	77.4	9103
	24k	73.6	1727	75.9	5359	75.9	9558	77.1	1836	78.0	5426	76.6	9574	79.3	1927	79.0	5483	77.1	9593
	0	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322
	6k	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322	92.3	4322
DEPOTS	12k	81.6	1884	85.2	2552	89.5	3687	83.3	1957	86.3	2607	90.0	3711	85.0	2013	87.5	2653	90.7	3721
(600)	18k	85.0	533	84.9	556	86.6	1418	87.3	579	87.3	600	88.6	1456	88.7	631	88.7	653	89.9	1501
	24k	87.9	328	87.9	328	88.3	474	90.1	364	90.1	364	90.3	507	91.3	389	91.3	389	91.4	531
	0	65.6	10494	65.6	10494	65.6	10494	65.6	10494	65.6	10494	65.6	10494	65.6	10494	65.6	10494	65.6	10494
	6k	66.0	10179	65.9	10237	65.8	10263	66.0	10186	65.8	10243	65.7	10269	66.0	10192	65.8	10249	65.7	10276
LOGISTICS	12k	70.4	6273	70.4	6273	69.9	6535	70.1	6339	70.1	6339	69.8	6593	70.0	6389	70.0	6389	69.7	6638
(600)	18k	74.7	4615	74.7	4615	74.7	4615	74.8	4666	74.8	4666	74.8	4666	74.7	4686	74.7	4686	74.7	4686
	24k	78.1	3309	78.1	3309	78.1	3309	77.7	3378	77.7	3378	77.7	3378	77.7	3419	77.7	3419	77.7	3419
	0	96.2	5532	96.2	5532	96.2	5532	96.2	5532	96.2	5532	96.2	5532	96.2	5532	96.2	5532	96.2	5532
	6k	90.4	159	90.4	159	90.7	182	92.0	178	92.0	178	92.3	200	93.5	198	93.5	198	93.6	220
ZENOTRAVEL	12k	90.6	196	90.6	196	90.6	196	92.6	214	92.6	214	92.6	214	93.4	230	93.4	230	93.4	230
(600)	18k	90.9	200	90.9	200	90.9	200	92.5	222	92.5	222	92.5	222	93.6	235	93.6	235	93.6	235
	24k	91.0	204	91.0	204	91.0	204	93.2	224	93.2	224	93.2	224	94.6	236	94.6	236	94.6	236

Table 5.3: Performance of FSGR in terms of accuracy (A_{FSGR} in %) and Time (T_{FSGR} in seconds) considering different threshold values for the confidence (τ_1) and the experience (τ_2) metrics for the DEPOTS domain.

Chapter 6

Transformer Based Architectures for Automated Planning

In 2007, the work in [25] drew an interesting parallel between automated planning and natural language processing (NLP), presenting a new formulation for plan recognition, that is the task of inferring an agent's plan observing some of its actions, based on grammars. The authors claimed that the two disciplines could share some of the research results, but they stated that much of the recent work in both fields had gone unnoticed by the researchers in the other field. In the following years, the separation between these two fields has not diminished. For NLP, deep learning techniques such as the ones presented in Chapter 3, like the attention mechanism and Transformer-based architectures have revolutionised the field, reaching a completely new state-of-the-art in many different tasks [17]. On the other hand, deep learning had a limited impact on automated planning, and it is mostly used for predicting heuristics [20], processing sensor data in order to create a symbolic representation of a planning problem [5] and for goal recognition tasks [3, 46].

Inspired by the parallelism pointed out by Geib and Steedman [25], in the following sections, we aim to transfer Transformer-based techniques to the automated planning field. We claim that plans and actions can be seen similarly to sentences and words, and we design a *planning language modeling* task to train the model into which the model has to reconstruct an incomplete sequence of actions. With this technique, for each considered domain, we train a BERT model, described in Section 3.4.2, using tokenized actions, where each token can be either the action name or one of its grounded predicates. This model can be used as a *foundation model* for the considered domain, that is a large machine learning model trained on a vast quantity of data at scale such that it can be adapted to a wide range of downstream tasks. For this reason, we evaluate the performance of the model on its training task and in three additional experiments: Next Token Prediction (NTP), into which the model has to predict the token that follows a input sequence; Previous Token Prediction (PTP), into which the predicted token is the one that precedes the input sequence. Finally we fine-tune the BERT model to perform a goal recognition task. These tasks were made to demonstrate how well the model understands about how a planning domain works, its rules, its actions and their effects. Moreover, in this chapter, we present a detailed report of the main difficulties in the training process, the necessary choices the user has to make in order to train a similar model, the dimension of the datasets required to obtain good results, etc. Finally, we discuss the positive and negative aspects of applying these techniques to the planning domain.

This chapter describes and extends the work presented in [60]

6.1 PlanBERT

In this section, we focus on one of the most recent deep learning architectures for NLP: BERT, which is described in Section 3.4.2. By processing a huge quantity of sentences and documents, BERT is able to learn how the language works and its capabilities can be exploited for different tasks such as text classification [52], sentiment analysis [32], question answering [53] and other NLP tasks. This is done by training the model to perform the so-called *masked language modeling* task. Basically, the input of the BERT model is an incomplete sentence, into which some words are replaced by a special marker, and the model has to reconstruct the complete sentence, predicting the missing words from the context.

