Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

NatBDI: Combining BDI Reasoning and Natural Language Inference for Autonomous Agents

Alexandre Yukio Ichida
Pontifical Catholic University of Rio Grande do Sul
Felipe Meneguzzi
felipe.meneguzzi@abdn.ac.uk
University of Aberdeen
Rafael C. Cardoso
University of Aberdeen

Research Article

Keywords: belief-desire-intention, natural language processing, machine learning, autonomous agents, text-based environments

Posted Date: April 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8990521/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

NatBDI: Combining BDI Reasoning and Natural Language Inference for Autonomous Agents

Alexandre Yukio Ichida[1*], Felipe Meneguzzi[1,2†] and Rafael C. Cardoso[2†]

[1]Polytechnic School, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, 90619-900, Rio Grande do Sul, Brazil.
[2]Department of Computing Science, University of Aberdeen, Aberdeen, AB11 7DF, United Kingdom.

*Corresponding author(s). E-mail(s): yukioichida@gmail.com;
Contributing authors: felipe.meneguzzi@abdn.ac.uk;
rafael.cardoso@abdn.ac.uk;
†These authors contributed equally to this work.

Abstract

Developing autonomous agents to deal with real-world problems is challenging, especially when developers are not necessarily specialists in artificial intelligence. Recent advances in machine learning to address natural language processing tasks are reaching performance levels suitable for practical applications, although these approaches rely on opaque and inscrutable models. This poses three key challenges: the interface of the programming with the developer, the efficiency of the resulting agents, and the scrutiny of their behaviour.

Purpose: We tackle the challenge of developing autonomous agents over natural language environments in an efficient agent architecture that leverages recent developments in natural language processing, and the intuitive folk psychology abstraction of the beliefs, desires, intentions (BDI) architecture.

Methods: This article introduces NatBDI, a new class of agent architecture that uses the BDI reasoning cycle with components driven by natural language processing. The resulting architecture handles natural language environments using a combination of language models and natural language inference to bootstrap the agent's reasoning processing.

Results: NatBDI agents leverage natural language components based on mental attitudes, enhancing intuitive understanding of the agent's mental state. This allows a developer to instruct the agent more directly using a combination of

controlled natural language structure and natural language knowledge as its programming interface. We empirically assess the efficiency gains of this combination while introducing a more intuitively programmed autonomous agent. Instructions in this interface substantially improve agent performance in the experimental scenario over a baseline agent created using a pure machine learning approach.
Conclusion: The resulting architecture shows that combining learned policies with intuitively engineered domain knowledge yields substantial performance gains. We expect this class of agents to provide a powerful, yet intuitive, tool for agent-driven programming.

Keywords: belief-desire-intention, natural language processing, machine learning, autonomous agents, text-based environments

1 Introduction

The widespread deployment of Artificial Intelligence (AI) systems in applications that interact directly with people introduces two central challenges for the design of autonomous agents [1–3]. On the one hand, such agents must be capable of interpreting human commands and producing responses that users can readily understand. On the other hand, developers need mechanisms to ensure that the agent's behaviour remains appropriate, even when training data is limited or uneven. Contemporary AI applications often rely on natural language as a medium for communicating with users during everyday tasks [4]. Consequently, agents must process human-provided natural language inputs to guide their decisions, which may involve generating textual responses or executing action sequences (e.g., plans). To promote safety and reduce the likelihood of unintended behaviour [3], these systems benefit from an interpretable mental model that enables scrutiny of the agent's reasoning and clarifies why particular decisions are made [5, 6].

Effectively handling natural language is essential for autonomous agents that aim to communicate and collaborate with humans. Yet, interpreting human language remains a complex computational problem because natural language is inherently unstructured and often ambiguous [7]. To provide meaningful assistance, an agent must be able to reason over linguistic input, understand the situations described by users, and select actions that remain intelligible to humans. In natural language environments, this involves analysing textual observations, identifying the relevant facts conveyed in the perceived descriptions, and determining which actions or plans should follow [8, 9]. Such decision-making depends on context-sensitive reasoning, allowing the agent to assess whether a specific action or plan is appropriate for the circumstances it currently perceives.

Recent deep learning approaches for natural language processing, such as Large Language Models (LLMs), achieve exceptional benchmark results when using textual information for classification and generation tasks [10]. One of the key differences between such models and previous machine learning approaches is that an LLM learns word representations that can be reused in other NLP tasks as fine-grained input features, which speeds up the training convergence [11–14]. LLMs use large amounts

of natural language data, capturing the semantic meaning of words grounded by the context of sentences to generate textual responses that satisfy user input [15, 16]. The LLM training process includes information from human feedback to refine the language model's outputs according to human preferences, reflecting how closely the LLM result matches the expected answer. To train on such vast amounts of information, LLMs consist of neural networks with a very large number of trainable parameters, which require significant and specialised computational resources.

Although LLMs leverage a massive volume of information in their training data, relying solely on such approaches to handle more complex reasoning tasks remains a challenge. Recent work [17, 18] corroborate such an argument, highlighting the limitations in problem-solving tasks that require multistep reasoning, such as common-sense planning [19–21] and logical consistency [18, 22] over natural language information [23]. In fact, the LLM's reasoning process is opaque [24] because such approaches rely entirely on black-box models, and hence understanding the agent's decisions remains difficult. An explicit mental model representation of the agent's behaviour is essential for understanding these reasoning limitations and addressing them.

Traditional AI approaches frequently draw on concepts from folk psychology to structure decision-making, using human-inspired mental attitudes to define rational agent behaviour [25–27]. Within this line of work, the Belief–Desire–Intention (BDI) model [28, 29] provides a well-established framework in which an agent's beliefs encode what it knows, its desires specify the states it aims to achieve, and its intentions capture the commitments it adopts in pursuit of those aims. Incorporating human-authored plans into such architectures offers a practical route toward more predictable and controllable autonomous agents, helping them avoid undesirable or unsafe behaviours [3]. Plans written in natural language enable designers to embed prior knowledge directly, without relying solely on machine learning methods that may require retraining or additional data curation. By specifying an ordered set of steps for the agent to follow, the designer eliminates the need for exploration-based training procedures (e.g., ϵ-greedy reinforcement learning algorithms [30]), allowing the agent to execute the prescribed plan immediately.

Our contributions can be summarised in three parts. First, we represent the unstructured textual observations arising from human activities as natural language beliefs, and we construct a corresponding library of natural language plans for the agent. Second, we introduce an interpreter grounded in the BDI model whose reasoning back-end operates directly on natural language content. This component employs natural language inference to determine entailment relationships between the agent's belief base and the contextual conditions specified in its plan library, enabling context-sensitive plan selection. Third, when no plan-rule is deemed applicable, we integrate a fallback policy based on black-box reinforcement learning to allow the agent to act autonomously. To assess NatBDI's natural language reasoning and decision-making abilities, we conduct experiments in the simulated textual environments provided by the ScienceWorld benchmark [31]. Our findings show that incorporating even a modest number of natural language plans, combined with our adapted BDI reasoning cycle for textual domains, results in substantial performance improvements compared to agents that rely solely on machine learning.

In this article, we present an extended version of Ichida et al. work [1], where we summarize the differences and the additional content in the following points: First, we provide more details of the NATBDI components, describing the agent interpreter's internal algorithms and event flow in depth. Second, we include additional experiments by including a newer reinforcement learning approach as an alternative fallback policy implementation. This fallback policy integrates the most recent language model to improve the decision-making capabilities when no plan-rules have been provided, demonstrating the NATBDI architecture generalisability. Finally, the related work section has been updated to include recent literature covering BDI agent implementations in other natural language domains.

2 Background

First, we detail concepts of logical inferences over natural language in Section 2.1. Second, we present the main concepts of the BDI model, which is a central topic of this work in Section 2.2. Finally, we describe the environment we use in this work, providing details about modelling approaches and examples of existing environments in Section 2.3.

2.1 Natural Logic and Inference

In this section, we provide a brief overview of methods for identifying logical relations in natural language. We leverage these concepts since NatBDI represents its mental state entirely in natural language. We introduce a formal system for logical inference in natural language in Section 2.1.1, and describe a method for performing natural language inference, focusing on informal reasoning in Section 2.1.2.

2.1.1 Natural Logic

Natural Logic is a formal system that describes logical inferences over natural language information [32]. Precisely, natural logic aims to develop a method for handling logical inference in natural-language sentences that is close to that of first-order logic systems. Given that most human reasoning takes place in natural language and most uses of natural language express reasoning of some sort, this model aims to create a correspondence between logical and grammatical structure [33]. Although Lakoff developed the main concepts of Natural Logic [33], Aristotle also used a similar approach to introduce syllogisms through logical arguments expressed in natural language.

Syllogistic reasoning from natural logic handles simple sentences without concerning linguistic syntax but focusing on semantic consequences [32]. Formally, a hypothesis sentence s is a semantic consequence of a set of premise sentences Γ if every model of all sentences in Γ is also a model of s considering the semantic relation. That is, the semantic consequence represents an entailment relation between the sentence s with Γ induced by the linguistic semantics. The Example 1 shows an entailment relation between the sentence set Γ representing the premise (above the line) and the hypothesis sentence s. This example illustrates the transitivity property that the verb *is* conveys: assuming that "every skunk is a mammal" and "every mammal is an

animal" are true (i.e., the premises), then "every skunk is an animal" is a semantic consequence of both sentences.

$$\frac{\begin{array}{l}\text{Every skunk is a mammal}\\ \text{Every mammal is an animal}\end{array}}{\text{Every skunk is an animal}} \tag{1}$$

$$\frac{\begin{array}{l}\forall x\,(\text{skunk}(x) \rightarrow \text{mammal}(x))\\ \forall x\,(\text{mammal}(x) \rightarrow \text{animal}(x))\end{array}}{\forall x\,(\text{skunk}(x) \rightarrow \text{animal}(x))} \tag{2}$$

Notice that, as this example of syllogistic reasoning from natural logic uses sentences that are syntactically very simple, we can also represent this entailment relation in first-order logic language, which we detail in Example 2. Sentences from natural logic lack individual variables, comprising only instantiated terms (e.g., skunk, mammal, animal) to make logical inference. In contrast, first-order logic statements can rely on lifted variables such as x in this example. In Section 6.3, we survey related work that introduce natural logic system approaches to reason over natural logic, such as monotonicity calculus [34] and semantic exclusion [35].

2.1.2 Natural Language Inference

Automated reasoning and inference are essential topics of artificial intelligence [36]. Natural Language Inference (NLI) is a widely studied natural language processing task concerned with determining the inferential relation between a premise p and a hypothesis h [37]. In contrast to the first-order logic system, which focuses on formal methods for inference, this task addresses logical inference through more informal reasoning that encompasses linguistic aspects such as semantic knowledge and the variability of linguistic expressions [35, 38].

Recent advances in machine learning methods for natural language processing tasks introduce a wide range of approaches that leverage large text corpora to develop NLI methods [11, 13, 39]. In general, machine learning NLI approaches rely on annotated datasets [37, 39, 40] to capture the semantic aspects of sentences to infer entailment relations. That is, such approaches develop a natural logic system (recall Section 2.1.1) that infers whether a sentence pair (p, h) entails a semantic consequence by learning linguistic patterns from a large dataset. In the field of machine learning, NLI includes two different representations of classification tasks: two-way classification and three-way classification [41].

Two-way classification is the simplest way to represent the machine learning task of NLI, treating it as a binary decision. The objective of this NLI task is to classify whether the hypothesis follows the premise (entailment) or does not (non-entailment). This form of classification was used in the Recognising Textual Entailment (RTE) competition [42]. This task representation cannot detect a contradiction between p and h due to the reason that the inference of a non-entailment does not specify the relation that differs from an entailment.

Alternatively, the three-way classification addresses contradiction by creating an additional category of inferred relations. In three-way classification, the relations consist of three categories: entailment, contradiction and neutral. Given a pair of

premise-hypothesis p and h, the *entailment* relation occurs when h can be inferred from p [41]. When h infers the negation of p, the pair results in a *contradiction*. Otherwise, if none of these relations can be inferred, the relation of p and h is *neutral*.

In NLI, both p and h are sentences written in natural language. The challenge of this task differs from formal deduction in classical logic due to the need to deal with informal reasoning [41]. The emphasis of the NLI is on aspects of natural language such as lexical semantic knowledge and dealing with the variability of linguistic expression. Consider the following premise p and hypothesis h as an instance of an NLI scenario [41]:

- p: Several airlines polled saw costs grow more than expected, even after adjusting for inflation.
- h: Some of the companies in the poll reported cost increases.

In the NLI context, this example is considered a valid entailment inference because any person who interprets p would likely accept that h implies in the information of p. Although it is a valid NLI classification, h is not a strict logical consequence of p due to the fact that p informs that airline companies *saw* the growth of the cost, not necessarily *reporting* the growth of the cost. This example reflects the informal reasoning of the task definition due to the ambiguity of natural language [41].

2.2 BDI Model

The Belief-Desire-Intention (BDI) model is a framework for developing autonomous rational agents that are bound to mental attitudes in its architecture [43]. Inspired by Bratman's philosophical work [44], the BDI framework introduces a practical reasoning approach comprising three basic components: beliefs, desires, and intentions. BDI inherits two generic components from traditional agent architectures [45, 46], namely sensors that translate information from the external world into an internal representation, and actuators that translate the agent's actions into concrete results in the environment in which the agent is embedded. The reasoning cycle of traditional BDI agents consists of two steps: deliberation and means-end reasoning [29, 47]. The deliberation step defines which state the agent should achieve, and afterwards, the means-end step defines how to achieve it. More concretely, the agent reasons over its beliefs about the environment and adopts a plan from the plan library to achieve its intended desires (i.e., goals). This reasoning cycle depends on the agent developer's effort to program plans by anticipating and managing all possible agent interactions over the environment. We illustrate a variant of the BDI agent architecture in Figure 1.

2.2.1 BDI Components

In this section, we detail the components and definitions that comprise the BDI model using the AgentSpeak language [48], a well-known language for programming BDI agents. Besides belief, desire and intention, we describe other structures involved in the agent's reasoning cycle, such as the plan library and the event queue. According to the definition formulated in Meneguzzi and de Silva [47] work, we formalise the BDI agent components as follows:

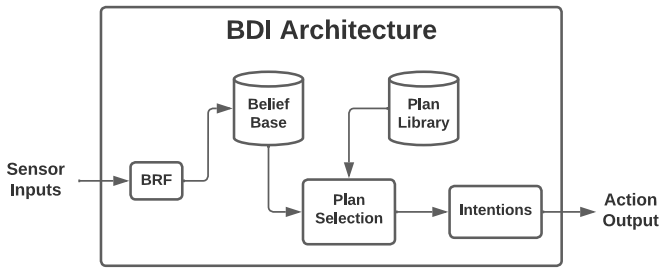


Fig. 1 Diagram illustrating a simple BDI agent architecture and its components.

Definition 1 (BDI Agent) A BDI agent is a tuple $\langle Ag, \mathcal{E}, \mathcal{B}, \mathcal{PL}, \mathcal{I} \rangle$, where Ag is the agent identifier; $\mathcal{E}$ is the queue of events; $\mathcal{B}$ is the belief base; $\mathcal{PL}$—the plan library—is a set of plan rules; and $\mathcal{I}$—the intention structure—is a set of intentions.

Belief represents information about what the agent understands of the world, based on its perceptions of the environment. In traditional BDI agents, beliefs encode information into grounded predicates represented by symbols defined by the agent developer. For instance, the ground predicate $in(glassjar, cupboard)$ expresses the belief that a glass jar is perceived to be inside a cupboard. During agent execution, the agent observes events from the environment and may add new beliefs or delete existing ones in a structure called the Belief Base, which stores all perceived beliefs. To maintain consistency in beliefs, the BDI architecture includes a function, the Belief Revision Function (BRF), that controls changes to the belief base.

