ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA251223

Exploiting GenAI for Plan Generation in BDI Agents

Giovanni Ciatto[a,*], Gianluca Aguzzi[a], Riccardo Battistini[a], Martina Baiardi[a], Samuele Burattini[a] and Alessandro Ricci[a]

[a]Department of Computer Science and Engineering
Alma Mater Studiorum—Università di Bologna, Italy
ORCID (Giovanni Ciatto): https://orcid.org/0000-0002-1841-8996, ORCID (Gianluca Aguzzi): https://orcid.org/0000-0002-1553-4561, ORCID (Martina Baiardi): https://orcid.org/0009-0001-0799-9166, ORCID (Samuele Burattini): https://orcid.org/0009-0009-4853-7783, ORCID (Alessandro Ricci): https://orcid.org/0000-0002-9222-5092

Abstract. Extending BDI agents with the ability to autonomously generate plans has long been a goal in the field of cognitive agent engineering to enhance their adaptability. Recent advances in GenAI are now opening new possibilities for plan generation, by leveraging the natural-language understanding, mean-end reasoning, and abstraction capabilities of LLMs. In this paper, we investigate the integration of GenAI-based plan generation into AgentSpeak(L) agents, and we analyse the implications of transferring knowledge between the LLM and the BDI agent, for the sake of dynamic plan generation. We propose a coherent framework where AgentSpeak(L) is extended with plan generation, and we model the boundaries of the generative process. We prototype our framework via the JaKtA BDI agent technology, and we methodologically assess the quality of the plans generated by LLMs of different sorts.

1 Introduction

Long before the current surge of interest in Generative AI (GenAI), the Belief-Desire-Intention (BDI) model has worked as a paradigm for rational agents, leading to the development of several BDI agent-oriented programming architectures [33], as well as agent programming languages based on them—most notably, AgentSpeak(L) [32]. BDI languages design rational agents able to reason about complex and dynamic environments, and proactively act upon them.

Nevertheless, traditional BDI programming frameworks typically lack mechanisms allowing agents to autonomously acquire or build new plans at runtime [11]. The procedural behaviour of AgentSpeak(L) agents is then limited to scenarios where the agent behaviour can be suitably defined at design time (pre-programmed): an effective option in terms of computational efficiency, which however constrains agent flexibility, responsiveness, and overall *autonomy* when facing unpredictable environments. Extending BDI agents with planning capabilities has thus been extensively explored [28].

Prior approaches typically attempt to bridge first-principles planning techniques with BDI architectures [11], requiring a detailed model of the action space. However, those modelsare typically costly and often unavailable for highly-dynamic environments.

Recent advances in GenAI methodologies promise to overcome such limitations by offering more dynamic and flexible solutions. GenAI technologies such as Large Language Models (LLMs) provide unprecedented opportunities for empowering cognitive agents with intelligent features involving the generation of cognitive abstractions (such as beliefs, goals, plans) as well as the ability to interact in natural language—just like human beings would do.

Along that line, we ground our exploration on the assumptions that: *(i)* LLMs exhibit planning capabilities [19]; *(ii)* LLMs easily handle natural language, and have absorbed *common sense* knowledge that can be leveraged in the planning process [8]; and, finally *(iii)* LLMs systems exhibit abstraction and *imagination* capabilities—e.g., [2], which may lead to more flexible plans than those generated by classical planning techniques. Furthermore, observing that BDI developers often rely on mnemonic and semantically-relevant names akin to natural language to represent cognitive abstractions, we suggest that LLMs can be exploited to generate BDI plans by simply relying on the natural-language semantics of goals, actions, and plans, rather than explicitly-defined preconditions and effects.