6.1.1 Planning Language Modeling

Our BERT model is trained using a slight variation of the typical language modeling task that we call *Planning Language Modeling* (PLM). In order to perform PLM, we approach the planning domains as a NLP task. In this technique, given a sequence of actions π composed of n actions a_i, with $i \in [1, n]$, we split each action into separated tokens, dividing the action name and the action's predicates. For instance, the action `(Stack Block_A Block_B)` is divided into three separate tokens: `Stack`, `Block_A` and `Block_B`.

Initially, the action sequence is divided into separated tokens, using the WordPiece tokenizer [66]. Next, we substitute a certain percentage of tokens with the special token `[MASK]` and give this incomplete sequence as input. The BERT model has the task of predicting the missing token from the overall context of the action sequence. If the model is capable of performing this operation, it is likely that it has the ability to understand how the planning domain works, the impact of the actions, when they are performed by an agent, and which effects they have. In order to perform the planning language modeling, the BERT model needs processing a large quantity of these training instances, progressively adjusting the training weights of the model with the backpropagation algorithm until it reaches a satisfying level of accuracy.

6.1.2 Training Technique

In order to properly train the model, there are some necessary choices that the user must make. Although typically BERT models treat a specific language, there are several multi-language models. Given that a planning domain has its own set of actions, predicates, and rules, we can consider it as a sort of an independent language. Therefore, the user should evaluate whether training a BERT model for each domain, perhaps specialising and improving its predictive capabilities, or building a multi-domain model, selecting which domains to consider and their quantity. In order to properly evaluate the capabilities on each domain separately and to try to achieve higher performance, we opted for training a BERT model for each domain.

Next, an important constraint of the BERT model is that it must have a predefined vocabulary of tokens. Therefore, before the training process, the user has to define a maximum number of ground objects that the model can handle in each domain. Although this process makes the model sensitive to the chosen object names, in practice this operation only affects the choice of the maximum number of objects. In fact, a mapping algorithm can be easily implemented for translating new names into predefined ones.

Considering that the input sequence of actions π is a solution to a classical planning problem $P = \langle F, I, A, G \rangle$, we decided to include the initial state I and the goal G into the input sequence as they are crucial to correctly process the sequence. For this reason, we add two new special tokens, `[EIS]` (End of Initial State) and `[EG]` (End of Goal), to help the model to understand when the initial state and the goal end. On the other hand, we don't want the model to learn the initial state or the goal; for this reason, during training we used the `[MASk]` token only in the action sequence (i.e. after the `[EG]` token).

As it happens with all deep learning techniques, a crucial aspect for obtaining good performance is the number of training instances, i.e. the number of plans necessary to train the BERT model. The choices of how many domains to consider for training and how many objects to include have a serious impact on the number of plans which have to be collected or generated for the training of the model, the amount of time required to this process, and its difficulty. The overall dimension of our data set is about 350 MB of data; therefore, it is definitively smaller w.r.t. the dataset ($\sim$ 3 TB) used for training standard BERT architectures [17, 39].

Our model architecture is very similar to the one proposed by Devlin et al. [17]. We

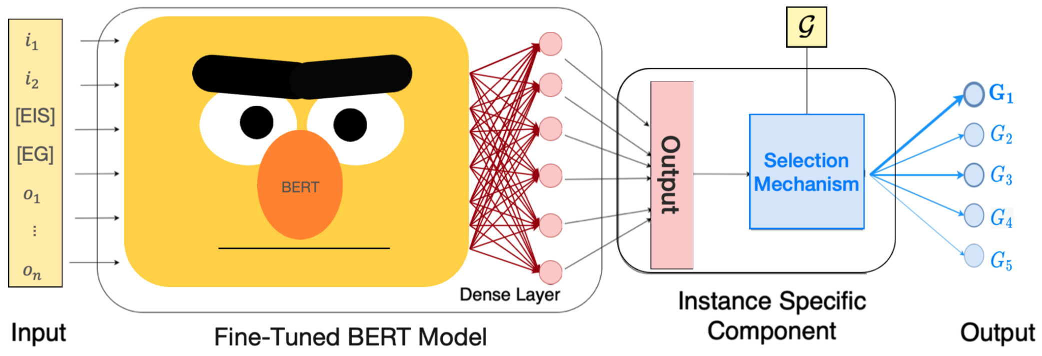


Figure 6.1: Architecture for goal recognition using a fine-tuned BERT model. The architecture receives as input the initial state, an empty goal and the observation sequence. The fine-tuned BERT model outputs $|F|$ neurons, each one representing a possible fluent in the domain. As in GRNet, the output is used by the instance specific component for selecting the goal with the highest score.

use a total of 12 encoding layers, each one of them with 12 heads, and we use 768 as the embedding size, i.e. tokens are represented as vectors of 768 real numbers across the model.

6.1.3 Goal Recognition Fine-tuning

As described in Section 3.4.2, BERT models can be fine-tuned by adding an additional output layer. In order to perform goal recognition, similarly to GRNet, we add an additional feed-forward layer which has $N = |F|$ output neurons with *sigmoid* activation. For the overall working of GRNet, please refer to Section 4.2.