Definition 2 (Beliefs and Belief Base) A belief φ is a grounded predicate that represents the agent's perceptions. The belief base $\mathcal{B}$ is a structure that stores a finite set of beliefs.

Desires represent states of affairs that the agent aims to achieve [43]. The agent can have different goals during its execution, and each one can trigger different plans to pursue them. In BDI languages such as AgentSpeak [48], there are two types of goals: achievement goals and test goals. Achievement goals consist of a true belief that represents the state the agent wants to achieve. Test goals are grounded predicates representing a formula that the agent wants to verify whether it is true according to its belief base.

In BDI, the reasoning cycle relies on an event-driven approach to manipulate the belief base and the adoption of goals and plans. This approach stores all events perceived during the reasoning process in a structure called *event queue*. The role of events in belief base maintenance is to control either inclusion of new beliefs (i.e., *belief addition* event), or deletion of existing ones (i.e, *belief deletion* event). Regarding plan adoption, the reasoning cycle uses triggering events to determine which plan to execute based on a goal addition or deletion. Formally, we follow Meneguzzi and De Silva [47] definition to describe events structure as below:

Definition 3 (Events) Given a predicate ϕ, an event e is either:

1. a belief addition expressed as $+\varphi$, with φ representing a belief added in belief base $\mathcal{B}$;
2. a belief deletion expressed as $-\varphi$, with φ representing a belief removed from belief base $\mathcal{B}$;
3. a goal addition expressed as $+!\varphi$, with φ representing a new goal for the agent to achieve;
4. a goal deletion expressed as $-!\varphi$, with φ representing a goal dropped by the agent;
5. a test goal addition expressed as $+?\varphi$, with $?\varphi$ representing a test goal is posted to the agent; or
6. a test goal deletion expressed as $+?\varphi$, with $?\varphi$ representing a test goal is dropped by the agent.

Implementations of BDI agents encode their behaviour as plan-rules consisting of two parts: the plan header, which describes when the plan can be executed; and the plan body, which contains a sequence of steps for the agent to perform. The plan header specifies the triggering event for the plan, along with the plan context, which describes the conditions under which it is considered a candidate option. The plan context comprises grounded predicates that represent a first-order formula over the agent's belief base. That is, to be considered a candidate option, the belief base should entail the plan context.

Definition 4 (Plan rules and plan library) A plan rule $\mathcal{P}$ is a tuple $\langle t, \mathcal{C}, bd \rangle$, where t is the triggering event to call $\mathcal{P}$; $\mathcal{C}$ is the context condition encoded in a first-order formula indicates whether a plan can be adopted as a candidate given the current belief base; and bd is the plan body consisting in the steps to achieve the goal. A plan library $\mathcal{PL}$ is a set of plan rules $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$, which the agent queries to find candidate plans.

The intention component is a structure consisting of a set of instantiated plans adopted by the agent to achieve a subset of its desires. This component is a stack that contains either actions the agent can execute directly in the environment or an event that triggers another plan. When an intention includes a goal-addition event, the agent executes another plan and decomposes its steps back onto the previous intention stack, allowing the development of plans in a hierarchical approach [49]. Agent developers implement plans to instruct the agent on how to achieve a goal by specifying a sequence of steps along with the context of their applicability, and store them in a component called Plan Library.

Definition 5 (Intention structure) An intention structure $\mathcal{I}$ is a set of intentions, and each intention is a tuple $\langle \sigma, st \rangle$ containing a substitution σ, and a set of steps st in a stack indicating the agent how to accomplish the goal.

2.2.2 BDI Interpreter

In this section, we describe a basic BDI interpreter and how the components from the previous section operate within the BDI reasoning cycle. Initially, the BDI agent perceives events from environment changes or goal definitions and adds them to its event queue. In cases where events indicate changes in the environment, the agent updates its belief base by checking which beliefs should be included (i.e., belief-addition event) or removed (i.e., belief-deletion event). To start acting on the environment, the agent receives its goal through events and selects a plan to achieve it. Given a candidate plan, the agent executes intentions from its plan body, which can generate additional events or trigger action execution. In Algorithm 1, we follow the Meneguzzi and De Silva [47] work to specify an abstract implementation of a BDI interpreter.

Algorithm 1 Abstract implementation of a BDI interpreter [47].

1: **procedure** BDIINTERPRETER($\langle Ag, \mathcal{E}, \mathcal{B}, \mathcal{PL}, \mathcal{I} \rangle$)
2: **while** True **do**
3: $\mathcal{E} \leftarrow updateEvents(\mathcal{E})$
4: $\mathcal{B} \leftarrow updateBeliefs(\mathcal{B}, \mathcal{E})$
5: $\mathcal{E}, \mathcal{I} \leftarrow selectPlans(\mathcal{E}, \mathcal{B}, \mathcal{PL}, \mathcal{I})$
6: $\mathcal{E}, \mathcal{I} \leftarrow executeIntentions(\mathcal{E}, \mathcal{I})$
7: **end while**
8: **end procedure**

The interpreter receives the BDI agent information as defined in Definition 1 to operate its components. The function to update events shown in Algorithm 2 gathers the perceived event, pushing it to the event queue $\mathcal{E}$ and returns its updated version ($\mathcal{E}$'). It assumes the existence of a function PERCEIVEEVENTS that provides perception encapsulated as events in some manner. The UPDATEBELIEFS function scans all events from $\mathcal{E}$ and, for each $e \in \mathcal{E}$, it verifies whether e represents a belief addition or belief deletion, which we describe in Algorithm 3. In the UPDATEBELIEFS function, the BDI interpreter either includes the belief predicate (φ) in the belief base (Line 5 of Algorithm 3) when a belief addition event is perceived or removes it (Line 7 of Algorithm 3) when a belief deletion event is perceived.

Algorithm 2 Function that updates the event queue given perceptions obtained from the environment [47].

1: **function** UPDATEEVENTS($\mathcal{E}$)
2: $e \leftarrow perceiveEvents$
3: $\mathcal{E}' \leftarrow push(e, \mathcal{E})$
4: **end function**

Algorithm 3 Function to update belief base according to the belief addition/deletion events perceived in event queue [47].

```
1: function updateBeliefs(ℰ, ℬ)
2:     ℬ′ ← ℬ
3:     for e ∈ ℰ do
4:         if e = +φ then
5:             ℬ′ ← ℬ′ ∪ {φ}
6:         else if e = −φ then
7:             ℬ′ ← ℬ′ − {φ}
8:         end if
9:     end for
10:    return ℬ'
11: end function
```

The selectPlans function (Algorithm 5) dequeues the next event from the event queue in order to find options in the plan library that satisfy the invocation condition. Given the plan $\mathcal{P}_i$ containing a triggering event t, and the plan context $\mathcal{C}$, if an event e unifies with t via substitution σ and the context $\mathcal{C}$ is entailed by the belief base $\mathcal{B}$, then $\mathcal{P}_i$ is a candidate option (Line 6 of Algorithm 4) for achieving the agent's goal [47]. These operations are illustrated in Algorithm 4, which details how the BDI interpreter acquires a list of plan candidates along with their substitution σ_{opt}.

In the BDI model, options are plans that satisfy the conditions set in their respective plan rules to become intentions, that is, plans that have its plan context entailed by the agent's belief base. When an option is queried from the plan library, the interpreter adds it to the intention structure. In cases where the event represents a test goal, if the belief base entails the belief being tested, the interpreter adds a new intention with the unifier of that belief test. When there is no option available in the plan library (i.e., no plans triggered by the event e), then the interpreter indicates a failure in the intention that generated the event by calling the intentionFailure function. We assume there is a function that handles the failure since this is an abstract implementation of a BDI interpreter and there is a wide option for handling BDI failures [26, 47].

Algorithm 6 describes the addIntention function used in selectPlans to include a new intention to the intention structure. Initially, it instantiates a new intention encapsulating the event e, the plan body bd, and the substitution σ_{opt} obtained from the unify function of event e with the plan invocation condition t mentioned before. After generating the new intention, the interpreter appends it into the original intention structure $\mathcal{I}$ and returns the updated intention structure $\mathcal{I}$'. Assuming the intention that generated the event is accessible through the event e (int_e, Line 3), the createIntention function (Algorithm 7) generates a new intention ($\langle \sigma'_{int}, st' \rangle$) by using a new substitution σ'_{int} and a new intention stack st', which is composed of the st of int_e jointly with the plan body with σ'_{int} applied (Line 5)

After the plan selection phase, the interpreter executes intentions from the intention structure as detailed in Algorithm 8. The function executeIntention retrieves

Algorithm 4 Function that queries the plan library and returns the candidate plan [47].

1: **function** SELECTOPTIONS(e, $\mathcal{B}$, $\mathcal{PL}$)
2: $Opt \leftarrow \emptyset$
3: **for** plan $\langle t, \mathcal{C}, bd \rangle \in \mathcal{PL}$ **do**
4: **for** σ, σ' such that $unify(e, t, \sigma) \wedge (\mathcal{B} \models \mathcal{C}(\sigma, \sigma'))$ **do**
5: $\sigma_{opt} \leftarrow \sigma \cdot \sigma'$
6: $Opt \leftarrow Opt \cup \{\langle e, \mathcal{C}, bd, \sigma_{opt} \rangle\}$
7: **end for**
8: **end for**
9: **return** $\mathcal{B}$'
10: **end function**

Algorithm 5 Function to select plans and fill the intention structure [47].

1: **function** SELECTPLANS(e, $\mathcal{B}$, $\mathcal{PL}$, $\mathcal{I}$)
2: $\mathcal{I}' \leftarrow \emptyset$
3: $\mathcal{E}', e \leftarrow pop(\mathcal{E})$
4: $Opt \leftarrow selectOptions(e, \mathcal{B}, \mathcal{PL})$
5: **if** $Opt \neq \emptyset$ **then**
6: Pick one option $\langle t, \mathcal{C}, bd, \sigma_{opt} \rangle \in Opt$
7: $\mathcal{I}' \leftarrow$ ADDINTENTION(e, $bd, \sigma_{opt}, \mathcal{I}$)
8: **else if** $(e = +?\varphi) \wedge (\mathcal{B} \models \varphi\sigma_b$ for some $\sigma_b)$ **then**
9: $\mathcal{I}' \leftarrow$ ADDINTENTION($e, \emptyset, \sigma_b, \mathcal{I}$)
10: **else**
11: $\mathcal{E}', \mathcal{I}' \leftarrow$ INTENTIONFAILURE($int_e, \mathcal{I}, \mathcal{E}'$)
12: **end if**
13: **return** $\mathcal{E}$', $\mathcal{I}$'
14: **end function**

the next step contained in the intention stack and verifies whether it is an action or an event. The interpreter executes actions immediately in the environment (Line 6). In contrast, in cases where intentions generate events, the interpreter adds them into the event queue to proceed the next reasoning cycle (Lines 12, 14). When the executed action is the last step from the intention stack (Line 7), that is, the remaining intention stack is empty, then the algorithm removes the intention and the function execution terminates (Line 9). We abstracted the EXECUTEACTION function since it depends on how the environment in which the interpreter is running is implemented.

2.3 Environments

In this section, we discuss the environment specifications that we handle in our work. Sections 2.3.1 and 2.3.2 describe approaches to modelling sequential decision-making that underpin the implementation of the natural language environments used in this

Algorithm 6 Function to add new intentions in the agent intention structure [47].

function ADDINTENTION(e, bd, σ_{opt}, $\mathcal{I}$)
  $\mathcal{I}' \leftarrow \emptyset$
  int_e is the intention that generates e
  $\langle \sigma, st \rangle$ = CREATEINTENTION(e, bd, σ_{opt})
  if e is in either of the forms $+!\varphi, -!\varphi, +?\varphi, or -?\varphi$ **then**
    $\mathcal{I}' \leftarrow (\mathcal{I} - \{int_e\}) \cup \{\langle \sigma, st \rangle\}$
  else
    $\mathcal{I}' \leftarrow \mathcal{I} \cup \{\langle \sigma, st \rangle\}$
  end if
  return $\mathcal{I}$'
end function

Algorithm 7 Function to instantiate intentions given a new intention stack with all substitutions applied [47].

function CREATEINTENTION(e, bd, σ)
  if e is in either of the forms $+!\varphi, -!\varphi, +?\varphi, or -?\varphi$ **then**
    $int_e \leftarrow \langle \sigma_{int}, st \rangle$ // int_e is the intention that generates e
    $\sigma'_{int} \leftarrow \sigma_{int} \cdot \sigma$
    $st' \leftarrow$ PUSH$((bd \cdot \sigma'_{int}), st \cdot \sigma'_{int})$
    return $\langle \sigma'_{int}, st' \rangle$
  else
    return $\langle \sigma, bd\sigma \rangle$
  end if
end function

article. Section 2.3.3 introduces the concepts of a text-based environment and details how states and actions are represented. Section 2.3.4 describes ScienceWorld, a text-based environment instance, which we use in our experiments.

2.3.1 Markov Decision Process

A Markov Decision Process (MDP) is a modelling approach to address sequential decision problems that require the agent to perform a sequence of actions to achieve its goal [36, 50]. In an MDP, the agent interacts with the environment by transitioning between fully observable states and receiving a feedback signal indicating how good the transition from the previous state to the current state is. Formally, we describe an MDP as follows:

Definition 6 (MDP) An MDP comprises a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ represents a set of the environment states; $\mathcal{A}$ a set of actions; $\mathcal{T}$ a transition function; $\mathcal{R}$ the reward function; and γ the discount rate, $0 \leq \gamma \leq 1$.

Algorithm 8 Function to instantiate intentions given a new intention stack with all substitutions applied [47].

```
1: function EXECUTEINTENTION(I, E, B)
2:     E', I' ← ∅
3:     int ← topmost intention from I
4:     int', s ← POP([s_1, ..., s_n])
5:     if s is an action then
6:         EXECUTEACTION(s)
7:         if int = ⟨σ, []⟩ then
8:             I' ← I − {int}
9:             return E', I'
10:        end if
11:    else if s is a subgoal g then
12:        E' ← PUSH(+g · σ, E)
13:    else if s is a belief event then
14:        E' ← PUSH(s · σ, E)
15:    end if
16:    I' ← (I − {int}) ∪ {int'}        //removes old intention and includes new one
17:    return E', I'
18: end function
```

The goal of MDPs is to acquire a decision-making function π, called policy, that maps a state $s \in stateset$ to an action $a \in \mathcal{A}$. That is, an MDP aims to find a policy π that returns an action a that maximises the expected discounted return G_t, which incorporates the discount rate γ from Definition 6. The discount rate is the present value of future rewards, which measures how myopic or far-sighted G_t is. For example, if γ is 0, the G_t value considers only the immediate reward (i.e., $\mathcal{R}$), whereas values closer to 1 consider a more long-term reward. We describe G_t in the equation below:

$$G_t = \sum_{k}^{\infty} \gamma^k \mathcal{R}_{t+k+1} \tag{3}$$

2.3.2 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP) is a formal model for an agent's decision-making in cases where it is not possible to access the complete state information [36]. That is, the agent makes decisions under uncertainty by relying on observations that provide partial information about the current state. In this case, the agent needs to observe its surroundings to gather more information about the environment. To measure the uncertainty of observations, the agent uses a sensor model to formulate a function that returns the probability of perceiving an observation given a specific state. Formally, we define a POMDP model as follows.

Definition 7 (POMDP) A POMDP model tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \Omega, \mathcal{R} \rangle$, where $\mathcal{S}$ represents a set of the environment states, $\mathcal{A}$ a set of actions, $\mathcal{O}$ a set of observations, $\mathcal{T}$ a transition function, Ω the observation probability function, and $\mathcal{R}$ the reward function.