Hence, in this paper, we focus on the problem of augmenting BDI agents with the ability to autonomously generate *plans* by exploiting GenAI under the hood. In our exploration we aim to answer the following research questions: **(RQ1)** *what* information would LLMs require to generate BDI plans? **(RQ2)** *how* should knowledge be transferred between LLMs and BDI agents? **(RQ3)** *how* does automatic plan generation impact BDI agents operation and specification? **(RQ4)** *can* LLMs generate *reusable* BDI plans? To that end, we analyse requirements for integrating GenAI into the BDI architecture, propose a conceptual framework extending BDI for plan generation, and discuss preliminary results using a JaKtA [3] prototype.

Accordingly, the remainder of this paper is structured as follows. In Section 2, we provide background on BDI agents, related works on planning in BDI systems, and GenAI agents. In Section 3, we present our approach to the integration of a GenAI-based Plan Generation Procedure (PGP) into AgentSpeak(L) agents, followed by some practical remarks in Section 4. In Section 5, we present preliminary experiments, and in Section 6 we conclude and outline future work.

2 Background

Here, we recall the main notions behind BDI agents, plan generation in BDI agents, and GenAI agents.

* Corresponding Author. Email: giovanni.ciatto@unibo.it

2.1 BDI Agents and their Plans

The BDI model, inspired by human cognitive processes [5], and grounded upon a formal framework [34] for modelling *rational agents*' decision-making, has been adopted for agent programming languages (e.g., [4]) as well as for simulating intelligent agents [20] or human behaviour [1]. BDI systems represent a wide yet specific class of Multi-Agent Systems (MASs), where rational agents feature *cognitive* abstractions such as beliefs, goals, and intentions.

Specifically, a BDI agent [33]: *(i)* maintains *beliefs* about itself and its environment, updated by perception, reasoning, or communication; *(ii)* is guided by the *desires* (or goals) it aims to achieve, which may evolve over time or generate other desires to pursue; *(iii)* commits to multiple concurrent *intentions* to fulfil its desires; *(iv)* executes *plans* (i.e., sequences of *actions*) in order to achieve its goals.

Several architectures have been proposed to implement BDI agents in software, including AgentSpeak(L) [33], Procedural Reasoning System (PRS) [23], and Distributed Multi-Agent Reasoning System (dMARS) [12]. In the following, we focus on the widely-adopted AgentSpeak(L)—so that we may use the terms BDI agents and AgentSpeak(L) agents interchangeably in the following.

AgentSpeak(L) agents perform so-called *procedural reasoning* [23]: they are equipped by human developers with procedural knowledge in terms of a *plan library*, a collection of plans agents can select for execution at run time. Agents' deliberation process is then aimed at selecting the most adequate plan for any event that may occur, failing if none has been provided.

AgentSpeak(L) plans are referred to as *plan rules*, denoted as: `E : C <- A1; ...; An.` The rule describes how the agent should react to some *triggering event* `E`, stating for instance what to do when a new percept arises, or how to achieve a given goal. `C` is the set of conditions under which the rule is applicable (a.k.a. the rule's *context*), and `A1; ...; An` is the *body* of the rule, corresponding to the actual plan, i.e., the actions to perform to handle the event.

Plan rules differ significantly from first-principles planning, where the plan is a sequence of actions to be performed to reach a desired world state—i.e., *declarative* goals [42]. In fact, although it is possible to write plans rules and goals in a declarative style to mimic classic planning [21, 18], it is often the case that a goal expressed as the triggering event of a BDI plan rule is simply a *mnemonic name* of an event the agent may react to—i.e., *procedural* goals [42].

Similarly to goals, actions are basically *mnemonic names* of functionalities causing effects. Unlike other Artificial Intelligence (AI) sub-fields, though, pre-/post-conditions for an action are not explicitly represented in AgentSpeak(L) as the actions are usually invoking low-level software/hardware facilities that bring side effects about. Typically, in BDI agents, actions may even fail due to unpredictable environment dynamics making pre-/post-conditions hard to define.

Lastly, a notable feature of AgentSpeak(L) plan rules is that the *body* of a plan may contain subgoals—i.e., statements triggering the selection and execution of other plans. Through this mechanism, AgentSpeak(L) agents can pursue complex goals by breaking them down into smaller subgoals for which the specific plans are selected at runtime, resulting in a non-linear execution. Yet, the underlying assumption is that all plans are known in advance, and the agent can select them based on the current context.