The architecture is shown in Figure 6.1. Given a goal recognition instance $T = \langle \Pi, I, O, \mathcal{G} \rangle$, the BERT model receives as input the initial state I and the observation sequence O, tokenized using the same process described in Section 6.1.1. Similarly to the Environment Component of GRNet, the output of the i-th neuron $\bar{o}_i$ corresponds to the i-th fluent f_i and the activation value of $\bar{o}_i$ gives a rank for f_i being true in the goal. The instance component remains the same and takes as input the ranks generated by the fine-tuned model and uses them to select a goal from the candidate goal set $\mathcal{G}$.

6.1.4 PLM Example

In this section, we present a simple example of how we structure our planning language modeling task. As in previous examples, we use the BLOCKSWORLD domain and we assume that our plans involve at most 22 blocks. In this case, our vocabulary is made of 22 tokens for the blocks (i.e. a different name for each block), 4 tokens for the possible actions (i.e. `Stack`, `Unstack`, `Pick-Up`, `Put-Down`), 4 tokens for the possible predicates (i.e. `clear`, `on-table`, `holding`, `arm-empty`) and the 3 for the additional tokens (i.e. `[MASK]`, `[EIS]`, `[EG]`) for a total of 33 different tokens.

In order to perform a planning language modeling task, we need an initial state I, a goal G and an action sequence π that leads from I to G. The initial state, reported in Figure 6.2a, is $I = \langle$ `(on-table Block_F)`, `(on-table Block_C)`, `(on Block_B Block_F)`, `(arm-empty)` $\rangle$. The goal, reported in Figure 6.2b is $G = \langle$ `(on Block_B Block_C)` $\rangle$. Finally, Figure 6.2c reports the solution plan $\pi = \langle$ `(Unstack Block_B Block_F)`, `(Stack Block_B Block_C)` $\rangle$.

Therefore, the tokenized input sequence is "`on-table`" "`Block_F`" "`on-table`" "`Block_C`" "`on`" "`Block_B`" "`Block_F`" "`arm-empty`" "`[EIS]`" "`on`" "`Block_B`" "`Block_C`" "`[EG]`" "`Unstack`" "`Block_B`" "`Block_F`" "`Stack`" "`Block_B`" "`Block_C`". Next, we mask 1 token after `[EG]` (17%), chosen randomly; in this example the `Stack` token is replaced by `[MASK]`.

This sequence is then processed by the embedding layer and by all the encoding layers, producing a final representation of each token as a vector of 768 real numbers. Given that the fourth element of the sequence is masked, its embedding vector is passed to a feed-forward layer composed of 33 neurons (one for each possible token of the domain) with the softmax activation function. Supposing that the output neuron corresponding to the `Stack` token has index 8, we want in output a vector formed by all zeroes except

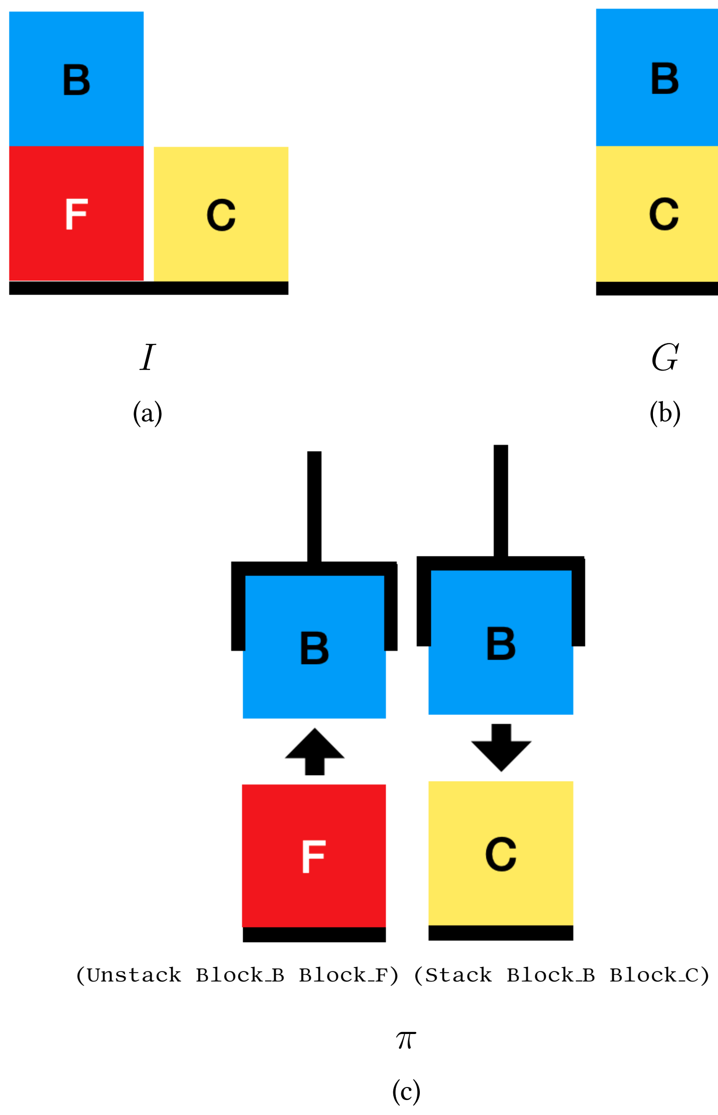


Figure 6.2: Representation of the toy classical planning problem in the BLOCKSWORLD domain and a solution plan π. (a) The initial state I, (b) The goal G, (c) A solution plan $\pi = \langle$ `(Unstack Block_B Block_F)`, `(Stack Block_B Block_C)`$\rangle$.