However, since states are not fully observable, a POMDP agent should compute a probability distribution over all actual states to generate the belief state [51]. A valid option for initialising the probability distribution of the belief state is to use a discrete uniform distribution over $|\mathcal{S}|$ [36]. Given an observation $o \in \mathcal{O}$ perceived and actions performed by the agent, we compute the belief state $bs(s')$ of a state s' as follows:

$$bs(s') = \alpha \Omega(o|s') \sum_{s} \mathcal{T}(s'|s, a) bs(s) \tag{4}$$

Equation 4 formulates a recursive Bayesian update to compute $bs(s')$ to predict the s' probability. It multiplies the probability of the new observation o given the state s' (i.e., $\Omega(o|s')$) with the sum of all belief state of states $s \in \mathcal{S}$ considering the transition to s' from s by taking action a (i.e., $\mathcal{T}(s'|s, a)$). The α term is a constant that normalises the probability distribution to make the sum of belief states 1.

In contrast to a fully observable Markov Decision Process (MDP), which uses a policy to map states to actions, in a POMDP we cannot rely on this definition, since the agent does not have certainty about the current state. Instead, we can use the agent's belief state to predict the next action, that is, using a policy as a function to map the current belief state to an action [36]. In this case, the policy of POMDP agents does not depend on the actual state to predict their next action, but rather on their current belief state.

2.3.3 Text-Based Environments

Text-based environments are simulated environments that represent the world's information through natural-language descriptions. Interactive Fiction games are an instance of a text-based environment where the player should interpret textual observations of the simulated world and take text actions to progress through the game story [8, 9]. Such an environment aims to evaluate players' competencies in natural language understanding, common-sense reasoning, and deduction to make decisions that achieve the game objective. Regarding the development of autonomous agents to address real-world problems, text-based environments provide a challenging testbed for designing agents capable of understanding natural language and interacting effectively with humans.

Implementations of text-based environments [8, 9, 31] model it as a POMDP, assuming that the sentences in the perceived observation do not fully describe the game world. That is, the agent cannot assume full observability in such environments. For example, textual observations convey information about what the agent is seeing at its current location, which requires the agent to explore the world to know what the other locations may have. We leverage Definition 7 to specify each component in text-based environment as follows: $\mathcal{O}$ corresponds to sentences describing the performed action effects in the environment, $\mathcal{A}$ to textual actions encoded in imperative sentences

(e.g., go outside), and $\mathcal{S}$ to latent states encoded in a textual description informing location details, object seen, and objects in inventory.

When an agent executes an action, the environment returns an updated state reflecting the effects and changes that have occurred. Specifically, for each agent's performed action, the environment provides two pieces of textual information: a sentence describing the action's effects (i.e., the observation), and a paragraph describing the situation at the current location after the action. Besides the texts that inform the current environment condition, the environment also sends a reward signal based on an unknown reward function $\mathcal{R}$, along with a boolean signal indicating whether the agent can achieve the environment's objective.

2.3.4 ScienceWorld

Science World is a text-based environment that simulates engines for thermodynamics, electrical circuits, matter and chemistry reactions, and biological processes [31]. This environment aims to create a benchmark to evaluate the scientific reasoning abilities of autonomous agents at the level of the standard elementary school science curriculum. The environment provides 30 benchmark tasks across 10 topics covering understanding changes in the state of matter, building electrical circuits, and the pollination process. Similar to the Jericho framework [9], the Science World environment provides a set of valid actions for a given state and an object model to support the development of autonomous agents.

Science World map contains 10 distinct locations around a house (kitchen, bathroom, workshop, art studio, greenhouse, outside, etc.) and 195 specific types of objects. Each location contains a set of random objects contextually relevant to the task objective. The Science World object model includes 23 animals, 11 plants, 25 substances, 10 canonical liquid containers (such as tin cups or glass jars), 13 electrical components (such as light bulbs, motors, wires, and generators), 16 devices (including a stove, thermometer, and stopwatch), and 15 common pieces of furniture (e.g., tables, chairs, etc.) [31].

Such complex settings aim to evaluate the agent's reasoning about location transitions and object interactions, and to test its understanding of combining distinct objects, considering their state of matter. For example, to boil the water, the agent should walk around the map to find a metal pot, search for water, and use a stove. In this test case, the agent should leverage prior knowledge about the object state to avoid choosing a wooden cup instead of the metallic one, since it will come into contact with fire. Furthermore, the agent should use the thermometer object to monitor the water state, as waiting too long can cause it to turn into vapour.

Each task in Science World includes a natural-language description of the main objective and high-level instructions for each sub-goal. The state consists of a set of sentences that describe the objects available in the current room and which rooms the agent can go through according to the Science World map. For each action execution, the environment gives a score as a reward signal to apply reinforcement learning methods following the implementation of POMDP text-based environments, as we detail in Section 2.3.3). Throughout the agent interaction in the environment, the

task score increases as the agent approaches the task objective. Figure 2 illustrates an example of tasks provided by the Science World web application[1].

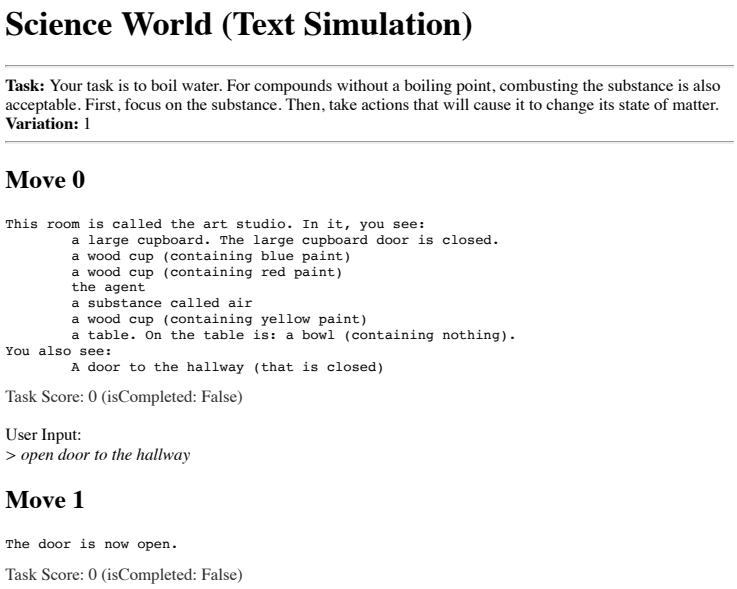


Fig. 2 Example of Science World task with the state encoded in natural language and the action *open door to the hallway* selected.

3 Agent Architecture Overview

In this section, we provide an overview of the NatBDI agent architecture for performing tasks in text-based environments, outlining how we organise the BDI components within the agent's mental state to handle natural language information. In text-based environments, the agent starts by receiving a task description consisting of a small text detailing how to achieve the goal. Such a goal indicates what the agent should do and is written in natural language as an imperative sentence (e.g., "boil water", "take a metal pot", etc.).

At each interaction with the environment, the agent perceives the state as a textual description of its current location. The textual observations detail what the agent perceives from the environment, consisting of sentences that can describe objects perceived, doors in the room, etc. The reasoning process uses such observations as input to update its belief base, which it then uses to make inferences over its course of action. In Figure 3, the agent perceives that it is in the kitchen and perceives a set of objects that are in the room. Since its goal is to take the metal pot, the agent searches its

[1] https://github.com/allenai/ScienceWorld (last accessed on November 30, 2025).

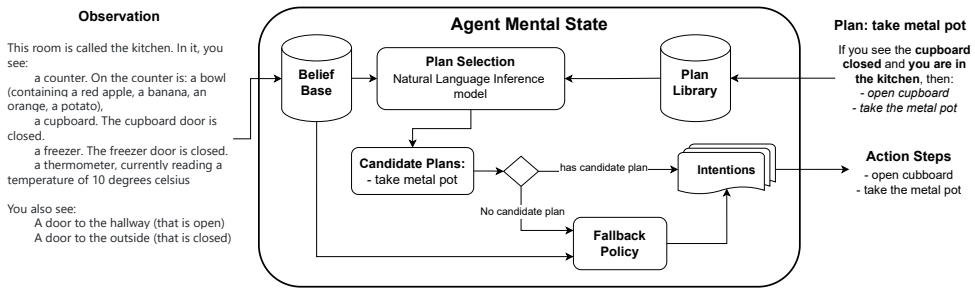


Fig. 3 Diagram illustrating the NatBDI architecture to handle and actuate over natural language environments.

natural-language plan library and finds a plan that instructs it to open the cupboard and take the metal pot, which moves it toward its goal.

Once the agent perceives the goal from the environment, the reasoning process begins to find candidate plans that address it. The plan selection mechanism searches for options that consist of natural-language plans. These plans represent the goal, plan context and instructions in their body as sentences. We introduce a novel component that executes logical inferences over sentences, addressing plan context verification by checking whether it logically follows the belief base (recall Section 4). Given a candidate plan found by the reasoning process, each step in its plan body is included in the intention structure. During the execution of the agent's reasoning process, intention failures may occur when no plan is found (i.e., no plan candidates), given a specific goal addition event. Since text-based environments inform the agent whether the task is finished after each action, the agent can execute the entire candidate plan and still fail to reach the goal. To deal with such cases, the agent executes an intention failure mechanism, which we call the fallback policy (further detailed in Section 3.3).

Given the components presented above, along with the new fallback policy, we extend Definition 1 to formally describe our architecture as follows:

Definition 8 (NatBDI Agent) An NatBDI agent is a tuple $\langle Ag, \mathcal{E}, \mathcal{B}, \mathcal{PL}_{nl}, \mathcal{I}, \pi_{FB} \rangle$, where Ag is the agent identifier; $\mathcal{E}$ is the queue of events; $\mathcal{B}$ is the belief base consisting of natural language observations; $\mathcal{PL}_{nl}$ is the plan library, a set of plan rules written in natural language; $\mathcal{I}$ is a set of intentions; and π_{FB} is the agent's fallback policy.

3.1 Belief Base

Our approach to representing the natural language belief base follows the traditional BDI model, which organises the information perceived into literals in its mental model, except that our agent receives natural language observations rather than symbols. The natural language belief base consists of sentences written in the second person that convey the agent's perceptions in a human-readable format. Each belief is a sentence that indicates what the agent perceives in the current situation, together with the objects currently in its inventory.

Each sentence corresponds to a symbolic belief representation from traditional BDI approaches, expressing information such as what the agent perceives (e.g., "you see a table") or where the agent is located (e.g., "you are in the kitchen"). Instead of relying on ground predicates with terms of predetermined structure, natural language beliefs are composed in a less constrained structure. It leverages the intrinsic nature of human language to represent the same semantic meaning using different expressions (i.e., ambiguity), thereby making language processing more complex [52, 53]. We argue that representing beliefs as natural-language sentences helps humans understand what the agent has perceived about the world, allowing them to scrutinise its mental state further and gain a better understanding of its behaviour.

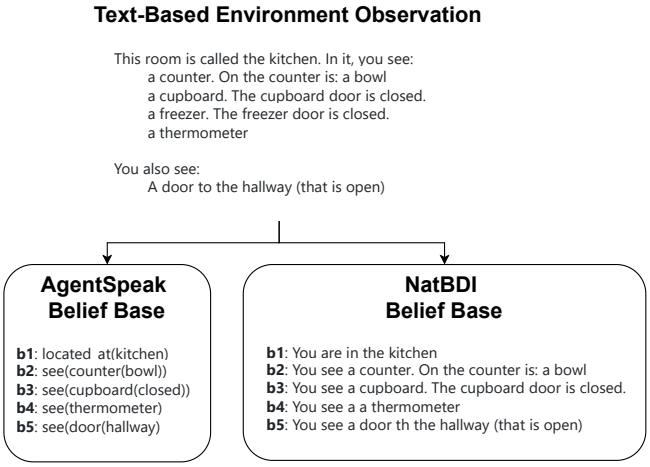


Fig. 4 Image comparing the belief base representation by using symbolic beliefs in AgentSpeak language and natural language beliefs by using our approach. Grounded predicates in this image are exclusively for illustrative purposes.

Figure 4 shows a comparison of the belief base formulation process based on perceived information from the environment between traditional BDI approaches and the NatBDI belief base. To apply traditional BDI agents in a text-based environment, we require a language parser that follows the domain definition to identify each fact from observation and convert it into a grounded predicate [54]. Such symbols should follow a domain ontology designed manually by human experts in the environment. In contrast, NatBDI beliefs consist of sentences contained in the perceived observation, which require only manipulation of the text string by segmenting each sentence into a single belief. Our research focuses on applying the BDI reasoning process directly to natural-language sentences, without requiring any specific formalism or text structure for sentences that represent beliefs.

3.2 Plan Definition

In this section, we explain how NatBDI handles natural language plans in its BDI reasoning cycle [1].

3.2.1 Natural Language Plan Library

In addition to operating in environments described in natural language, our agent architecture provides a natural language interface that enables developers to specify plan rules without requiring expertise in specialised agent programming languages such as AgentSpeak [48]. We express these plans using a controlled natural language [55], which constrains the vocabulary and structure to produce clearly defined conditions and beliefs. Such controlled languages form a restricted subset of natural language that is easier to process automatically, allowing developers to embed natural-language descriptions directly within the structure of a plan rule. Unlike approaches that convert controlled natural language into symbolic forms such as Prolog clauses [55], our architecture performs plan selection by reasoning directly over the natural-language plan rules themselves.

In NatBDI, this controlled natural language is used to define plans that mirror the structure of AgentSpeak plan-rules. Each plan-rule, therefore, comprises three elements: a sentence specifying the intended goal, a description of the plan context, and a sequence of sentences forming the plan body. The goal statement indicates the task the agent aims to achieve and corresponds to the triggering event t (see Section 2.2.2). During execution, NatBDI selects the appropriate natural language plans by aligning goal-addition events from the agent's event queue with the triggering events defined in the plan library. The plan context is expressed as a collection of natural language sentences joined by the keyword "AND", indicating their logical conjunction. Formally, plans expressed in our controlled natural language adhere to the structure illustrated in Listing 1:

Listing 1 Template for a plan written in NatBDI controlled natural language.

```
IF <goal statement>
CONSIDERING <plan context statements>
THEN:
<plan body>
```

To support hierarchical plans, the agent interpreter allows the plan body to contain sentences that denote either actions or subgoals. These subgoals function in a manner analogous to goal-addition events in AgentSpeak [48]. Within the plan body, we introduce the keyword "**PLAN TO**" to clearly mark sentences that correspond to goal additions, distinguishing them from those that specify concrete actions to be executed. As in AgentSpeak, the agent manages subgoals using a stack-based structure: newly adopted goals are pushed onto the stack, and completed goals are popped as execution progresses. This explicit keyword enables the interpreter to differentiate between commands intended for the external environment and those that represent internal reasoning steps for handling nested subgoals.

For instance, the natural-language plans presented in Listing 2 demonstrate how a human designer can directly specify the steps an agent should follow to accomplish

specific tasks. In the first plan, the designer knows that the metal cup is located inside a cupboard that starts closed, and provides explicit instructions for retrieving it. The triggering event for this plan is the goal "get the metal pot", and the plan becomes applicable when its context holds (i.e., when the agent is in the kitchen and observes that the cupboard is closed). The plan body contains two actions, expressed in natural language, that the textual environment will interpret: the agent should open the cupboard and then pick up the metal pot. The second plan is activated when the goal is to melt water, and its structure shows how a subgoal (in this case, obtaining the metal pot) can be embedded within a larger plan.