2.2 Plan Generation in BDI Agents

The main limitation of procedural reasoning in BDI agents is that the agent programmer has to anticipate all possible situations the agent may encounter. Although this is beneficial for the agent's predictability and controllability, it limits the agent's flexibility and adaptability to unforeseen situations. For this reason, the BDI community has long been interested in the problem of *automatic plan generation*, incorporating techniques from AI planning into the BDI architecture. Reference [28] nicely summarises the many approaches proposed in the literature so far. For this paper, we highlight the main implications of integrating plan generation techniques into BDI agents.

In the previous section we have already surfaced the main differences between BDI plans and classical (a.k.a. first-principles) planning (e.g., STRIPS [14]) which can be essentially be summarised as: *(i)* BDI plans and goals are procedural, while classical planning are declarative [42]; *(ii)* BDI actions have no clearly stated pre- and post-conditions, while classical planning actions are defined by them *(iii)* BDI plans can include subgoals for a more flexible execution flow, while classical planning is usually assembling a linear sequence of actions to move from one world state to another. Hierarchical Task Network (HTN) planning [16], based on the idea of task decomposition, allows generating more abstract plans that can mimic the invocation of subgoals addressing *(iii)*. Nevertheless, tasks must be defined in advance, have pre- and post-conditions and potentially ordering constraints. Interestingly, as emerging from [28], classic planning techniques have mostly been integrated to generate new plans, while HTN have been exploited to perform lookahead in the plan library and understand if a suitable chain of plans is available.

Regardless of the specific planning technique, integrating plan generation in BDI agents requires deciding *when* to trigger the generation process. The most straightforward approach is to trigger the generation process when the agent is unable to find a suitable plan in its library. Another common approach is to trigger the generation through an agent action that can be invoked intentionally by the agent programmer as part of a plan as introduced in [29]. The plans generated can then be used to extend the original plan library, or be executed immediately. In the first case, plans should ideally be reused in the future [11], although this may lead to a bloated plan library and conflicts with the original plans that need to be handled [7].

2.3 GenAI Agents

Recent advances in GenAI have sparked significant interest in developing agents that incorporate LLMs, thanks to their remarkable ability to perform effectively across a diverse range of tasks without specialised training for each domain [6]. In these systems, LLMs serve as the core cognitive engine or "brain," providing capabilities in natural language understanding, reasoning, planning, and knowledge recall [31, 38, 41]. This integration enables the development of agents capable of complex, emergent behaviours, often described under paradigms like "generative agents", "agentic AI" [30] or "autonomous agents" in the Natural Language Processing (NLP) community. Several approaches have emerged within this broad category.

One common approach focuses on creating 'general-purpose agents' [30] capable of executing actions, managing a memory, and using software tools with minimal human intervention. Works following this approach, e.g. Auto-GPT [43], demonstrate that LLMs can autonomously decompose goals into actionable steps.

Another approach aims to situate LLMs in physical environments. For instance, Voyager [40] represents a significant step towards lifelong learning agents by autonomously exploring the complex environment of Minecraft, acquiring new skills through interaction and feedback, and storing them in a skill library for future use. Similarly, SayCan [17] and PaLM-E [13] integrate LLMs with robotic control,

using them to interpret high-level natural language instructions and generate action plans for execution.

Rather than proposing entirely new agent architectures, some works focus on enhancing the reasoning capabilities of existing ones. The ReAct framework [44] enables LLMs to interleave reasoning (generating thought traces) and acting (e.g., tool use), enabling agents to dynamically plan, execute, and adjust based on observations. Similarly, Reflexion [37] introduces agents that can reflect on past failures, maintain dynamic memory, and use self-reflection to improve their decision-making capabilities over time.