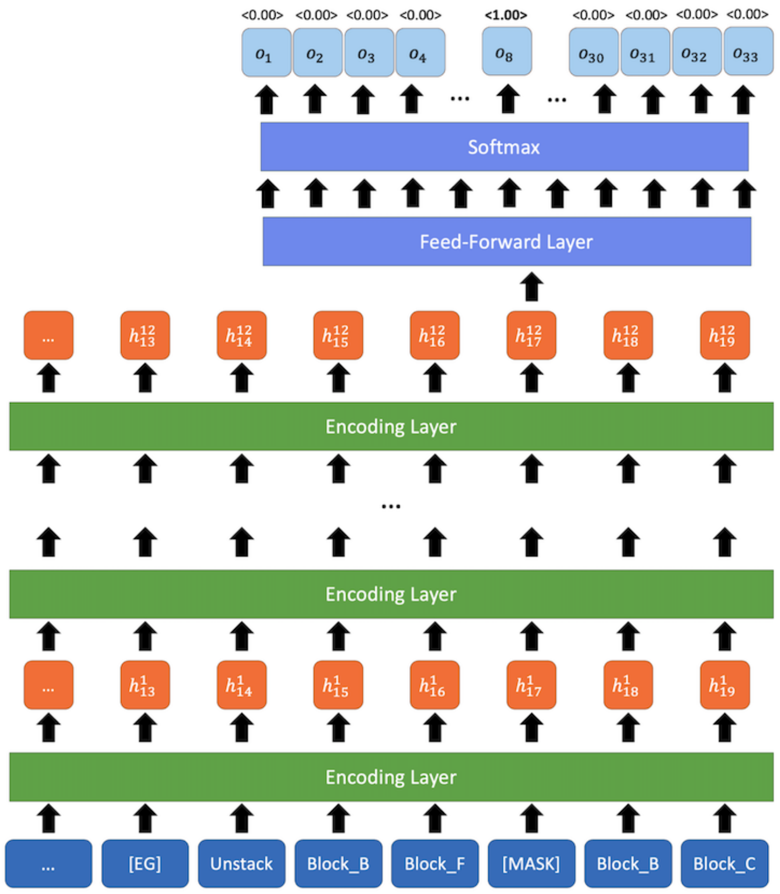

<0.00> <0.00> <0.00> <0.00> <1.00> <0.00> <0.00> <0.00> <0.00>
o_1 o_2 o_3 o_4 o_8 o_{30} o_{31} o_{32} o_{33}
Softmax
Feed-Forward Layer
… h_{13}^{12} h_{14}^{12} h_{15}^{12} h_{16}^{12} h_{17}^{12} h_{18}^{12} h_{19}^{12}
Encoding Layer
Encoding Layer
… h_{13}^{1} h_{14}^{1} h_{15}^{1} h_{16}^{1} h_{17}^{1} h_{18}^{1} h_{19}^{1}
Encoding Layer
… [EG] Unstack Block_B Block_F [MASK] Block_B Block_C

Figure 6.3: Schematic representation of a BERT model performing the PLM task

for a 1 at index 8. Therefore, the result produced by the feed-forward layer is compared with the desired output. This procedure is repeated for every training instance, evaluating the overall error made by the architecture and calculating a loss function, that is *cross-entropy* in our experiments, which will be used by the backpropagation algorithm to adjust the model weights.

In Figure 6.3 we show a simplified representation of the model architecture performing the planning language modeling task. The tokens are encoded into vectors by several encoding blocks and, in the last step, the vectorial representation of [MASK] is passed

Domain	Training Set Size	Vocabulary Size	Avg. Tokens	Avg. Actions
BLOCKSWORLD	200k	40	171.3	39.2
DEPOTS	192k	50	243.2	32.9
DRIVELOG	239k	84	204.5	30.9
LOGISTICS	225k	66	187.8	28.8
SATELLITE	154k	83	114.3	17.5
ZENOTRAVEL	252k	41	170.8	22.1

Table 6.1: Number of training plans, number of tokens in the vocabulary, average number of input tokens (Avg. Tokens) and average plan length (Avg. Actions) of the plans in the training set for each considered domain.

to a feed-forward layer with softmax activation function that predicts a 1 at the index 8 and 0 for all the other indexes.

6.2 Benchmark Suite and Datasets

For our experiments, we consider six benchmark domains: BLOCKSWORLD, DEPOTS, DRIVELOG, LOGISTICS, SATELLITE and ZENOTRAVEL; these domain are described in more detail in Section 4.4.

6.2.1 Training Sets

Concerning the training set, for each domain, we randomly generated a large collection of (solvable) plan generation problems of different size. We consider the same ranges of objects used for the GRNET experimental evaluation (see Table 4.2). As we did in Section 4.4, for each of these problems, we computed up to four (sub-optimal) plans solving them using LPG. The generated training set consists of input sequences $\langle I, G, \pi \rangle$ where I is the initial state of the problem, G is the goal and π is a plan that leads from I to G. Table 6.1 reports the number of training plans, the number of considered tokens, the average number of token that the model receives as input and the average length of training plans for each domain.