Listing 2 Plans in our controlled natural language to pick the metal pot and melt water in ScienceWorld.

```
IF your task is to get the metal pot
CONSIDERING: you are in the kitchen
        AND you see the cupboard closed
THEN:
open the cupboard,
take the metal pot

IF your task is to melt water
THEN:
PLAN TO get the metal pot
pick up thermometer,
...
```

3.2.2 Entailment with Natural Language Beliefs

Most BDI implementations choose plans by checking whether the agent's belief base entails their context conditions, and this mechanism determines which options the agent can pursue next [29]. In practical terms, when the beliefs logically support the conditions specified in a plan, the agent adopts that plan and commits to executing its steps within an intention structure. Because our agents operate over natural language, determining entailment in this setting becomes a central challenge. To address this, we use a machine learning based NLI model [13, 39] to approximate the entailment relation over natural language within the NatBDI architecture.

In the BDI framework, plan selection is based on an entailment check that produces a Boolean value indicating whether a plan is applicable. Modern NLI models, however, typically frame inference as a three-way classification task that distinguishes between entailment, contradiction, and neutral relations [37, 39] (see Section 2.1.2). Since our goal is to determine whether the agent's beliefs support the context of a plan, we treat both the contradiction and neutral categories as instances of non-entailment [35]. As a result, the NLI model is used as a binary classifier that returns a Boolean outcome under a closed-world assumption. We further assume that the classifier provides both the entailment decision and a confidence score, which allows the BDI interpreter to prioritise among candidate plans. This behaviour is formally captured in the function defined in Definition 9.

Definition 9 (NLI Function) A nli function is a function $nli : (p, h) \rightarrow \langle ent, nli_{confidence} \rangle$ where p and h represents the premise and the hypothesis sentences respectively, ent is a Boolean value indicating whether p entails h, and $nli_{confidence}$ is a numerical value indicating the model inference confidence.

In practice, determining whether the belief base entails a given plan context requires checking that each sentence in the context is supported by at least one belief. This means the agent must compare every belief in the belief base with every sentence in the plan context to assess whether the plan is applicable. Because both the belief base and the plan context may contain multiple sentences, NatBDI performs this entailment computation by forming the Cartesian product of the two sets. Formally, given the sets of sentences $\mathcal{B}$ and $\mathcal{C}$, we construct the matrix $\mathcal{C} \times \mathcal{B}$ as shown in Equation 5. For each pair consisting of a belief $b_i \in \mathcal{B}$ and a plan context sentence $c_i \in \mathcal{C}$ within this product, we apply the natural language inference function nli (Definition 9) as described in Equation 6, which yields a Boolean matrix $E_{i,j}$. This matrix stores all entailment outcomes for each i-th context sentence and each j-th belief.

$$M_{i,j} = \mathcal{C} \times \mathcal{B} = \{(c_i, b_j) \mid c \in \mathcal{C} \wedge b \in \mathcal{B}\} \tag{5}$$

$$E_{i,j} = \{nli(c_i, b_j) \mid (c_i, b_j) \in M_{i,j}\} \tag{6}$$

To determine whether a plan qualifies as a candidate for selection, we first take the disjunction of the Boolean values in $ent_{i,j}$ for each context sentence i across all beliefs j. Each resulting Boolean value expresses whether that context sentence is supported by at least one belief. We then take the conjunction over all context sentences c, producing a single Boolean outcome. This final value specifies whether the belief base entails the entire plan context, and we formalise this operation in Equation 7.

$$\mathcal{B} \models \mathcal{C} \doteq \bigwedge_{c_i \in \mathcal{C}} c_i \bigvee_{b_j \in \mathcal{B}} b_j \tag{7}$$

For illustration, assume a state in which the belief base contains the following sentences:

- "this room is called the kitchen"
- "you see a cupboard, the cupboard door is closed"

Now consider a plan whose context consists of the sentences:

- "you are in the kitchen"
- "you see a closed cupboard"

The entailment computation proceeds as follows. We first construct the matrix $E_{i,j}$, where each row i corresponds to a context sentence and each column j corresponds to a belief sentence. The resulting matrix is:

$$\begin{bmatrix} T & F \\ F & T \end{bmatrix} \tag{8}$$

Here, the belief "this room is called the kitchen" entails the context sentence "you are in the kitchen", and the belief "you see a cupboard, the cupboard door is closed" entails "you see a closed cupboard". Applying the disjunction across each row yields:

$$\begin{bmatrix} T \\ T \end{bmatrix} \tag{9}$$

A final conjunction over these values gives the scalar Boolean $\begin{bmatrix} T \end{bmatrix}$, which indicates that the belief base entails the full plan context. Therefore, this plan qualifies as a valid candidate for selection.

3.3 Fallback Policy

When the agent retrieves a goal addition event that does not correspond to the triggering condition of any plans in the plan library, the traditional BDI reasoning cycle raises a failure during plan selection, as we describe in Section 2.2. Plans might not achieve the task goal even when each plan step is executed successfully, due to a failure in the plan design [47]. In our agent architecture, we introduce a fallback policy to address plan failures when a goal-addition event does not correspond to any plan rule in the agent's plan library.

We automatically generate the fallback policy by employing reinforcement learning methods in natural language domains, which leverage prior experience to map text states to actions. Recall from Section 2.3.3 that approaches that apply reinforcement learning on text-based environments [9, 31, 56, 57] model the agent's decision-making component as a POMDP. We incorporate existing RL approaches by encapsulating the trained policy as a function that predicts an action given the current belief base, as formally detailed in Definition 10. Our approach leverages the correspondence between POMDP belief state bs and BDI belief base $\mathcal{B}$ to design the function π_{FB}[51]. Conceptually, our architecture is agnostic to specific policy implementations, allowing us to integrate different methods and compare them in our experiments.

Definition 10 (Fallback Policy) The fallback policy is a function $\pi_{FB} : \mathcal{B}_t \rightarrow a_t \in \mathcal{A}$ that maps the belief base $\mathcal{B}_t$ to an atomic action a_t in turn t.

In our approach, each action a predicted by the fallback policy behaves the same as a BDI action, since the reasoning cycle treats them uniformly. The reasoning process inserts the fallback action a into the intention structure $\mathcal{I}$ to execute it, interleaved with plan actions in a multistep reasoning process. Notice that the fallback policy produces only atomic actions. That is, it does not support generating events, such as goal-addition events, to trigger other plans in the agent's plan library. Despite not generating hierarchical plans or plans with multiple sequential steps within a *single plan body*, as traditional BDI approaches do, the agent can produce multistep plans by repeatedly executing the fallback policy, resulting in sequences of actions during task execution.

4 NatBDI Interpreter

In this section, we introduce the NatBDI interpreter and all mechanisms involved in the BDI reasoning cycle. We organise this section as follows: First, we describe the reasoning cycle loop adaptation to handle text-based environment goals in Section 4.1. Second, we detail our approach to events perceived by the interpreter from the environment in Section 4.2. Third, we describe how the interpreter handles the action execution and what objects the environment returns in Section 4.3. Finally, we explain the fallback mechanism in cases where the interpreter raises an intention failure in Section 4.4, and demonstrate an example in Section 4.5.

4.1 Control Cycle

In this section, we describe the NatBDI BDI reasoning cycle that controls the agent's behaviour in text-based environments. In contrast to traditional BDI interpreters, which assume that a successful plan execution achieves the main goal, the NatBDI interpreter distinguishes between the plan goal and the environment goal. Text-based environments [9, 31] provide textual information that instructs the agent to accomplish the task objective. Manually developed plans might contain design errors that can lead to a successful execution in the reasoning cycle while failing to accomplish the environment's goal. Algorithm 9 outlines the NatBDI interpreter reasoning cycle, which receives the BDI agent tuple that follows Definition 8 along with a text-based environment instance.

Algorithm 9 Abstract implementation of the NatBDI interpreter based on the BDI interpreter from Algorithm 1.

1: **procedure** NatBDIInterpreter($\langle Ag, \mathcal{E}, \mathcal{B}, \mathcal{PL}, \mathcal{I}, \pi_{FB} \rangle$, env)
2: $\quad \mathcal{E} \leftarrow$ perceiveObjective($env, \mathcal{E}$)
3: $\quad finished \leftarrow$ `False`
4: $\quad$ **while** is not $finished$ **do**
5: $\quad\quad \mathcal{B} \leftarrow$ updateBeliefs($\mathcal{B}, \mathcal{E}$)
6: $\quad\quad \mathcal{E}, \mathcal{I} \leftarrow$ selectPlans($\mathcal{E}, \mathcal{B}, \mathcal{PL}, \mathcal{I}, \pi_{FB}$)
7: $\quad\quad \mathcal{E}, \mathcal{I}, finished \leftarrow$ executeIntentions($\mathcal{E}, \mathcal{I}, env$)
8: $\quad$ **end while**
9: **end procedure**

To start the reasoning cycle, we encode the environment-provided task description as a goal-addition event, assuming the agent has a plan for handling the task in its plan library. The instruction encoding process depends on the text-based environment implementation, since there are different approaches to writing task descriptions. We show examples of task descriptions converted into BDI goals in Section 5.2. Our interpreter bootstraps the agent reasoning through a function perceiveObjective that obtains the task description in Line 2 of Algorithm 9.

In Lines 5 and 6 of Algorithm 9, the interpreter calls the functions updateBeliefs and selectPlans, mentioned in Section 2.2.2, which follows the same underlying logic of Algorithm 3 and 5, respectively. However, since NatBDI handles both beliefs and plans in natural language, we adapt the implementation of both functions as we detail in Section 5.1 for a specific text-based environment.

Recall that the interpreter assumes the task is not yet accomplished after the agent successfully executes all intentions in the plan. Considering this, we introduce the Boolean variable $finished$, which is provided to the agent after an action is performed, to control the BDI reasoning cycle. As mentioned in Section 2.3.3, the interpreter considers that the text-based environment implementation returns a signal indicating whether the agent has accomplished the main task, after executing its actions. Although we conduct our experiments in a specific environment (Section 5), NatBDI architecture is generalisable to other POMDP-based text-based environments. Thus, function executeIntentions (Algorithm 9, Line 7) controls this variable throughout the execution of all intentions, which we describe further in Section 4.3.

4.2 Events

The NatBDI interpreter follows a BDI model, representing the agent's relevant perceptions as events to control the reasoning cycle. Each time the agent acts, the text-based environment sends the agent the updated state resulting from the action's effects. The text describing the updated scene covers both updated and unmodified observations, helping maintain the agent's perception coherence without requiring any specific processing to compare previous observations with the new ones [58]. Thus, we simplify the maintenance of the NatBDI natural language belief base throughout the reasoning process in text-based environments by merging the semantics of belief addition and deletion events into a belief update event.

The belief update event carries a list of new beliefs that the agents perceive after performing actions in the environment. The updateBeliefs function updates the belief base by retrieving the belief update event and overwriting the entire belief base completely with the new perceived beliefs. Notice that the NatBDI interpreter does not have any sophisticated belief revision mechanism, as it assumes that the environment is responsible for maintaining the coherence of the information the agent perceives while acting in the environment. This feature of text-based environments [8, 9, 31] enables the agent to bypass the frame problem [58], as it represents the environment information, including both modified and unchanged elements resulting from the actions performed.Thus, overwriting the entire agent's belief base is a simple yet effective approach to preventing inconsistencies, particularly when the effects of performed actions might lead to discrepancies in the belief base. We formalise the belief update event by extending Definition 3 as follows:

Definition 11 (Belief Update Event) A belief update event, denoted as $\pm\varphi$, updates the belief base $\mathcal{B}$ by overwriting the existing beliefs with the observations φ.

For example, if the agent moves from the location loc_a to loc_b, it requires the interpreter to remove all beliefs that represent the objects perceived from loc_a, since the agent is not observing them currently, and to include the beliefs perceived from loc_b in the belief base. The environment returns a new textual observation containing the scene perceived exclusively from loc_b from which the interpreter formulates a new set of beliefs to overwrite the current belief base. If the belief base was not overwritten, a potential inconsistency this example might introduce is including the belief "you are in loc_a" along with the existing belief "you are in loc_b", which would result in the agent being in two locations at the same time. Our interpreter prevents this belief incoherence during the execution of UPDATEBELIEFS function (Algorithm 9, Line 5) by deleting "you are in loc_a" and adding "you are in loc_b" in the belief base, because at every execution step the belief base is overwritten.

4.3 Plan Selection

In the BDI model, agents act on the environment by selecting plans from their plan library and executing the plan body. In this section, we leverage the underlying logics of SELECTPLANS and SELECTOPTIONS mentioned in Section 2.2.2, adapting such functions to use the NATBDI architecture components.

The mechanism for selecting options queries the plan library to find candidate plans, relying on the inference of entailment relations between the beliefs in the belief base and the statements that constitute a plan's context. We refine the SELECTOPTIONS function (Algorithm 4) to introduce the entailment mechanism over natural language sentences following Definition 9. The interpreter applies the *nli* function (discussed in Section 3.2.2) by using all beliefs in the belief base as premises and the plan context as a hypothesis. The list of candidate plans this operation aims to achieve includes the plan rule found, along with the inference confidence to guide plan selection. Notice that both beliefs and plan contexts are grounded sentences, which neither support the $unify$ operation (Line 4 in Algorithm 4) nor substitutions (Line 4,5,6 in Algorithm 4) that handle lifted predicates. We describe the adapted version of *selectOptions* in Algorithm 10.

Algorithm 10 Option Selection algorithm adapted from Algorithm 4 to select natural language plans from the NATBDI's plan library.

```
1:  function SELECTOPTIONS(e, B, PL)
2:      Opt ← ∅
3:      for plan ⟨t, C, bd⟩ ∈ PL do
4:          ent, nli_confidence ← nli(B, C)        // NLI function from Definition 9
5:          if ent is true then
6:              Opt ← Opt ∪ ⟨plan, nli_confidence⟩
7:          end if
8:      end for
9:      return Opt
10: end function
```

Given the use of the NLI function (Definition 9) in the option selection function, we adapt the plan selection mechanism to leverage the inference mechanism's results to rank plans. Specifically, the plan selection mechanism selects the plan with the highest confidence ($nli_{confidence}$) as the plan to be performed within the BDI reasoning cycle. In Algorithm 11, we describe the adjustments to support the inference over natural language, where we modify Line 6 of Algorithm 5 to specify the option selection criterion (i.e., Line 5 of Algorithm 11) in cases where there is more than a single candidate. Recall that the interpreter does not work with belief and plan contexts that use lifted representations and, hence, we remove the substitutions (σ) originally included in SELECTPLANS of Algorithm 11. To handle failed intentions regarding plan selection, we replace the INTENTIONFAILURE (Line 10 of Algorithm 5) function with the fallback policy function, which we explain further in Section 3.3.

Algorithm 11 Plan selection algorithm adapted from Algorithm 5 to retrieve candidate options from the NATBDI's plan library.