While these approaches focus on enhancing reasoning, the generative agent architecture underlying "Generative Agents" [31] provides a blueprint for single agents with believable, human-like, means-end reasoning capabilities. These agents leverage memory structures (capturing past observations and deliberations) and LLMs-driven planning to simulate daily routines, form relationships, and generate emergent social dynamics from experiences and environment.

Finally, regarding BDI integration, several approaches explore combining BDIs principles with LLMs. The work by [22] proposes NatBDI, a BDI agent architecture designed for natural language environments. It leverages LLMs, specifically for Natural Language Inference (NLI), within the BDI reasoning cycle to determine the applicability of plans based on natural language beliefs and context descriptions, rather than generating the plans themselves. Another direction, presented by [24], uses the BDI structure itself to create structured prompts ("BDIPrompting") for LLMs. This guides the LLM to generate more proactive and explainable task plans by framing the planning problem in terms of beliefs, desires, and intentions directly within the prompt given to the LLM. Another interesting approach is [15], which involves LLMs to enhance conversation capabilities between BDI agents and humans.

Our approach instead focuses on the integration of generative capabilities into the well-established AgentSpeak(L)-based architecture. Instead of replacing the BDI paradigm or merely using LLMs to guide reasoning directly, we embed generative capabilities in terms of plan generation into the BDI architecture, ensuring that these capabilities complement (rather than replace) the fundamental AgentSpeak(L) architecture, with the goal of maintaining the cognitive structure of the agent architecture to preserve the controllability, predictability, and explainability of the agent's behaviour while leveraging generative capabilities [35]. The ability for "old-school" agents to interrogate an LLM on-demand to adapt their plans have been recently proposed in the context of Web environments [36]. Our approach instead focuses on the generation of reusable plans from the agent's existing knowledge, rather than relying on it for selecting actions to perform among those dynamically discovered at runtime.

3 Generative Plan Generation for BDI Agents

In this section, we detail how BDI agents can exploit LLMs to *dynamically* synthesize actionable plans that directly address distinct goals identified by the agent. To this end, we equip each agent with a *Plan Generation Procedure (PGP)*, which the agent may trigger to create one or more plans to be added to its own plan library.

Conceptually speaking, such a PGP involves *(i)* meticulously encoding the agent's current cognitive state and operational context – specifically: available actions, existing sub-plans, comprehensive beliefs, and the active goals the agent seeks to accomplish – into a structured prompt comprehensible to the LLM, and *(ii)* parsing the LLM output to extract the generated plans. We first address the question of *how* should such a PGP work, and *when* the should it be triggered (cf. Section 3.1), and how the overall AgentSpeak(L) architecture should be adapted to accommodate generative capabilities.

In our design, the PGP functionality works as follows: it accepts a goal `G` as input and it returns a non-empty set of plans $\mathcal{P}$, such that at least one plan $p \in \mathcal{P}$ handles the event `+G`. The plans in $\mathcal{P}$ are subject to the following constraints: **(C1)** each generated plan $p \in \mathcal{P}$ must handle different contexts (i.e., there should never be two plans $p, p' \in \mathcal{P}$ such that $p \neq p'$ yet both have the triggering event `E` and context `C`); **(C2)** the generated plans' contexts and bodies *should* refer to other goals or beliefs that are already known to the agent, if these help pursuing the goal `G`; **(C3)** yet they may also reference novel, unknown goals or beliefs; **(C4)** the generated plans' bodies *must not* refer to actions which are unknown to the agent.

The rationale behind these constraints is straightforward: the PGP should handle the goal it was triggered for, and it should reuse existing knowledge *whenever possible*. In particular, if the generated plans need to affect the environment, they should do so by leveraging the actions the agent is already aware of—which were supposedly provided by the agent programmer(s) in advance. This implies that the PGP should keep into account which and how much information is available to the agent, and use it in the generation process. Yet, the PGP should also be open to the generation of new knowledge in the form of beliefs, and goals. This may open up to the possibility, for the agent, to reuse the PGP multiple times, recursively generating hierarchies of interrelated plans.