Additionally, we designed a data set for fine-tuning the model to perform goal recognition. For each domain we randomly selected 25k plans. From these plans, we derived the observation sequences for the fine-tuning samples by randomly selecting actions from the plans (preserving their relative order). The selected actions are between 30% and 70% of the plan actions. The generated data set consists of pairs $(\langle I, O \rangle, G^*)$ where I is the initial state of the planning problem, O is a sequence of observed actions obtained by sampling a plan π, and G^* is the hidden goal corresponding to the goal of the planning problem solved by π. Please note that we reduced the amount of training data from 55k plans used to train GRNet to 25k because we are taking advantage of the information learned by the BERT model during its training phase.

6.2.2 Evaluation Tasks and Test Sets

For each domain, we evaluate the capabilities of our model using two test sets: one used for the evaluation of PLM, NTP and PTP tasks and the other one for the goal recognition task.

The first test set is composed by 2k plans which were not used during training. In the planning language modeling task, we check whether the predictions of the masked input tokens are correct. Given that the softmax activation function of the output layers actually predicts a score between 0 and 1 for all the possible tokens of our domains, we evaluate the planning language masked in two configurations. We consider only the token predicted with the highest score. The planning language modeling (PLM) task is evaluated using the following percentages of masked tokens: 10%, 15%, 25% and 50%.

In the second task, called **Next Token Prediction** (NTP), we ask our model to predict the token that follows an input token sequence. In detail, given a token sequence of length l, for (NTP), we pass the first *l-1* tokens to the model and ask the model to predict the l-th token. Similarly, for the third task, called **Previous Token Prediction** (PTP), the model should predict the first token of the sequence while the remaining token are passed in input. We evaluate NTP and PTP using different token sequence lengths (i.e. 5, 10, 25, 50) and with the entire plan without the last and first token, respectively (Tot). Please note that the initial state and the goal are always included in the input string; however, the token sequence length is computed counting only the tokens that belong to the plan (i.e. after the [EG] token).

For the goal recognition evaluation, we use $TS_{LGR\mathit{Gen}}$, introduced in Section 4.4. This test set contains observations derived from plans computed by LAMA and it is an extension of TS_{LGR}; thus it has a richer diversification of the observation traces and a larger number of test instances. In our experiments, we consider action traces formed by 10%, 30%, 50%, 70% and 100% of the plan actions, respectively and use GR accuracy as evaluation metric.

6.3 Experimental Results

We experimentally evaluate BERT on the selected tasks. On the goal recognition task, we compare the obtained results with the ones obtained by GRNET.

6.3.1 Planning Language Modeling Task

In Table 6.2, we show the results for all the considered domains in terms of accuracy, of the planning language modeling task. In this experiment, we masked different percentages of tokens: 10%, 15%, 25% and 50%. As it should be expected, the accuracy decreases proportionally to the increasing of the percentage of masked actions (i.e. less input data).

The overall performance in this task is very good; in particular we can notice that, for all the considered masking percentages, the accuracy is always greater than 90%,

Masked tokens (%)	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
10	99.47	99.38	98.99	98.53	96.10	99.54
15	99.36	99.22	98.82	98.31	95.95	99.49
25	98.90	98.88	98.00	97.45	95.24	99.05
50	96.14	96.62	94.23	92.40	90.08	95.66

Table 6.2: Results for the Planning Language Modeling task in terms of accuracy. On the rows, we report the percentage of masked actions in the test sequences.

which is remarkable. In BLOCKSWORLD, DEPOTS and ZENOTRAVEL we obtain the highest performance with an accuracy greater than 95% even when half of the input token are masked. The relatively low performance in SATELLITE can be due to the lower number of training plan (i.e. 50k plans less than the other domains) and the short length of the generated plans which lead to less overall training samples.

6.3.2 Next Token Prediction Task

The results for the NTP task in terms of accuracy are shown in Table 6.3. In each row, we consider an input sequence of different length. For instance, if the length is 5 we have the inital state, the goal state and the first 5 tokens of the plan as input and we want to predict the 6-th token.

The results obtained on the considered domains are good. In each column, we can notice the same trend: the accuracy increases with the length of the plan with the exception of BLOCKSWORLD and DEPOTS in which the results for 5 input tokens are higher then the results with 10. This may be due to the fact that the very first possible actions in the plans of these two domains are limited (e.g. unstack the blocks in BLOCKSWORLD).

We can also see that, in all considered domains except SATELLITE, the performance noticeably increase of almost 15 point increasing the input sequence from 50 to 100 to-

Sequence Length	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
5	99.9	84.85	36.70	21.40	45.60	58.80
10	80.85	67.55	68.10	32.00	56.45	51.85
25	82.30	74.10	71.80	70.50	74.55	67.10
50	81.65	74.30	80.60	76.80	78.65	75.55
100	94.28	90.95	85.00	90.95	79.40	89.10
Tot	100.0	100.0	100.0	100.0	100.0	100.0

Table 6.3: Results for the NTP task. Sequence Length stands for the number of input tokens; initial state and goal tokens are not considered. In row *Tot* we pass the entire plan to the model except for the last token.

kens. This happens because BERT can exploit the sensible increase of input information to infer the final token. In fact, we can see that in the *Tot* row, where we pass the entire plan to the model except for the last token, we obtain 100% accuracy in all the domains. Regarding the performance in the SATELLITE domain with 50 input tokens, we think that the little increase in performance w.r.t. 25 input token is reflected by the lower performance obtained on this domain on the PLM task, which is an easier task compared to NTP.