1: **function** SELECTPLANS(e, $\mathcal{B}$, $\mathcal{PL}$, $\mathcal{I}$, π_{FB})
2: $\quad\mathcal{I}' \leftarrow \emptyset$
3: $\quad\mathcal{E}', e \leftarrow pop(\mathcal{E})$
4: $\quad Opt \leftarrow selectOptions(e, \mathcal{B}, \mathcal{PL})$
5: $\quad$**if** $Opt \neq \emptyset$ **then**
6: $\quad\quad$Pick the plan rule option $\langle t, \mathcal{C}, bd \rangle$ with the highest $nli_{confidence}$.
7: $\quad\quad\mathcal{I}' \leftarrow addIntention(e, bd, \mathcal{I})$
8: $\quad$**else**
9: $\quad\quad\mathcal{I}' \leftarrow fallbackPolicy(\mathcal{B}, \mathcal{I}, \pi_{FB})$
10: $\quad$**end if**
11: $\quad$**return** $\mathcal{E}$', $\mathcal{I}$'
12: **end function**

As mentioned in Section 4.2, the interpreter updates the belief base after the agent acts by using the environment's feedback, which provides a textual description of the agent's observation considering the effects caused by its action. When the step in the intention stack represents an action, it requires adding the belief update event after the action is executed. Considering this, we introduce Algorithm 12 to detail an implementation adapted from Algorithm 8 of intention execution on text-based environments. The NATBDI interpreter uses the feedback provided by text-based environments, which we explain in Section 2.3.3, using the Boolean variable $finished$ to end the loop inside the control cycle (Algorithm 9) and get the new observations after the agent acts. In this adapted version of NATBDI, we do not cover belief-addition/deletion events, since the textual plan language covers only steps that comprise actions and goal-addition events in the plan body.

Algorithm 12 Function to instantiate intentions given a new intention stack with all substitutions applied [47].

function EXECUTEINTENTION($\mathcal{I}$, $\mathcal{E}$, $\mathcal{B}$, env)
  $\mathcal{E}', \mathcal{I}' \leftarrow \emptyset$
  $int \leftarrow$ topmost intention from $\mathcal{I}$
  $int', s \leftarrow pop([s_1, ..., s_n])$
  if s is an action **then**
    $o, \mathcal{R}, finished \leftarrow executeAction(s, env)$
    $\pm b \leftarrow$ belief update event // containing new beliefs from o (Definition 11)
    $\mathcal{E}' \leftarrow push(\pm b, \mathcal{E})$
    if $int = []$ **then**
      $\mathcal{I}' \leftarrow \mathcal{I} - \{int\}$
      return $\mathcal{E}$', $\mathcal{I}$'
    end if
  else if s is a subgoal g **then**
    $\mathcal{E}' \leftarrow push(+g, \mathcal{E})$
  end if
  $\mathcal{I}' \leftarrow (\mathcal{I} - \{int\}) \cup \{int'\}$ //removes old intention and includes new one
  return $\mathcal{E}$', $\mathcal{I}$', $finished$
end function

4.4 Fallback Policy as Intention Failure Handling

In this section, we introduce an abstract implementation of the fallback policy mechanism for handling intention failure within the NATBDI interpreter (Algorithm 9). The agent employs the fallback policy when no plan candidate is available in its plan library.

Initially, as the plan selection mechanism selects candidate options by analysing the belief base, the FALLBACKPOLICY function maps the belief base into a fallback action a_{fb}. The interpreter instantiates a new intention stack st'_{fb} to formulate a new intention and includes it into the agent's intention structure $\mathcal{I}$. Thus, the NATBDI agent executes the action a_{fb} using the same approach as actions from the manually developed plans do, since the intention structure $\mathcal{I}$ consists of both action types.

Algorithm 13 Fallback policy function that includes the fallback action a_{fb} into the intention structure $\mathcal{I}$.

function FALLBACKPOLICY($\mathcal{B}$, $\mathcal{I}$, π_{FB})
  $a_{fb} \leftarrow \pi_{FB}(\mathcal{B})$ //action predicted by the fallback policy
  $st_{fb} = \emptyset$
  $st'_{fb} = push(a_{fb}, st_{fb})$ // new intention stack containing the fallback action
  $\mathcal{I}' \leftarrow \mathcal{I} \cup \{st'_{fb}\}$
  return $\mathcal{I}$'
end function

4.5 NatBDI Execution Example

In this section, we describe a concrete example of NatBDI execution in a text-based environment to show how the interpreter executes the reasoning cycle. Figure 4.5 illustrates an example in which the NatBDI reasoning cycle instantiates plans to address the task and shows how the NLI component performs inferences during plan selection. Recall from Section 3 that the agent receives the task description, which includes the goal to be achieved and the observation perceived by the agent in the initial state. This example illustrates NatBDI addressing the "melt" task in the ScienceWorld environment. We describe an intermediate step in the execution of the reasoning cycle based on an instantiated plan, with the already executed steps represented by dashed lines and the current steps by a filled background.

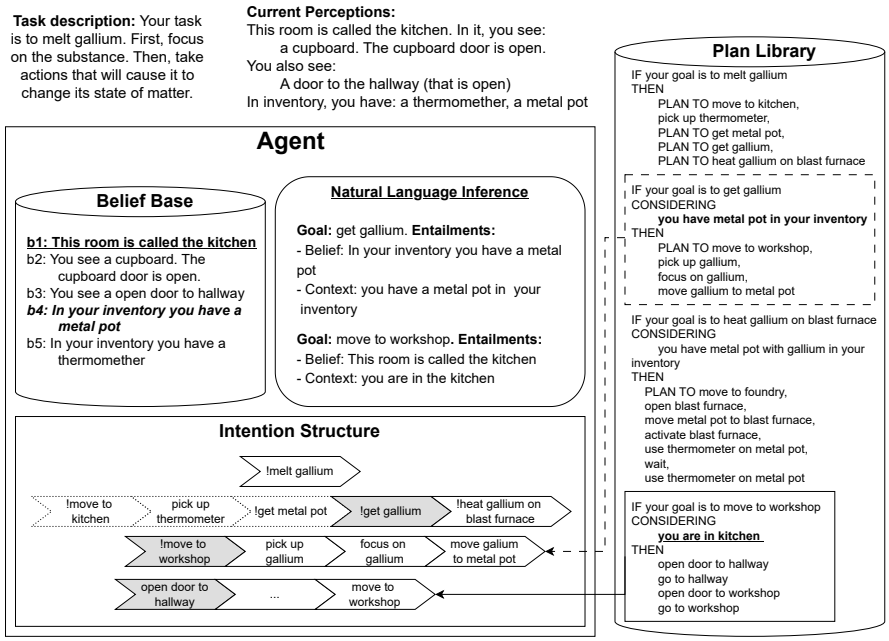


Fig. 5 NatBDI reasoning cycle example with plans addressing the ScienceWorld "melt" task.

In this example, the interpreter parses the text from the task description into a single sentence, as we detail in Section 5.2, to facilitate plan development. The main goal "melt gallium" triggers a plan, shown in dashed lines, that populates the intention structure with five steps, and Figure 4.5 illustrates the execution of the "get gallium" step, which is a goal addition. Notice that the interpreter selects the second plan (with the "get gallium" goal) as the option to execute because the belief "In your inventory, you have a metal pot" entails its plan context (i.e., "you have a metal pot in your inventory"). We illustrate in bold the beliefs from the belief base that entail the plan context.

The first step in the plan body for handling the "get gallium" goal instantiates another plan, extending the intention structure with four additional steps. That is, to obtain the gallium substance, the agent needs to go to the workshop room (fourth plan), which triggers a plan to move from the kitchen to the workshop. In this case, the plan from "move to workshop" includes the action "open door to hallway" into the agent's intention structure, indicating the first instantiated action from the "get gallium" plan. The belief that supports this option is shown in underlined text.

5 Experiments

In this section, we describe our experiments evaluating NatBDI on reasoning tasks in a textual environment. We discuss the implementation details of NatBDI architecture to perform our experiments in Section 5.1. We specify the text-based environment that we use in our experiments in Section 5.2. The process of creating natural language plans for performing our experiments is described in Section 5.3. We specify the existing approaches that we use as a fallback policy component in our experiments in Section 5.4. Finally, we detail the results of the experiments in Section 5.5.

5.1 NatBDI Agent Implementation

We developed the NatBDI components in Python to integrate with the latest neural network libraries. We used a neural network library, *pytorch*[2], and employed a pretrained language model from the HuggingFace repository [59]. Our code is available in the following GitHub repository: https://github.com/yukioichida/nat-bdi (last accessed on November 30, 2025).

In our experiments, we use an existing neural network pretrained on NLI-annotated datasets, which contains optimised weights openly available on the HuggingFace platform. Specifically, we implement the NatBDI agent using a *roberta-large*[3] [13] language model trained on the MultiNLI dataset (MNLI) [39]. For each sentence pair of belief and plan context, our implementation concatenates both sentences as input to the language model, which outputs class probabilities (i.e., entailment and not-entailment) and predicts the logical relationship by selecting the class with the highest probability (i.e., greedy selection). Recall from Definition 9 that NatBDI requires a confidence value to prioritise options when multiple plans are available. Since we employ a neural network whose output layer returns an entailment probability, we calculate the average probability across all sentences classified as entailment and use it as a confidence value for plan selection.

Recall from Section 3.2.2 that NatBDI architecture employs a natural language inference model to infer the entailment relation between the agent's belief base and a plan context formula. Concretely, we develop a matrix-based implementation of the selectOptions function (Algorithm 10) that executes a pretrained neural network to perform logical inferences for each sentence pair (p, h).

To detail further the implementation of the NatBDI plan selection, let us consider an example of a belief base containing the following beliefs:

[2]https://pytorch.org/
[3]https://huggingface.co/FacebookAI/roberta-large-mnli

- "This room is called the kitchen"
- "You see a cupboard. The cupboard is closed"
- "You see a door to the hallway (that is open)"

And a plan context containing the following statements:

- "you see the cupboard closed"
- "you are in the kitchen"

Below, we enumerate the steps that the matrix-based implementation of the *selectOptions* function performs for this example:

1. *Allocate the matrix M to represent all belief, context sentence pairs:* The function *selectOptions* instantiates the matrix M with each row representing all beliefs from the belief base and each column representing statements from the plan context. We formalise the matrix M in Equation 5 and illustrate this example in Figure 6.

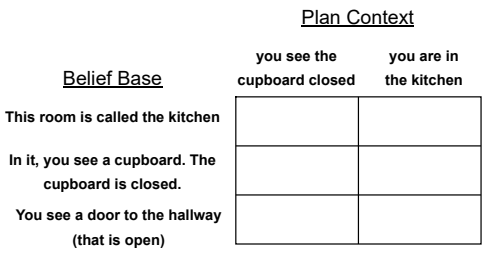


Fig. 6 Matrix M consisting of every belief-plan context pair to be inferred.

2. *Perform nli inference to generate the matrix E.* In this step, given the matrix M instantiated with all belief-plan context pairs, the function calls the NLI model inference in a batch mode to efficiently use the GPU. That is, the function generates the matrix E (recall Equation 6) by filling the matrix M with the results of the NLI neural network batch inference. We illustrate this execution in Figure 7.

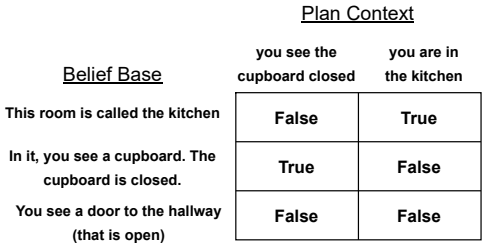


Fig. 7 Matrix E with the boolean results of NLI model inference for each belief-context sentence pair.

3. *Check whether the belief base entails the plan context using the matrix E.* In this step, the function performs Boolean operations on the NLI results from the matrix

E to determine whether the plan, given its context, is a valid candidate. Recall from Equation 7 that the function SELECTOPTIONS performs a disjunction and a conjunction operation over the matrix E. Concretely, the function executes the column-wise disjunction operation to check whether there is a belief sentence that entails the j-th context and, afterwards, a row-wise conjunction to verify whether the belief base entails the plan context. We illustrate the execution of this step in Figure 8. Each condition in the plan context is entailed by at least one belief, and hence this plan is a valid candidate for the agent to select.

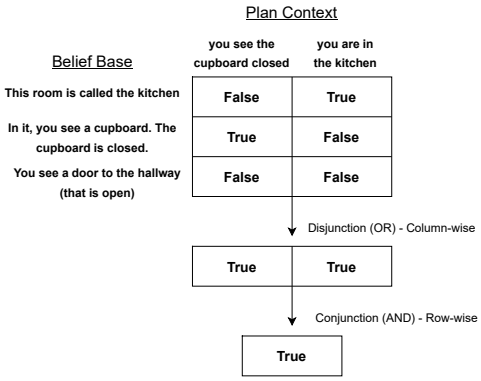


Fig. 8 Boolean operation performed to check if the belief base entails the plan context.

We run our experiments in a machine with 12 core AMD Ryzen 5 with 32GB of RAM, and an NVIDIA RTX 3060 GPU. In this hardware specification, the execution time of the NATBDI implementation is around 3 seconds per test variation. We limit to 50 fallback actions to avoid infinite loops in the NATBDI reasoning cycle.

5.2 ScienceWorld Environment

We perform all our experiments with the NATBDI implementation using the ScienceWorld environment [31]. Our objective is to evaluate NATBDI across tasks of varying difficulty. To this end, we chose to use the tasks *find-non-living-thing* and *melt* for our experiments, as these are the tasks on which existing approaches achieve the highest and lowest scores, respectively [31]. Furthermore, we focus on evaluating the agent across test variations for each task, since these variations include scenarios unseen during fallback policy training (i.e., out-of-distribution states).

Recall from Section 2.3.4 that the ScienceWorld environment encodes information about the state in an observation that mentions all objects within the room where the agent is present. Given that the observation follows a clear format to describe the current scene for the agent, we develop a simple parser to segment it into multiple sentences, each of which describes a single object to represent a natural language belief, as discussed in Section 3.1. We develop a similar approach to extract the goal and represent it as a single sentence, where we identify the sentence part that describes the goal using predefined parser rules.

Table 1 Examples of variations for the ScienceWorld task *find-non-living-thing*.

Variation	Task Description	BDI Goal
225	Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the orange box in the living room.	find a(n) non-living thing by moving it to the orange box in the living room
226	Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the yellow box in the living room.	find a(n) non-living thing by moving it to the yellow box in the living room
227	Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the purple box in the living room.	find a(n) non-living thing by moving it to the purple box in the living room
228	Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the red box in the bedroom.	find a(n) non-living thing by moving it to the red box in the bedroom
229	Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the green box in the bedroom.	find a(n) non-living thing by moving it to the green box in the bedroom

The ScienceWorld task *find-non-living-thing* requires the agent to find an inanimate object and place it in a container, as the task description specifies. This task aims to evaluate reasoning capabilities related to object comprehension to identify a non-living object, and spatial perception to find the target container to place the object. The ScienceWorld environment contains 75 variations of the *find-non-living-thing* task, each with different container types (e.g., red box, blue box, etc.), and the placement of the containers changes for each variation. In this task, the environment generates positive rewards only in three situations: when the agent identifies and picks up a non-living object, moves to the target container location, and places the object in it. In Table 1, we show examples of how ScienceWorld describes the task description along with the parsed BDI goal.

In contrast to *find-non-living-thing* task, machine learning approaches that use ScienceWorld as environment achieve low scores on the *melt* task. In the melt task, the agent's objective is to melt the substance specified in the task description, requiring the agent to understand how to measure the substance's temperature and act accordingly based on its melting point. The ScienceWorld environment provides 9 test variations covering 5 substances with different melting points. This task poses a challenge for machine learning approaches because test variations involve substances with melting points different from those seen in the training phase, requiring handling substances with different melting points. In Table 2, we show examples detailing the task description for the *melt* task along with the goal sentences in BDI.

Table 2 Examples of variations for the ScienceWorld task *melt*.

Variation	Task Description	BDI Goal
21	Your task is to melt lead. First, focus on the substance. Then, take actions that will cause it to change its state of matter.	melt lead
22	Your task is to melt tin. First, focus on the substance. Then, take actions that will cause it to change its state of matter.	melt tin
24	Your task is to melt mercury. First, focus on the substance. Then, take actions that will cause it to change its state of matter.	melt mercury
26	Your task is to melt gallium. First, focus on the substance. Then, take actions that will cause it to change its state of matter.	melt gallium

5.3 Natural Language Plans Setup

We manually create the plans using the controlled natural language from Section 3.2.1 by analysing the task objective and defining a sequence of steps. We leverage annotated trajectories and organise them into subsequences to evaluate how the agent operates with hierarchical instructions, handling subgoals (i.e., goal-addition events). Working with subgoals allows the plan developer to reuse existing plans and introduce plans with smaller plan bodies.