Example. From now on, we rely on a running example. Consider the case of the AgentSpeak(L) agent depicted in Figure 1. The agent is located in a grid 2D environment, where it can move in 9 admissible directions (i.e., the 8 cardinal directions plus `here`, which implies no movement), via the action `move(Direction)`, unless there is an obstacle or a border in the way. In any given moment, the agent can perceive the environment, expressed as in Figure 1c.

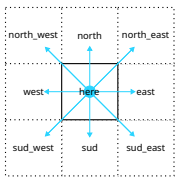
Assuming that the agent is initially located into some cell where object `home` is not even perceived, and that it has the initial goal to `reach(home)`, yet it has no plan to achieve this goal. In such a situation, the agent may trigger the PGP to figure out what to do to handle goal `!reach(home)`. A good implementation of the PGP would generate a set of plans $\mathcal{P} = \{p_1, p_2, p_3\}$ such as:

```
p1 ≡ +!reach(Obj) : there_is(Obj, here).

p2 ≡ +!reach(Obj) : there_is(Obj, Dir) <-
       move(Dir).

p3 ≡ +!reach(Obj) : ¬there_is(Obj, _) <-
       ?random_free_dir(Dir); move(Dir); !reach(Obj).
```

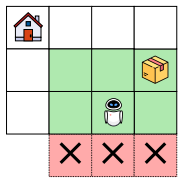
which are actually good to `reach` any object `Obj`—not just `home`. There, plan p_1 handles the success case where the object `Obj` is already in the agent's current position, then there is nothing to do, as the initial goal has been already achieved. Plan p_2 handles the case where the object is perceived in the surroundings of the agent, in some `Dir`ection: the agent can now achieve the initial goal in one single `move`ment. Finally, plan p_3 handles the case where the object is not perceived in the surroundings of the agent, which means the agent has to explore the environment to find it. The exploration is then attempted via a dummy strategy: *randomly* choosing a free `Dir`ection, then `move`ing and recursively pursuing `!reach(Obj)`.

From a technical perspective, the PGP generates a plan p_1 and p_2 by simply combining goals, beliefs, and action names that are already known to the agent. However, in generating plan p_3, the PGP also *invents* a new goal `?random_free_dir(Dir)`, which is not known to the agent either at compile or run time. Semantically, this is a *test* goal, aimed at selecting one random obstacle-free direction,

Figure 1: Environment modelling for our running example. The agent moves in a discrete 2D grid with 8 cardinal directions (north pointing up) plus `here` for the current position. Obstacles, free cells, and objects in surrounding cells are perceived via beliefs.



(a) Nine admissible directions



(b) Example of environment model

```
free(north).
free(north_west).
free(north_east).
free(west).
free(east).
obstacle(south).
obstacle(south_west).
obstacle(south_east).
there_is(box, north_east).
```

(c) Perceptions of the agent in fig. 1b

yet, technically, this is a goal for which the agent has no plan.

Before executing plan p_3, the agent may trigger the PGP again, to generate a plan for the subgoal `?random_free_dir(Dir)`. This time the PGP may generate a plan p_4 such as:

$p_4 \equiv$ `+?random_free_dir(Dir) : free(Dir) ∧ Dir≠here.`

Now the agent has all the plans it needs to achieve the initial goal `!reach(home)`, so it can proceed executing them.

3.1 When Should Plans Be Generated?

In other words, when should the agent trigger its PGP? Abstracting from technicalities, and taking inspiration from prior literature on plan generation in BDI agents, we can identify two main approaches to the triggering of PGP, namely: the reactive and the proactive ones.