6.3.3 Previous Token Prediction Task

In Table 6.4 we report the results for the PTP task in terms of accuracy using the same format used in the evaluation of the NTP task.

The results on this task are remarkable; in particular, we can notice that by considering only 5 input tokens, we manage to obtain an accuracy higher than 95% in almost all the considered domains. Similarly to the NTP results, we can notice that, for each column, the accuracy increases as the input length increases. Looking at the 10 input

Sequence Length	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
5	100.0	99.95	98.25	99.05	99.4	83.5
10	100.0	100.0	99.10	99.15	99.4	99.75
25	100.0	100.0	100.0	99.85	99.4	99.95
50	100.0	100.0	100.0	100.0	100.0	100.0
100	100.0	100.0	100.0	100.0	100.0	100.0
Tot	100.0	100.0	100.0	100.0	100.0	100.0

Table 6.4: Results for the PTP task. Sequence Length stands for the number of input tokens; initial state and goal tokens are not considered. In row *Tot* we pass the entire plan to the model except for the first token.

tokens row, we can see that in all the domains the accuracy is higher than 99%; from 50 input tokens onward, the accuracy is 100% for all the considered domains.

The fact that the performance are so high is related to the fact that, among all the designed tasks, PTP is the easier to solve for BERT. In fact, BERT processes the whole sequence together and can predict the masked token using the information that comes after it, which is often more informative.

6.3.4 Goal Recognition Task

Table 6.5 summarizes the performance results of GRNET and the BERT model fine-tuned on the goal recognition task in terms of GR accuracy on $TS_{LGR\mathit{Gen}}$. We can see that BERT performs generally well and improves its performance with the increase of the percentage of the observed actions. In DEPOTS, BERT obtains always better performance w.r.t. GRNET for all the considered percentages of observed actions; in particular, with 30, 50 and 70% of the observation, BERT improves the results by over 10 points. Similar results can be seen in DRIVELOG, and ZENOTRAVEL. However, in SATELLITE GRNET always

Plan %	*Model*	BLOCKS	DEPOTS	DRIVELOG	LOGISTICS	SATELLITE	ZENO
10	GRNET	22.85	32.95	**39.80**	**39.15**	**45.50**	**48.00**
	BERT	**23.00**	**36.07**	38.87	32.29	44.13	41.76
30	GRNET	52.35	59.80	65.40	**68.80**	**75.10**	76.90
	BERT	**56.31**	**74.81**	**75.40**	66.84	71.93	**81.14**
50	GRNET	71.10	74.70	78.40	80.40	**88.30**	89.20
	BERT	**78.06**	**89.27**	**91.11**	**82.17**	84.73	**96.64**
70	GRNET	84.90	84.90	86.20	89.20	**96.10**	96.80
	BERT	**89.70**	**95.19**	**96.94**	**91.17**	91.63	**99.67**
100	GRNET	**92.02**	91.95	90.78	**93.94**	**98.73**	98.73
	BERT	87.01	**96.39**	**98.21**	93.30	93.63	**99.99**

Table 6.5: Goal recognition accuracy (% of GR instances correctly predicted) by GRNET and BERT with test set $TS_{LGR\mathit{Gen}}$.

outperforms BERT; as previously shown in the MLM and NTP tasks, the performance on this domain were lower, compared to the other domains and this reflects also in the performance of the fine-tuned model.

6.4 Discussion

The results presented in Section 4.5 show that a deep learning model such as BERT, trained on a set of action sequences from well-known planning domains, can effectively accomplish three predictive tasks: the planning language modeling, i.e. predicting actions from context, the prediction of the next action given sequences of different length and, in the same conditions, the prediction of the previous action. We also shows that it is possible to fine-tune the BERT model to solve a different task, like we did with goal recognition.

In our opinion, this demonstrates that BERT is able to understand how a planning domain works and which rules are followed by the action sequences. However, we are also conscious of several limits. For instance, as we show in Table 6.1, the number of plans necessary to train such a model is very high. In a real-world applications, this can be a huge problem because it would require a long time to collect the necessary data to exploit this kind of model. Another related problem with this kind of architectures is the training time and the computational resources required. An interesting line of research could be apply knowledge distillation techniques and build smaller versions of our model [36].

Moreover, as we saw with goal recognition, BERT can be adapted for other tasks such as document classification. However, in natural language processing, Devlin et al. [17] introduced a special token called `[CLS]` for representing the entire document. In our context, we could have the same token for obtaining a vector representation of the entire plan and use it for other tasks such as heuristic prediction. However, training `[CLS]` in BERT requires another task, called Next Sentence Prediction, into which the model learns the ability to predict whether two sentences are consecutive in a document. Although this is a very intuitive operation in NLP, the same concept is not present in automated planning. Thus, there is the need to design a completely new task which could lead the model to learn an informative representation of a sequence of actions.

Chapter 7

Conclusions and Future Works

In this thesis we described the main research activities carried out during my three years of PhD and their experimental results. In the context of goal recognition, the main contribution of this thesis is to develop architectures based on deep learning that can resolve goal recognition tasks faster w.r.t. automated planning based approaches without losing accuracy. To do so, we designed three novel approaches where the goal recognition problem is formulated as a classification task.