To ease the manual process of creating plans, we automatically generate plans for navigating the environment's rooms by exploring and memorising the transitions. Specifically, we develop a function that performs random movement actions and collects traces to infer spatial information about the rooms in the environment. To represent this spatial information, we build a graph with nodes representing rooms and edges linking them to their neighbouring rooms, as illustrated by an example from ScienceWorld in Figure 9. We use this graph to automatically generate navigation plans by leveraging all available paths from a specific location, as shown in Listing 3.

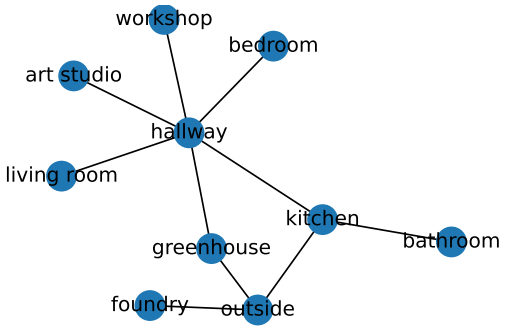


Fig. 9 Graph with rooms obtained from random explorations in the ScienceWorld environment.

Listing 3 Plan to move from art studio to kitchen considering the graph from Figure 9.

```
IF your task is to move to kitchen
CONSIDERING you are in the art studio room
THEN:
go to hallway,
go to kitchen
```

The navigation plans facilitate the development of our plans, which is a manual process. For example, Listing 4 illustrates a manually written plan to melt caesium, which includes a step in the intention structure to generate a goal-addition event for finding a navigation plan that moves the agent to the kitchen, where the metal pot is located. As we mentioned above, the navigation plan generation creates plans for all possible paths, ensuring the plan library contains plans for every room in the graph. This guarantees that the goal addition *move to kitchen* triggers a navigation plan, regardless of the agent's current location.

Listing 4 Plan to melt caesium containing several subgoals.

```
IF your goal is to melt caesium THEN
PLAN TO move to kitchen,
 pick up thermometer,
PLAN TO get metal pot,
PLAN TO get caesium,
PLAN TO heat caesium on blast furnace
```

For the task "find-non-living-thing", since it is more trivial and requires simpler plans, we develop a single plan for each variation. In contrast, for the task "melt", which requires more sophisticated and complex steps, such as manipulating different substance types and measuring the melting point, we introduce plans to handle seven of the nine variations. In ScienceWorld, it is technically possible to manually generate all possible plans by anticipating every scenario in the environment, making it trivial to achieve top scores. However, we assume a potentially incomplete plan library is a more realistic setting, as we would not expect the developer to be able to create all possible plan rules. We evaluate the effects of using the fallback policy when no candidate plan is available in the agent's plan library.

5.4 Fallback Policy Setup

We develop different implementations of the fallback policy component based on two existing machine learning approaches to evaluate their application in our experiments when interleaved with natural language plans. We use the DRRN [56] and Swift [57] approaches, adapting them to integrate into the agent's reasoning cycle seamlessly.

Recall from Section 3.3 that a fallback policy predicts an action that the interpreter includes in its intention structure. The DRRN follows the same approach in which the trained policy predicts a single action, assuming that the environment provides a set of valid actions to the agent. To generate input text in a format consistent with DRRN's expected input, we store the original ScienceWorld observations (i.e., before parsing into beliefs) in the belief base for use as DRRN network input. We follow

Wang et al. [31] to train the DRRN to replicate the same approach for training in the ScienceWorld environment[4].

We use the Swift module from the SwiftSage work [57] as the fallback policy because it performs better in the ScienceWorld environment than other machine learning approaches. Specifically, we employ the pretrained weights from HuggingFace[5] [59] to perform experiments without the need to reproduce the entire Swift training process. In this work, we use the Swift module in isolation from the Sage module since the latter requires ChatGPT-4 [60, 61], which is not publicly available. Similar to how we employ a DRRN as a fallback policy, we adapt the belief base to formulate the input prompt by including past information in NatBDI's belief base, enabling the Swift model to instantiate a valid input prompt.

In contrast to the DRRN training process [31], Lin et al. [57] train the Swift module by using the same language model instance for all ScienceWorld tasks. Thus, we use the same Swift pretrained model for all experiments. While DRRN uses a smaller network with 1.5 million parameters, the Swift module is based on a pretrained T5 language model with 770 million parameters, which is significantly larger than DRRN's.

5.5 Experiment Results

In this section, we present the results of our experiments. Section 5.5.1 shows the results of NatBDI in the ScienceWorld environment. Section 5.5.2 reports the scores from the fallback policy by analysing the effects of including training episodes. Section 5.5.3 reports the results which we obtain by using different NLI pretrained models.

5.5.1 NatBDI Results in ScienceWorld Environment

We now describe the scores our natural language BDI agent achieves in the ScienceWorld environment. First, we measure the BDI agent performance when varying the number of plan-rules in the agent's plan library for the ScienceWorld tasks "find-non-living-thing" and "melt". We show the effects on the overall score and its progress by adding natural-language plans designed by humans. Second, we evaluate the trade-off between plan-rules and the two fallback policies, and the impact each can have, depending on the size and quality of the plan-rules library. Finally, we evaluate both fallback policies separately, that is, an agent with no plan-rules that relies solely on the fallback policy.

In this experiment, we compare different setups by assessing the task score and the number of actions performed. In each execution, the ScienceWorld environment returns a score indicating progress toward the task objective, which we discretise by action type (i.e., natural language BDI plans and fallback policy) to evaluate the effectiveness of the fallback policy in the agent's reasoning cycle. For instance, in the "boiling water" task, if the plan leads to obtaining water, the environment rewards the agent with a partial score even if the main objective is not fully achieved. We

[4]Available in https://github.com/cognitiveailab/drrn-scienceworld (last accessed on November 30, 2025).
[5]Pretrained model available in: https://huggingface.co/yuchenlin/swift_sw (last accessed on November 30, 2025).

report the number of actions performed by the agent using the same categorisation to analyse the frequency of fallback policy triggering across setups.

Table 3 ScienceWorld results from NatBDI execution using two different fallback policy types: Swift and DRRN. We report results by detailing the total scores, the scores from plans (BDI) and fallback actions (FB), and the average number of actions executed from plans (BDI) and from the fallback policy (FB). We highlight the approach that contributes most to the total score in bold.

		Swift			DRRN		
Task	**Num Plans**	**Score (Total)**	**Score (BDI/FB)**	**Actions (BDI/FB)**	**Score (Total)**	**Score (BDI/FB)**	**Actions (BDI/FB)**
find-non-living-thing	0	0.91	0.00/**0.91**	0.00/10.43	0.66	0.00/**0.66**	0.00/50.00
	8	0.92	0.32/**0.60**	3.33/6.91	0.77	0.32/**0.45**	3.33/34.68
	15	0.92	**0.60**/0.32	6.25/5.28	0.86	**0.60**/0.26	6.25/20.40
	23	0.97	**0.81**/0.16	7.64/2.68	0.93	**0.81**/0.12	7.64/9.52
	30	1.00	**1.00**/0.00	9.19/0.00	1.00	**1.00**/0.00	9.19/0.00
melt	0	0.21	0.00/**0.21**	0.00/50.00	0.03	0.00/**0.03**	0.00/50.00
	4	0.32	0.11/**0.21**	5.00/45.33	0.14	**0.11**/0.03	4.78/45.33
	7	0.54	**0.34**/0.20	10.11/34.00	0.36	**0.34**/0.02	10.33/34.00
	10	0.60	**0.56**/0.04	15.89/22.67	0.57	**0.56**/0.01	15.89/22.67
	13	0.79	**0.78**/0.01	20.33/17.00	0.78	**0.78**/0.00	20.56/17.00

Table 3 shows the scores that the agent obtains in "find-non-living-thing" and "melt" ScienceWorld tasks using different fallback policy implementations (i.e., Swift and DRRN). We show different executions varying the number of plans included in the NatBDI plan library (Num Plans). The total score represents the agent's accumulated reward, averaged across multiple task variations in the same environment, and is broken down into the score obtained from using plan-rules (BDI) and the fallback policy (FB). To further detail the proportion of actions in the NatBDI intention structure, we report the average number of actions (Actions) originated from plans (BDI) and from the fallback policy (FB). The number of episodes seen in DRRN training for "find-non-living-thing" and "melt" are 302 and 457, respectively.

In "find-non-living-thing" task, the Swift fallback policy achieves a high score even in scenarios with no plans (i.e., setup with zero plan-rules), surpassing the DRRN approach by a large margin. Notice that Swift employs a significantly larger neural network (i.e., 770 million parameters) than the DRRN policy network (i.e., 1.5 million parameters). Recall from Section 5.2 that the "find-non-living-thing" task is the most straightforward task from the ScienceWorld environment, and the resulting scores from both approaches are noticeably higher compared to the "melt" task. In this task, as the setup that uses the Swift fallback policy already achieves better results without plans than DRRN, it requires a larger number of plans to improve the score. In contrast, the agent configuration that uses DRRN as a fallback policy heavily relies on natural-language plans to improve its score on the "find-non-living-thing" task, highlighting the performance difference between the two approaches.

Regarding the "melt" task, although Swift achieves better scores than DRRN, both fallback policy approaches perform poorly on a task that requires more sophisticated

reasoning. By including a few extra plans (e.g., 4 and 7 plans for the "melt" task), only the Swift fallback policy contributes to the overall score. Such examples corroborate that the fallback policy can partially handle situations where the agent lacks a plan to achieve the goal, given the generalisation capabilities of the fallback policy grounding model. Specifically, when the developed plan lacks sufficient steps to achieve the task objective, the Swift fallback can generate a new plan on the fly by predicting a sequence of actions to progress towards the environment's goal. However, in the same scenario, the DRRN policy could not effectively proceed with the agent's execution towards the goal after failing to find any plans, as it could not contribute to the overall score. Recall from Section 5.3 that we develop plans to cover 7 out of 9 "melt" test variations, which explains the 0.78 score that NatBDI obtains in the last row of Table 3.

Our experiment shows that DRRN policy alone cannot handle important aspects of the environment, such as state-of-matter and target objects, or reason about the task in both task types, corroborating the results presented in [31]. In particular, in the "melt" task, the DRRN-only agent (i.e., the agent with no user-supplied plan rules) obtains a score greater than zero due to random move actions that send it to the exact location of the target element. Although the trained policy could achieve better scores in the "find-non-living-thing", its performance is due to the environment, which rewards the agent with a 0.5 score for solely picking a non-living object, which is more prevalent than living objects. The DRRN fallback policy often correctly identifies a non-living object, as mentioned earlier, but struggles to move it into the task description's container, which lowers the overall score. Even a small number of plan-rules leads the BDI agent to outperform the DRRN-only by a large margin on the most challenging task (melt).

DRRN uses only the text provided by the environment in the current observation (i.e., inventory and scene details). In contrast, Swift uses a prompt that contains substantially more context, such as previous actions and observations, rooms visited, etc. Besides the fact that Swift uses a larger model, the substantial performance difference between the two approaches shows the impact of using a larger context to describe the input. That is, the Swift approach can interpret larger input texts than DRRN, allowing the fallback policy to incorporate more information from the belief base to predict the action. Training policies with additional information by representing textual perceptions through larger input texts enables the agent to retain more details about the environment [4]. Using a large model such as T5 [12] enables Swift to capture more sophisticated patterns in large input texts during training, thereby providing a significant advantage over DRRN.

Introducing natural language plan-rules also leads the agent to require fewer actions (i.e., using shorter plans) to achieve a goal, because our natural language plan-rules encode nearly optimal trajectories in their plan bodies to solve the task. Notice that configurations that use fewer plan rules rely more on the fallback policy and, consequently, lead the agent to generate more fallback actions. Except for the configuration that uses the Swift module (e.g., Line 1 of Table 3, in task "find-non-living-thing"), all fallback policy executions that do not use human-provided plan-rules fail to achieve the task description objective, as the number of fallback actions reaches the limit (50) along with the score lower than 1.

The BDI agent achieves better scores by using manually designed natural-language plan-rules. Still, it does not achieve a perfect score because we use a limited set of plan rules that do not cover all possible task variations. This is a more realistic setting, as we would not expect the developer to be able to create all possible plan-rules. For example, in the task "melt", there are variations in which some objects are not working (i.e., broken stove), which requires the agent to plan more dynamically to find an alternative plan at runtime.

5.5.2 Effects of Including Plans and Training Episodes

In this section, we examine the effects of increasing the number of episodes during DRRN training compared to incorporating more natural language plans. Recall from Table 3 that the DRRN-based fallback policy fails to improve the NatBDI reasoning capabilities in the ScienceWorld tasks, as its resulting score does not improve the agent's overall score. Thus, we further analyse the training process of the DRRN network by generating a set of checkpoints, distinguishing them by the number of trained episodes, to evaluate training efficiency and compare it with NatBDI as a fallback policy component.

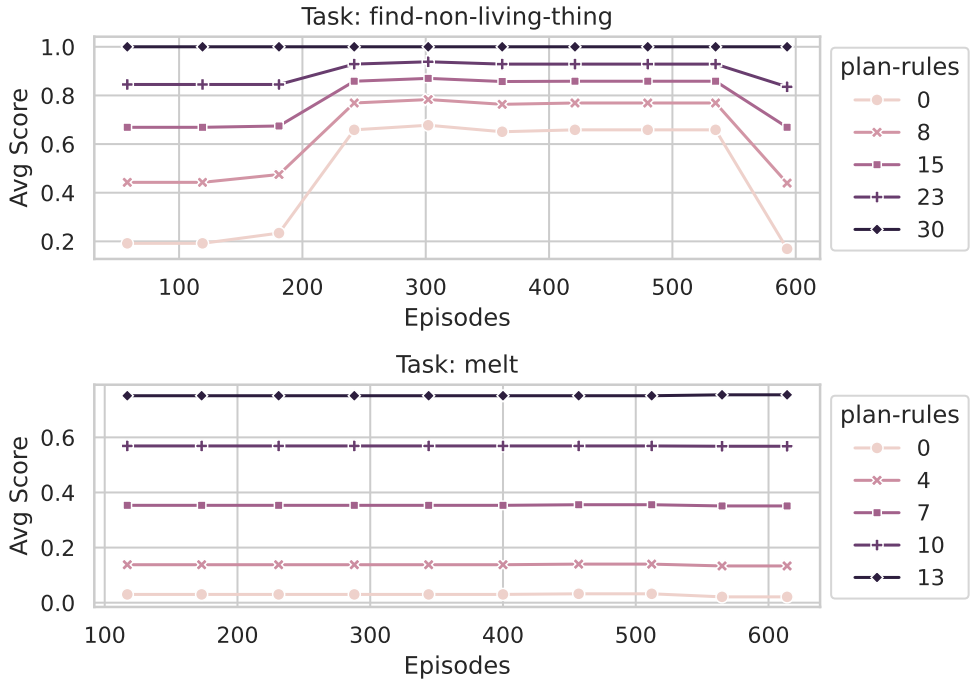


Fig. 10 Scores per episode when scaling the number of plan-rules and the number of training episodes.