On-demand PGP. The on-demand PGP is modelled as an *action* that the agent can invoke whenever it needs to generate a plan. Technically speaking, the agent is assumed to have a built-in action `generate_plan(E)` in its action repertoire, which it can invoke whenever it needs to generate a plan for event `E`. As an ordinary AgentSpeak(L) action, `generate_plan` may appear in the body of any plan, and be exploited by programmers to build smarter agents. The action would just require the agent to provide some actual event `E` as input, and its effect would be the production of a plan of the form `E <- C : ...`, where `C` is the context of the plan, to be added to the invoking agent's plan library. Any failure in the PGP execution results in a failure of the action, handled by the agent as it would do for any other action. The underlying assumption here is that the programmer decides when the agent should trigger the PGP.

As a consequence of constraint **(C2)**, the PGP may generate plans that use the `generate_plan` action in their body. So, generated plans may themselves trigger the PGP to generate new plans, when executed. This powerful feature allows the agent to hierarchically generate plans, possibly decomposing complex plans into simpler ones, and so on recursively—provided that the underlying LLM is smart enough to understand when it is wise to postpone the generation of a (sub-)plan, and when it is better to generate it immediately.

Reactive PGP. The reactive PGP is implicitly triggered by the agent's control loop whenever an event `E` occurs and the agent has no applicable plan for it. So, an unknown event would trigger the PGP instead of making the current intention fail. If the PGP fails, the current intention fails as when there is no plan were available; if it succeeds, the agent proceeds as if the plan existed from the start.

Provided that agents are equipped with the aforementioned `generate_plan` action, one may consider implementing the reactive approach as shown below:

$p_{genex} \equiv$ `-G : missing_plan_for(G) <- generate_plan(G); G.`

There, we borrow Jason's extended AgentSpeak(L) syntax (cf. [4]) for failure handling plans to express a plan (namely, p_{genex}) reacting to the failure event provoked by the absence of a plan for goal `G`, to which the agent should react by generating a plan for it, and finally pursue goal `G` one more time. The underlying assumption here is that agents are equipped by default with plan p_{genex}, as well as with a special belief `missing_plan_for(G)`, which computes whether plans are missing in the agent's plan library for goal `G`.

The reactive approach improves the on-demand approach by making the generation procedure automatic, and therefore implicit. Neither the programmer nor the LLM generating plans needs to explicitly trigger the PGP: the agent will handle this automatically when it encounters a goal without an associated plan during execution.

The reactive approach as well supports the generation of hierarchies of plans, in a way which is similar to the example from Section 3. The main difference here is that the LLM is not required to be aware of the recursive nature of the planning activity it is involved into. So, it can focus on the problem suggesting the best subgoals and actions to be performed to pursue a given goal.

3.2 Concurrency and PGP

Regardless of how/when the PGP is triggered, any implementation approach should consider that the PGP is inherently *blocking* for the agent. In fact, at the current state of technology, LLMs are commonly queried via Web services (contacted over the Internet), and they may take up to many seconds to complete their response (or even fail, e.g. due to network issues). Even by assuming that the BDI agent and the LLM are running on the same machine, the LLM may require entire seconds to generate a plan. Large time scales may be unacceptable for BDI agents, especially if waiting for the PGP to terminate implies blocking the agent's control loop. This would hinder the agents' ability to react to events in a timely manner, and therefore its autonomy.

To mitigate this problem, implementers should consider *suspending* only the intention that triggered the PGP, rather than the whole control loop of the agent. As multiple intentions may be concurrently carried on by an AgentSpeak(L) agent, this would allow it to "keep thinking about how to react to event `E`, while doing something else".

4 Transferring Knowledge between BDI agents and LLMs

Here we delve into the practical aspects of endowing BDI agents with a PGP, we discuss how to effectively engineer the prompts for LLM to let it generate plans, and how to write agent specifications in AgentSpeak(L), to enrich them with additional knowledge that can be exploited by the new generative capabilities.

4.1 From BDI Agents to LLMs and Back

In practice, how should the PGP operate to work as discussed so far?

Technically, when triggered for a goal `G`, the PGP should: *(i)* convert the agent's internal state, as well as `G`, into the textual prompt

used to query the LLM of choice, then *(ii)* parse the LLM response to extract the generated plans. The specific LLM chosen does not impact the design of the PGP itself, yet it may affect the quality of the generated plans—as we further discuss in Section 5.