The first system we presented, called GRNet, learns to solve goal recognition tasks from past experience in a given domain. The learning process consists in training only one LSTM network for the considered domain, allowing to solve a large collection of GR instances in the domain by the same trained network. Moreover, GRNet can be effectively integrated with the state-of-the-art model-based system LGR. The experimental analysis shows that GRNet and LGRNet, our system integrating GRNet and LGR, perform generally well for the considered benchmark domains, on all the tested metrics that are accuracy, θ-accuracy and spread.

FSGR the second presented architecture, introduces a novel approach to goal recognition which seamlessly integrates both intuitive and deliberative reasoning techniques. By proposing a dual-process model, FSGR harnesses the power of fast, intuitive recognition for immediate goal identification, while employing slow, deliberate analysis for deeper understanding. This unique combination leverages deep learning techniques and planning-based reasoning, effectively modeling the dual-process system. The experimental evaluation of this system demonstrates improved accuracy and robustness w.r.t. the state-of-the-art system, especially in complex scenarios. This approach bridges the gap between rapid inference and deep understanding, paving the way for advanced and proficient systems.

Finally, as a third approach, we trained a BERT model in the context of automated planning. We designed an adaption of the masked language modeling task, called *planning language modeling* into which the model learns how to identify masked action tokens from context. We have also presented a detailed account of the operations necessary to train such a model and show how this model can be fine-tuned to solve goal recognition tasks. In the experimental evaluation, our models obtains very promising results and we show that they have a high accuracy in all the analyzed tasks. Regarding the goal recognition task, the fine-tuned BERT performs generally well for the considered benchmarks and often achieves better results w.r.t GRNET.

Differently from model-based approaches, the GR instances addressable by the presented architectures are limited to those involving subsets of fluents and actions that are used during the training. If the GR instance to solve involves a new fluent, clearly such a fluent cannot be predicted; similarly, if the instance involves a new action, such an action cannot be part of the input observed actions. An interesting question for future work is how to extend these architectures to solve GR instances involving new actions and fluents; this can be performed considering the object names involved in the GR instance to solve, and defining a mapping with the object names of instances considered in the training phase. Additionally, we believe that it is possible to further exploit the integration between model-based and learning based approaches and that this is an interesting direction for future work. Furthermore, please consider that the BERT-based approach presented in Chapter 6 is a first step towards the use of BERT in the context of automated planning. It can be further analyzed and tested using different benchmarks and different training and testing tasks. Finally, we would like to test and extend our approaches also when dealing with real-time goal recognition (also known as Online Goal Recognition).

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD*, pages 2623–2631, 2019.

[2] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey. *CoRR*, abs/1901.09069, 2019.

[3] Leonardo Amado, João Paulo Aires, Ramon Fraga Pereira, Mauricio Cecilio Magnaguagno, Roger Granada, and Felipe Meneguzzi. LSTM-Based Goal Recognition in Latent Space. *CoRR*, abs/1808.05249, 2018.

[4] Leonardo Amado, Gabriel Paludo Licks, Matheus Marcon, Ramon Fraga Pereira, and Felipe Meneguzzi. Using Self-Attention LSTMs to Enhance Observations in Goal Recognition. In *Proceedings of IJCNN 2020*. IEEE, 2020.

[5] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 6094–6101. AAAI Press, 2018.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Ligia Batrinca, Nadia Mana, Bruno Lepri, Nicu Sebe, and Fabio Pianesi. Multimodal

Personality Recognition in Collaborative Goal-Oriented Tasks. *IEEE Transactions on Multimedia*, 18(4), 2016. doi: 10.1109/TMM.2016.2522763.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3, 2003.

[9] Luigi Bonassi, Alfonso Emilio Gerevini, and Enrico Scala. Planning with qualitative action-trajectory constraints in PDDL. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4606–4613. ijcai.org, 2022.

[10] Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jonathan Lenchner, Nick Linck, Andreas Loreggia, Keerthiram Murgesan, Nicholas Mattei, Francesca Rossi, and Biplav Srivastava. Thinking fast and slow in AI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 15042–15046, 2021.

[11] Daniel Borrajo, Sriram Gopalakrishnan, and Vamsi K. Potluru. Goal recognition via model-based and model-free techniques. *Proceedings of FinPlan 2020*, 2020.

[12] Sandra Carberry. Techniques for plan recognition. *User Model. User Adapt. Interact.*, 11(1-2):31–48, 2001.

[13] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[14] Mattia Chiari, Alfonso Emilio Gerevini, Andrea Loreggia, Luca Putelli, and Ivan Serina. Fast and Slow Goal Recognition. In *AAMAS 2024 (Accepted).*

[15] 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 *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 560–568, 2023.

[16] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Carmel Domshlak, Malte Helmert, Erez Karpas, Emil Keyder, Silvia Richter, Gabriele Röger, Jendrik Seipp, and Matthias Westphal. Bjolp: The big joint optimal landmarks planner. 2011.

[19] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[20] Patrick Ferber, Malte Helmert, and Jörg Hoffmann. Neural network heuristics for classical planning: A study of hyperparameter space. In *ECAI 2020*, pages 2346–2353. IOS Press, 2020.

[21] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.

[22] M Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Francesca Rossi, Biplav Srivastava, and Kristen Brent Venable. Thinking fast and slow in ai: The role of metacognition. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 502–509. Springer, 2022.

[23] Hector Geffner. Model-free, Model-based, and General Intelligence. In *Proceedings of IJCAI 2018*, 2018.