While the policy efficiency improves throughout the training phase for the "find-non-living-thing" task, it is prone to overfitting. Recall from Section 5.2 that the task objectives seen in the ScienceWorld test variations involve different substances compared to those in the training phase. Training on more episodes from training variations causes the policy to overfit to the training objectives. For example, adding too many episodes that handle the objective "melt water" misleads the policy when handling "melt gallium", as it overfits to trajectories focused on the water melting point rather than that of gallium. Increasing the number of episodes seen during training improves the DRRN score until it reaches a saturation point at 302 episodes, representing the best configuration we use in Table 3. However, increasing the number of plans yields substantially greater improvements for the agent than prolonging the training process with additional episodes. In fact, extending the number of episodes causes the DRRN parameters to overfit during training and consequently degrades performance in the test variations of the task "find-non-living-thing" after surpassing 535 episodes, as Figure 10 indicates.

In the "melt" task, the number of training episodes does not appear to affect DRRN performance. This makes sense since we have seen in previous results that DRRN performs very poorly in this task, and therefore increasing the number of training episodes here has no discernible effect. This is evident in the task "melt" curve in Figure 10 as it does not change as the number of episodes increases. Thus, this corroborates the claim that DRRN has significant limitations in reasoning over sophisticated tasks (i.e., "melt") that require planning by interpreting natural-language states.

5.5.3 Natural Language Inference Analysis

In this experiment, we focus our analysis on the number of missed plans caused by incorrect inferences in the NatBDI reasoning cycle. For each NLI model, we report scores, the average number of actions per plan, and metrics to illustrate the number of missed plans in Table 4. Additionally, we include the accuracy on the MNLI-m [39] dataset to compare each model's "melt" task score against an established external benchmark. NatBDI executes plans hierarchically, following the traditional reasoning cycle of BDI agents. Consequently, errors encountered during plan decomposition will abort the plan-selection process, making later subgoals unreachable. Incorrect inferences between beliefs and plan contexts lead to incorrect plan selection, causing the agent to select irrelevant or unachievable plans or to stop execution early when no candidate plan is found. Thus, we report the score in the ScienceWorld "melt" task, showing the number of missed plans due to incorrect logical inferences by NLI models, in Table 4.

Recall from our previous experiments (Sections 5.5.1 and 5.5.2) that we use the *roberta-large* [13] NLI model as it successfully executes all of our manually developed plans, resulting in zero missed plans with this NLI model. In fact, the *roberta-large* NLI module achieves higher scores than smaller language models such as *bert-base* and *MiniLM*, both on the MultiNLI matched test set and on the ScienceWorld "melt" task. Part of this experiment is to measure the extent to which the natural language inference model influences planning success. We would expect that large models improve the plan selection and, consequently, the plan's success. Table 4 shows that using the

Table 4 Experimental results of logical inferences from using different NLI models in NatBDI across all "melt" task variation. We show the best model, the smaller model and the best results in bold. The following columns describe the model name, number of parameters (in millions), MNLI accuracy on the development set, the score, average number of actions performed, total plan executions, total missed plan executions, and their percentage.

Model	Params	MNLI-m	Score (BDI)	Num Actions (avg)	Plans (Total)	Missed Plans (Total)	Missed Plans (%)
MiniLM L6	**22**	82.20	0.39	16.56	41	17	0.41
Bert Base	110	84.60	0.56	16.78	46	12	0.26
Roberta Large	355	**90.80**	**0.78**	**22.11**	**58**	**0**	**0.00**

roberta-large model for natural language inference leads to the selection of more plans during the agent's execution, resulting in a superior score compared to the other models. The number of missed plans when using smaller models, such as *bert-base* [11] and *MiniLM* [14], explains the lower scores as the agent is unable to execute all steps of the expected plans due to incorrect inferences. The lower number of actions from plans from such models highlights the incomplete execution of the developed plans across all task variations. Although including a smaller model such as *MiniLM* makes NatBDI more computationally efficient, resulting in 41% of plans being missed and a significant decrease in score due to not executing plans as expected. This indicates that larger models lead to more successful plan selection when used as the NLI model in the NatBDI architecture [62]. Thus, we show that incorporating NLI models using a more robust approach improves the intention progression of the NatBDI by selecting plans that align with human instructions.

Notice that the number of total plan executions in Table 4 counts the unique executions across all test variations of the "melt" task, which may include repeated plans. That is, it counts the number of triggered plans that NatBDI performs during execution, considering all task variations. In contrast, the number of plans from Table 3 represents the number of distinct plans included in the plan library.

We report lexical metrics to measure the complexity of detecting the logical relation between sentences that the NLI models process. To analyse the difficulty of inferring entailment between the belief base and the plan context, we compute the lexical overlap between sentence pairs along with the average number of words in both belief and plan context sentences. We compute the lexical overlap between sentences used as premises and hypotheses to count the number of identical words between the pair. Given a sentence pair consisting of a belief and a context, we compute the number of words contained in beliefs that are absent in the plan context. In cases where lexical overlap is high between the premise and the hypothesis, the inference mechanism tends to more easily infer the entailment relation since both sentences are similar and may express the same idea [52, 63]. For example, it is trivial for an NLI model to infer entailment between the belief-context pair "you see a pot" and "you see a container" due to the significant lexical overlap (i.e., three words). Hence, in such cases, sophisticated language models exploit shallow syntactic heuristics to infer logical entailment between sentences [40, 63].

Table 5 Results of our lexical analysis on all entailment sentence pairs. We report the average lexical overlap, the number of pairs, and the average word count in beliefs and plan contexts.

Model	Lexical Overlap (avg)	Num Pairs (Entailment)	Belief Num Words (avg)	Plan Context Num Words (avg)
MiniLM L6	1.35	26	8.77	5.12
Bert Base	1.25	28	9.18	5.25
Roberta Large	1.28	32	8.44	5.50

In Table 5, for each evaluated NLI model, we provide the following information. First, we report the average lexical overlap contained in sentence pairs to describe the average number of words shared between belief and plan context. Second, we present the number of sentence pairs classified as entailment to summarise the number of pairs evaluated in this analysis. Finally, we report the average number of words in the belief and plan context sentences, respectively.

The amount of lexical overlap is low when compared to the average word number in both sentences, considering our manually designed plan-rules. As the NLI model progresses through the NatBDI reasoning cycle execution by selecting more plans, the number of sentence pairs inferred as entailment also increases. In our experiment, the lexical overlap is significantly lower than the average number of words per belief-plan context pair. For example, in *roberta-large* inferences, the sentence pair inferences have an average of 1.28 common words, while the belief and plan context sentences have averages of 8.44 and 5.50 words, respectively. Thus, we demonstrate that NatBDI NLI implementation is robust to the linguistic variability present in natural language sentences, as it could detect entailment relations in sentences expressed in different ways.

6 Related Work

In this section, we briefly explain related work to our research and compare them with our approach, organising them according to their contribution as follows. First, we survey works that employ BDI agents in natural language environments and compare them with our approach. Second, we discuss works that integrate machine learning techniques in the BDI architecture. Finally, we describe works that introduce approaches for performing logical inference over natural language information.

6.1 Traditional BDI Agents in Natural Language Environments

Most of the related work that applies BDI in textual domains cover conversational agents and personal assistants, requiring understanding and communicating with humans. Although our work does not focus on such problems, we compare methodologies for modelling BDI agents that perceive and act in natural language.

The approach by Nestorovic et al. [64] introduces a BDI model for task-oriented dialogues, focusing on information representation and the deliberation process. This work formulates the agent's beliefs by extracting the user's intention (i.e., dialogue

act) and data (i.e., slots) and separating them into different layers to represent the dialogue context. Given this belief formulation schema, they apply a set of rules and heuristics over the representation of the dialogue context to infer the agent's response. Specifically, to maintain the dialogue with a user, the agent requests more information to keep the user talking, using predefined dialogue heuristics grounded in psychological strategies. Although this work handles task-oriented dialogues similar to ours, our work diverges in the dialogue progress: we train a dialogue policy and leverage an AgentSpeak-like formalism to introduce developed plans, rather than relying on heuristics to engage in dialogue.

Wong et al. [65] introduce a goal-oriented architecture based on BDI to develop a conversational virtual companion that aims to entertain children and perform collaborative tasks. This architecture consists of a component to manage the content of agent responses, named Activity Management, and a component that interprets perceived utterances and maps them to strategies defined as BDI plans, named Conversation Management. Similar to the work above, this approach extracts keywords and aspects, such as sentiment and entities, from input utterances and determines whether the extracted information matches a human-defined pattern.

Nguyen and Wobcke [66, 67] develop an approach that extends the BDI agent architecture for a Smart Personal Assistant (SPA). Their agent implementation focuses on email management tasks, which require the SPA to interpret the user's intent and generate a response. Such an SPA consists of a dialogue manager that uses the BDI reasoning cycle to formulate responses using developed plans. The agent architecture includes a rule-based parser that converts input utterances into a logical representation similar to traditional BDI approaches. Both approaches handle natural language information by employing semantic parsers grounded in predefined rules or linguistic syntax properties to convert text into a formal representation.

Instead of applying reasoning over an intermediate formal representation, our work focuses on developing the agent's reasoning cycle that processes natural language information directly by integrating existing machine learning methods. That is, we model a BDI agent by encoding its mental components entirely in natural language to handle tasks in a text-based environment using natural language plans.

6.2 Machine Learning Techniques and BDI Agents

Our work aims to integrate machine learning approaches to handle situations where no applicable plan exists, given the current representation of the agent's belief base (Section 3.3). Traditional BDI agents do not incorporate adaptive mechanisms to improve their decision-making during execution beyond predefined plan rules in their plan libraries. However, following predefined plan rules ensures that the agent behaves according to human instructions.

Singh et al. [68, 69] develop an approach to determine plan conditions (i.e., plan context) by employing machine learning techniques in environments with changing dynamics to improve hierarchical plan selection in the JACK agent programming language [25]. That is, this work introduces a mechanism that adapts the plan-rule context to environmental changes, enabling the same instructions encoded in existing plan-rules to be used. Concretely, this work uses a decision tree that leverages previous

experience with plan execution in different situations, as it can represent context formulas in the same way that BDI languages, such as JACK, encode the plan context formula. In our research, we apply machine learning to handle text-based environment tasks when no plan is available, rather than adapting existing plans from the plan library. Although this work has a different objective in applying machine learning to BDI agents compared to ours, we envision integrating a similar approach to infer whether the agent can select an existing plan in future work.

Later, Singh and Hindriks [27] introduced the GOAL agent programming language, which aims to improve action selection by applying machine learning techniques. This work employs the Q-learning algorithm to obtain a policy that selects the candidate action with the highest Q-value when multiple options are available. Similar to the existing models we encapsulate in the fallback function (recall from Section 5.4), Their approach encodes state information using the belief base and goals to compute action Q-values, which the agents develop using the GOAL language. This work shares the same objective as our approach to developing the fallback policy using machine learning, but there are conceptual differences as follows: First, we include only the main goal which the environment provides as input to the fallback policy. Second, our work encodes both plans and the agent's mental model into a natural-language representation, which poses greater challenges for performing logical inference than symbolic representations.

Bosello and Ricci [70] integrate a traditional BDI agent framework with a reinforcement learning approach to improve the agent development process using the Jason programming language [26]. Concretely, this approach uses reinforcement learning to learn a policy function that selects actions at runtime and, at the same time, reduces the need to create new plans manually. By introducing learning into the agent's development process, Bosello and Ricci introduce two different types of plans: hard plans, which consist of manually developed plans as in the traditional way; and soft plans, which consist of actions generated by the learned policy function. This work applies machine learning techniques in the same interpreter stage as our work (i.e., plan selection), where hard plans correspond to NatBDI developed natural language plans (recall Section 3.2.1), and soft plans correspond to the NatBDI fallback policy actions. As we compare with other BDI-related works above, the main contrast with our work is that NatBDI employs reinforcement learning over textual information, which processes both the agent's information (e.g., beliefs, goals) and natural-language plans.

Gatti et al. [71] introduce a framework to integrate LLMs' natural language understanding capabilities into BDI agents, in order to create a communication channel between the developer and the agent. This work leverages LLMs by mapping human developer messages into a language understandable to BDI agents and vice-versa, focusing on maintaining the intention and logic-driven dimension of the conversation. ChatBDI focuses on the usability for plan developers as it supports known BDI languages such as KQML as an intermediate language between agents and LLMs, whereas NatBDI extends the plan generation process for non programming experts by encoding agent behaviours in natural language.

Similarly, Ciatto et al.'s work [72] exploits the LLM text-generation abilities to synthesize plans for AgentSpeak(L) agents, named Plan Generation Procedure (PGP). The PGP-generated plans are encoded in a structured language (i.e., YAML) that carries AgentSpeak language elements such as goals, events, plan context and its body. Instead of introducing LLM abilities into the BDI reasoning cycle, the PGP embeds the generative capabilities to create plans on-demand directly while preserving the BDI plan structure. As future work, we plan to explore similar ideas to generate plans without prior domain information by exploiting LLMs, and represent plans in AgentSpeak language to support existing BDI agents, following such approaches.

6.3 Logical Inference on Natural Language Information

Finally, we briefly survey methods for performing logical inference over natural language sentences. First, we detail traditional approaches that handle more formal approaches to develop natural logic systems for inference in language. Second, we discuss more recent approaches that employ modern neural network models to make logical inferences by learning linguistic patterns in logic-annotated datasets.

Williams *et al.* [39] presents an NLI dataset to train machine learning approaches to infer the logical relation between sentences, called MNLI. The MNLI corpus is a large dataset that contains 433k sentence pairs annotated with NLI classes (*contradiction, entailment, neutral*). The MNLI dataset contains diverse sentences, including text and speech, covering a wide range of topics. In this work, we conduct experiments using different NLI models [11, 13, 14] trained using the MNLI train dataset. However, we merge the classes *neutral* and *contradiction* into a single class *non-entailment* since our focus is on identifying whether there is an entailment between sentences.

Richardson *et al.* [63] introduce an NLI dataset that aims to evaluate the different types of reasoning capabilities of machine learning approaches. Specifically, Richardson *et al.* organise the dataset into different semantic phenomena, each of which represents the following logic fragments: boolean coordination, quantification, conditionals, comparatives, and monotonicity reasoning. They train a BERT [11] language model on their dataset and demonstrate that BERT can learn logic fragments as long as the training dataset contains sentence pairs with the corresponding fragment type. That is, given a new semantic fragment, the model must be retrained on data containing sentence pairs that reflect it. Although the authors did not provide the trained models, in future work, we intend to integrate language models trained with such logic fragments to evaluate the NatBDI plan selection mechanism.

Clark *et al.* [38] investigate the capabilities of transformer-based neural networks to reason over facts and rules using synthetically generated data. Given a set of sentences representing facts and a hypothesis sentence, the model should determine whether the facts follow the hypothesis by performing deductive inference over English statements. They show that transformers can reason with facts expressed in natural language as long as the sentences follow a predefined format (e.g., "A is B", "if A and B then C"). In our work, we prefer to use models that follow the natural language inference task (recall Section 2.1.2) to handle sentence pairs to perform logic operations more closely aligned with traditional BDI methods.

7 Open Issues and Limitations

As NatBDI processes natural language texts without relying on a pre-determined format, the reasoning cycle does not manipulate lifted variables as traditional BDI agents do, assuming that every term is a grounded variable. That is, the natural language inference model does not handle substitutions or identify unification relations during logical inference in the plan context or when executing intentions. Identifying when an expression represents a lifted or grounded is a challenging task that arises when manipulating natural language expressions. A possible path to develop a method to handle lifted variables is to explore relations from the monotonicity calculus [34] by identifying forward entailment or other semantic relations between sentence words.

Simulated text-based environments [8, 9, 31] require the agent to act to cause changes in the environment, considering only synchronous events. Consequently, NatBDI can only select and execute plans by reacting to events resulting from its own actions. However, in a simulated environment that more closely reflects real-world conditions, the agent should be able to perceive events arising from external situations and reason about new perceptions not necessarily generated by its own actions. An alternative to overcome this limitation is to improve the NatBDI to handle belief addition/deletion events [43], as existing BDI agents do, by triggering plans or actions based on new perceptions.

To find candidate options for the agent to perform, the NatBDI inference method requires scanning every belief-plan context sentence pair to determine the entailment relation. Bosello and Ricci [70] developed a method to extract observations from the belief base according to the goal, aiming to reduce the state space. Matching the natural language semantics is more complex than handling formal languages since a belief can express the same meaning using different expressions. As an alternative, we plan to explore methods for measuring semantic similarity [52, 53] between plan context statements and beliefs to filter out irrelevant pairs and, hence, reduce the state space, similar to the observation function approach in Bosello and Ricci's work.

8 Conclusion

In this paper, we introduced a novel autonomous agent architecture based on BDI concepts to solve tasks in environments that require human language comprehension to make decisions over a natural language action space. We developed the NatBDI architecture, which represents the BDI mental components entirely in natural language, allowing humans to understand the information behind the agent's behaviour. In contrast to traditional BDI approaches, which encode their internal state in symbolic representations to leverage formal logic systems, NatBDI belief base comprises beliefs encoded as descriptive sentences that capture what the agent is seeing. The NatBDI architecture offers an alternative for developing autonomous agents, featuring more transparent components and more controllable behaviours than agents based solely on machine learning models. Combining beliefs expressed as natural-language sentences with a planning language provides the first steps towards allowing humans with virtually no programming skills to instruct the agent in natural language.

Developing a BDI architecture with mental states that use a natural language representation opens multiple avenues for creating more efficient and trustworthy autonomous agents to address real-world problems. Regarding our approach to performing logical inference in natural language statements, we intend to extend this work as follows. First, we intend to improve the operational semantics of the NatBDI interpreter to increase robustness in the natural language plans, as well-known formalisms in BDI do [48, 73]. Given natural language plans that include lifted terms in their plan context and plan body steps, the reasoning cycle should perform substitutions analogously to traditional BDI approaches in first-order logic. Second, we intend to develop a method to automatically generate plans by recognising possible goals for the agent to achieve, inferring the plan context, and identifying the steps to include in the plan body. Generating natural language plans more autonomously reduces manual efforts in developing plans and maintains the premise of designing transparent, controllable and explainable agents.

References

[1] Ichida, A.Y., Meneguzzi, F., Cardoso, R.C.: BDI agents in natural language environments. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, pp. 880–888 (2024). International Foundation for Autonomous Agents and Multiagent Systems

[2] Winikoff, M., Sidorenko, G.: Evaluating a mechanism for explaining bdi agent behaviour. In: International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems, pp. 18–37 (2023). Springer

[3] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM computing surveys **55**(12), 1–38 (2023)

[4] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., *et al.*: A survey on evaluation of large language models. ACM transactions on intelligent systems and technology **15**(3), 1–45 (2024)

[5] Qian, P., Unhelkar, V.: Evaluating the role of interactivity on improving transparency in autonomous agents. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, p. 10831091 (2022)

[6] Verhagen, R.S., Neerincx, M.A., Tielman, M.L.: A two-dimensional explanation framework to classify ai as incomprehensible, interpretable, or understandable. In: International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems, pp. 119–138 (2021)

[7] Hirschberg, J., Manning, C.D.: Advances in natural language processing. Science **349**(6245), 261–266 (2015)

[8] Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J.,

Hausknecht, M., El Asri, L., Adada, M., *et al.*: Textworld: A learning environment for text-based games. In: Computer Games: 7th Workshop, Held in Conjunction with the 27th International Conference on Artificial Intelligence, pp. 41–75 (2019). Springer

[9] Hausknecht, M., Ammanabrolu, P., Côté, M.-A., Yuan, X.: Interactive fiction games: A colossal adventure. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 7903–7910 (2020)

[10] Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., Huang, X.: Pre-trained models for natural language processing: A survey. Science China Technological Sciences **63**(10), 1872–1897 (2020)

[11] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4171–4186 (2019)

[12] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research **21**(1), 5485–5551 (2020)

[13] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv **abs/1907.11692**(arXiv:1907.11692 [cs.CL]), 1–27 (2019)

[14] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In: Proceedings of the Advances in Neural Information Processing Systems, vol. 33, pp. 5776–5788 (2020)

[15] Tafjord, O., Dalvi, B., Clark, P.: ProofWriter: Generating implications, proofs, and abductive statements over natural language. In: Findings of the Association for Computational Linguistics, pp. 3621–3634. Association for Computational Linguistics, Online (2021). https://doi.org/10.18653/v1/2021.findings-acl.317 . https://aclanthology.org/2021.findings-acl.317/

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)

[17] Dziri, N., Lu, X., Sclar, M., Li, X.L., Jiang, L., Lin, B.Y., Welleck, S., West, P., Bhagavatula, C., Bras, R.L., Hwang, J.D., Sanyal, S., Ren, X., Ettinger, A., Harchaoui, Z., Choi, Y.: Faith and fate: Limits of transformers on compositionality. In: Conference on Neural Information Processing Systems, pp. 70293–70332 (2023). https://openreview.net/forum?id=Fkckkr3ya8

[18] Gendron, G., Bao, Q., Witbrock, M., Dobbie, G.: Large language models are not strong abstract reasoners. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 6270–6278. International Joint Conferences on Artificial Intelligence Organization, Jeju, South Korea (2024). https://doi.org/10.24963/ijcai.2024/693 . https://doi.org/10.24963/ijcai.2024/693

[19] Valmeekam, K., Marquez, M., Olmo, A., Sreedharan, S., Kambhampati, S.: Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In: Proceedings of the Advances in Neural Information Processing Systems, vol. 36, pp. 38975–38987. Curran Associates, Inc., ??? (2023)

[20] Valmeekam, K., Olmo, A., Sreedharan, S., Kambhampati, S.: Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In: NeurIPS Foundation Models for Decision Making Workshop, pp. 1–21 (2022). https://openreview.net/forum?id=wUU-7XTL5XO

[21] Stechly, K., Valmeekam, K., Kambhampati, S.: Chain of thoughtlessness? an analysis of cot in planning. In: Proceedings of the Advances in Neural Information Processing Systems, vol. 37, pp. 29106–29141 (2025)

[22] Nye, M., Tessler, M., Tenenbaum, J., Lake, B.M.: Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In: Proceedings of the Advances in Neural Information Processing Systems, vol. 34, pp. 25192–25204 (2021)

[23] Verma, M., Bhambri, S., Kambhampati, S.: Theory of mind abilities of large language models in human-robot interaction: An illusion? In: Companion of the ACM/IEEE International Conference on Human-Robot Interaction. HRI '24, pp. 36–45. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3610978.3640767 . https://doi.org/10.1145/3610978.3640767

[24] Rudin, C., Radin, J.: Why are we using black box models in ai when we don't need to? a lesson from an explainable ai competition. Harvard Data Science Review **1**(2), 1–9 (2019)

[25] Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents-components for intelligent agents in java. AgentLink News Letter **2**(1), 2–5 (1999)

[26] Bordini, R.H., Wooldridge, M., Hübner, J.F.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, Chichester, UK (2007)

[27] Singh, D., Hindriks, K.V.: Learning to improve agent behaviours in goal. In: Programming Multi-Agent Systems: 10th International Workshop, ProMAS, pp. 158–173 (2013). Springer

[28] Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence **4**(4), 349–355 (1988)

[29] Silva, L., Meneguzzi, F., Logan, B.: Bdi agent architectures: A survey. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 4914–4921. International Joint Conferences on Artificial Intelligence Organization, ??? (2020)

[30] Brafman, R.I., Tennenholtz, M.: R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research **3**, 213–231 (2002)

[31] Wang, R., Jansen, P., Côté, M.-A., Ammanabrolu, P.: ScienceWorld: Is your agent smarter than a 5th grader? In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 11279–11298. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates (2022). https://doi.org/10.18653/v1/2022.emnlp-main.775 . https://aclanthology.org/2022.emnlp-main.775/

[32] Moss, L.S.: Natural Logic, pp. 559–592. John Wiley & Sons, Ltd, Chichester, UK (2015). Chap. 18. https://doi.org/10.1002/9781118882139.ch18 . https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118882139.ch18

[33] Lakoff, G.: Linguistics and natural logic. Synthese **22**(1), 151–271 (1970) https://doi.org/10.1007/BF00413602

[34] Valencia, V.S.: Studies on natural logic and categorial grammar. PhD thesis, Universiteit van Amsterdam (1991)

[35] MacCartney, B., Manning, C.D.: An extended model of natural logic. In: Proceedings of the International Conference on Computational Semantics, pp. 140–156 (2009). Association for Computational Linguistics

[36] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall Press, Upper Saddle River, NJ, USA (2009)

[37] Bowman, S.R., Angeli, G., Potts, C., Manning, C.D.: A large annotated corpus for learning natural language inference. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 632–642. Association for Computational Linguistics, Lisbon, Portugal (2015)

[38] Clark, P., Tafjord, O., Richardson, K.: Transformers as soft reasoners over language. In: Proceedings of the Twenty-Ninth International Joint Conferences on Artificial Intelligence, pp. 3882–3890 (2020)

[39] Williams, A., Nangia, N., Bowman, S.: A broad-coverage challenge corpus for sentence understanding through inference. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1112–1122. Association for Computational

Linguistics, New Orleans, Louisiana, USA (2018)

[40] McCoy, T., Pavlick, E., Linzen, T.: Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics, pp. 3428–3448. Association for Computational Linguistics, Florence, Italy (2019)

[41] Maccartney, B.: Natural language inference. PhD thesis, Stanford University, Stanford, CA, USA (2009)

[42] Dagan, I., Glickman, O., Magnini, B.: The pascal recognising textual entailment challenge. In: Proceedings of the International Conference on Machine Learning, pp. 177–190 (2006). https://doi.org/10.1007/11736790_9 . http://dx.doi.org/10.1007/11736790_9

[43] Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Proceedings of the European Workshop on Modelling Autonomous Agents in a Multi-agent World, vol. 1038, pp. 42–55 (1996)

[44] Bratman, M.E.: Intention, Plans and Practical Reason. Harvard University Press, ??? (1987)

[45] Weiss, G.: Multiagent Systems. MIT Press, Cambridge, MA (2013)

[46] Cardoso, R.C., Ferrando, A.: A review of agent-based programming for multi-agent systems. Computers **10**(2), 16 (2021)

[47] Meneguzzi, F., Silva, L.: Planning in bdi agents: a survey of the integration of planning algorithms and agent reasoning. Knowledge Engineering Review **30**(1), 1–44 (2015)

[48] Rao, A.S.: Agentspeak (l): Bdi agents speak out in a logical computable language. In: European Workshop on Modelling Autonomous Agents in a Multi-agent World, pp. 42–55 (1996). Springer

[49] Sardina, S., De Silva, L., Padgham, L.: Hierarchical planning in bdi agent programming languages: A formal approach. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1001–1008 (2006)

[50] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge, MA (2018)

[51] Schut, M., Wooldridge, M., Parsons, S.: On partially observable mdps and bdi models. In: Foundations and Applications of Multi-Agent Systems, pp. 243–259. Springer, Berlin, Germany (2002)

[52] Ichida, A.Y., Meneguzzi, F., Ruiz, D.D.: Measuring semantic similarity between

sentences using a siamese neural network. In: Proceedings on the International Joint Conference on Neural Network, pp. 1–7 (2018). IEEE

[53] Gao, T., Yao, X., Chen, D.: Simcse: Simple contrastive learning of sentence embeddings. In: Proceedings on the Conference on Empirical Methods in Natural Language Processing, pp. 6894–6910 (2021). Association for Computational Linguistics (ACL)

[54] Ichida, A.Y., Meneguzzi, F.: Modeling a conversational agent using bdi framework. In: Proceedings of the ACM/SIGAPP Symposium on Applied Computing, pp. 856–863 (2023)

[55] Fuchs, N.E., Kaljurand, K., Schneider, G.: Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In: Proceedings of the International Florida Artificial Intelligence Research Society Conference, vol. 2006, pp. 664–669 (2006)

[56] He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., Ostendorf, M.: Deep reinforcement learning with a natural language action space. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics, pp. 1621–1630. Association for Computational Linguistics, Berlin, Germany (2016)

[57] Lin, B.Y., Fu, Y., Yang, K., Brahman, F., Huang, S., Bhagavatula, C., Ammanabrolu, P., Choi, Y., Ren, X.: Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 23813–23825 (2024)

[58] Shanahan, M.: Solving the Frame Problem: a Mathematical Investigation of the Common Sense Law of Inertia. MIT press, Cambridge, MA (1997)

[59] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45 (2020)

[60] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 1877–1901 (2020)

[61] Wu, T., He, S., Liu, J., Sun, S., Liu, K., Han, Q.-L., Tang, Y.: A brief overview of chatgpt: The history, status quo and potential future development. IEEE/CAA Journal of Automatica Sinica **10**(5), 1122–1136 (2023)

[62] Brutzkus, A., Globerson, A.: Why do larger models generalize better? a theoretical perspective via the xor problem. In: Proceedings of the International Conference on Machine Learning, pp. 822–830 (2019)

[63] Richardson, K., Hu, H., Moss, L., Sabharwal, A.: Probing natural language inference models through semantic fragments. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 8713–8721 (2020)

[64] Nestorovic, T., Matouek, V.: Modified conversational agent architecture. In: IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 682–689 (2013). https://doi.org/10.1109/ICTAI.2013.106

[65] Wong, W., Cavedon, L., Thangarajah, J., Padgham, L.: Flexible conversation management using a BDI agent approach. In: Intelligent Virtual Agents - International Conference. Lecture Notes in Computer Science, vol. 7502, pp. 464–470. Springer, ??? (2012). https://doi.org/10.1007/978-3-642-33197-8_48 . https://doi.org/10.1007/978-3-642-33197-8_48

[66] Nguyen, A., Wobcke, W.: An agent-based approach to dialogue management in personal assistants. In: Proceedings of the International Conference on Intelligent User Interfaces. IUI '05, pp. 137–144. Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1040830.1040865 . https://doi.org/10.1145/1040830.1040865

[67] Nguyen, A., Wobcke, W.: Extensibility and reuse in an agent-based dialogue model. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops, pp. 367–371 (2006). https://doi.org/10.1109/WI-IATW.2006.70

[68] Singh, D., Sardina, S., Padgham, L., Airiau, S.e.p.: Learning context conditions for bdi plan selection. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, pp. 325–332 (2010)

[69] Singh, D., Sardina, S., Padgham, L., James, G.: Integrating learning into a bdi agent for environments with changing dynamics. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1–6 (2011)

[70] Bosello, M., Ricci, A.: From programming agents to educating agents–a jason-based framework for integrating learning in the development of cognitive agents. In: International Workshop on Engineering Multi-Agent Systems, pp. 175–194 (2019). Springer

[71] Gatti, A., Mascardi, V., Ferrando, A., *et al.*: Let me talk to you! natural language interaction between humans and bdi agents via chatbdi. In: Proceedings of the European Conference on Artificial Intelligence, pp. 3646–3654. IOS Press, ??? (2025)

[72] Ciatto, G., Aguzzi, G., Battistini, R., Baiardi, M., Burattini, S., Ricci, A.: Exploiting genai for plan generation in bdi agents. In: Proceedings of the European Conference on Artificial Intelligence, pp. 3495–3502. IOS Press, ??? (2025)

[73] Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, pp. 470–481 (2002)