[24] Christopher Geib and David Pynadath. Plan, activity, and intent recognition. *AI Magazine*, 28(4):124, 2007.

[25] Christopher W. Geib and Mark Steedman. On natural language processing and plan recognition. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1612–1617, 2007.

[26] Alfonso Gerevini and Ivan Serina. Planning as propositional CSP: from walksat to local search techniques for action graphs. *Constraints An Int. J.*, 8(4):389–413, 2003.

[27] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *J. Artif. Intell. Res.*, 20:239–290, 2003.

[28] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[29] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[32] Mickel Hoang, Oskar Alija Bihorac, and Jacobo Rouces. Aspect-based sentiment analysis using bert. In *Proceedings of the 22nd nordic conference on computational linguistics*, pages 187–196, 2019.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[34] Jörg Hoffmann. Ff: The fast-forward planning system. *AI Mag.*, 22:57–62, 2001. URL https://api.semanticscholar.org/CorpusID:9968823.

[35] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res.*, 22:215–278, 2004.

[36] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4163–4174. Association for Computational Linguistics, 2020.

[37] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[38] Daniel Kahneman. *Thinking, Fast and Slow.* Macmillan, 2011.

[39] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[40] Derek Long and Maria Fox. The 3rd International Planning Competition: Results and Analysis. *J. Artif. Intell. Res.*, 20, 2003.

[41] Mariane Maynard, Thibault Duhamel, and Froduald Kabanza. Cost-Based Goal Recognition Meets Deep Learning. *Proceedings of PAIR 2019*, 2019.

[42] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998. URL `https://api.semanticscholar.org/CorpusID:59656859`.

[43] Drew V. McDermott. The 1998 AI Planning Systems Competition. *AI Mag.*, 21(2): 35–55, 2000.

[44] Felipe Meneguzzi and Ramon Fraga Pereira. A Survey on Goal Recognition as Planning. In *Proceedings of IJCAI 2021*, 2021.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[46] Wookhee Min, Bradford W. Mott, Jonathan P. Rowe, Barry Liu, and James C. Lester. Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks. In *Proceedings of IJCAI 2016*. IJCAI/AAAI Press, 2016.

[47] Reuth Mirsky, Roni Stern, Ya'akov (Kobi) Gal, and Meir Kalech. Sequential Plan Recognition. In *Proceedings of IJCAI 2016*. IJCAI/AAAI Press, 2016.

[48] Reuth Mirsky, Ya'ar Shalom, Ahmad Majadly, Kobi Gal, Rami Puzis, and Ariel Felner. New Goal Recognition Algorithms Using Attack Graphs. In *CSCML 2019, Proceedings*, volume 11527. Springer, 2019.

[49] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318, 2013.

[50] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[51] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-based approaches for goal recognition as planning. *Artif. Intell.*, 279, 2020.

[52] Luca Putelli, Alfonso Emilio Gerevini, Alberto Lavelli, Tahir Mehmood, and Ivan Serina. On the behaviour of bert's attention for the classification of medical reports. In Cataldo Musto, Riccardo Guidotti, Anna Monreale, and Giovanni Semeraro, editors, *Proceedings of the 3rd Italian Workshop on Explainable Artificial Intelligence co-located with 21th International Conference of the Italian Association for*

Artificial Intelligence(AIxIA 2022), Udine, Italy, November 28 - December 3, 2022, volume 3277 of *CEUR Workshop Proceedings*, pages 16–30. CEUR-WS.org, 2022. URL http://ceur-ws.org/Vol-3277/paper2.pdf.

[53] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1133–1136, 2019.

[54] Colin Raffel and Daniel PW Ellis. Feed-forward networks with attention can solve some long-term memory problems. *arXiv preprint arXiv:1512.08756*, 2015.

[55] Miquel Ramírez and Hector Geffner. Plan Recognition as Planning. In *Proceedings of IJCAI 2009*, 2009.

[56] Miquel Ramírez and Hector Geffner. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In *Proceedings of AAAI 2010*. AAAI Press, 2010.

[57] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

[58] Gabriele Röger, Florian Pommerening, and Malte Helmert. Optimal planning in the presence of conditional effects: Extending lm-cut with context-splitting. 2014.

[59] Luísa R. A. Santos, Felipe Meneguzzi, Ramon Fraga Pereira, and André Grahl Pereira. An LP-Based Approach for Goal Recognition as Planning. In *Proceedings of AAAI 2021*. AAAI Press, 2021.

[60] Lorenzo Serina, Mattia Chiari, Alfonso E Gerevini, Luca Putelli, Ivan Serina, et al. A preliminary study on bert applied to automated planning. In *CEUR WORKSHOP PROCEEDINGS*, volume 3345. CEUR-WS, 2022.

[61] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. Plan Recognition as Planning Revisited. In Subbarao Kambhampati, editor, *Proceedings of IJCAI 2016*. IJCAI/AAAI Press, 2016.

[62] Franz A. Van-Horenbeke and Angelika Peer. Activity, plan, and goal recognition: A review. *Frontiers Robotics AI*, 8, 2021.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

[64] Naveen Venkat. The curse of dimensionality: Inside out, 2018.

[65] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers Robotics AI*, 2:28, 2015.

[66] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[67] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical Attention Networks for Document Classification. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of NAACL HLT 2016*, 2016.