So, *prompt generation* (structured information into free text) and *parsing* (free text back into structured information) are the two main *complementary* activities of PGP. They may be implemented in several ways, mostly differing in terms of *what to encode*, and *how to encode* it, in the LLM prompt and in the response.

4.1.1 What to Encode

The Response. The PGP should produce a set of BDI plans by querying some LLM, which in turn produces a textual response. The response represents a (possibly empty) collection of plans, where each plan should involve: *(i)* a triggering event, *(ii)* a context – i.e., a (possibly empty) collection of conditions to test –, and *(iii)* a body—i.e. a (possibly empty) collection of subgoals and actions.

The Prompt. To support the aforementioned plans generation, the PGP should provide all (and possibly only) relevant information to the LLM via some *automatically*-generated prompt, following a typical zero-shot prompting approach [25], therefore encoding instructions and information about the agent's current state in natural language and in structured form, thus addressing **(RQ1)**.

Conceptually speaking, relevant information should include: **(I1)** the intended meaning of the goal `G` for which the plans are needed – plus the explicit request to generate plans for it –, **(I2)** goals, actions, and beliefs known to the agent, **(I3)** the agent's currently handled plans, goals, and beliefs; as well as instructions on **(I4)** what is the intended outcome of the plan generation process, **(I5)** the AgentSpeak(L) syntax and meaning, **(I6)** how to impersonate a BDI agent willing to generate a plan (i.e., role-playing prompting [26]), and **(I7)** what a BDI agent is in the first place.

Whereas the LLM needs **(I1)** to know what goal the agent is willing to pursue – in our running example, the goal `!reach(home)` – **(I2)** is required to know which goals, actions, or beliefs (akin to tool specifications in ReAct [44]) the LLM could use to generate the plans. For instance, some generated plans may check for obstacles in the surroundings of the agent via belief `obstacle(Dir)`, even if the agent is currently not perceiving any obstacle.

For the LLM to account for the context currently perceived by the agent when generating the plans, **(I3)** is needed. For instance, beliefs such as `there_is(box, north_east)` and `obstacle(north)` should be included here. **(I4)** informs the LLM about the purpose and the constraints of the plan generation process, as defined in Section 3. This is where, for instance, the possibility to formulate new goals and beliefs should be mentioned—along with the impossibility of creating new actions.

(I5) lets the LLM comply with the AgentSpeak(L) syntax, when both reading the prompt and producing its response, and makes it aware of how cognitive abstractions are modelled in AgentSpeak(L). This is where, for instance, the LLM is informed about the three parts of a plan (event, context, and body), and the components of a plan body (subgoals and actions). Along with **(I7)**, – where the general notions of beliefs, desires, and intentions are provided – the core idea here is to provide the LLM with enough background knowledge to operate in the BDI domain, regardless of whether BDI literature has been included in the training set of the LLM or not.

Finally, **(I6)** provide the LLM with meta-level suggestions about how to devise plans in general. There, recommendations such as "decompose the task hierarchically" or "re-use the known goals and actions whenever useful" could be included.

It is worth noticing that, while **(I1)**, **(I2)**, and **(I3)** are specific to the agent's current state, **(I4)**, **(I5)**, **(I6)**, and **(I7)** are more general, and could be factoreised across different runs of the PGP.

4.1.2 How to Encode **(RQ2)**

In principle flexibility in syntax is allowed for both the prompt and the response, provided that *(i)* the syntax used for the prompt is easily manipulable by the LLM, *(ii)* the syntax used for the response is easily parsable by some non-LLM-based parser. A complete example of a prompt satisfying our recommendations is provided in the supplementary materials [10].

To achieve *(i)*, we recommend maximizing the exploitation of natural language (as done in related works in LLM as a "planner" [30])— which does not mean discarding AgentSpeak(L) syntax, but rather complementing it with natural language descriptions. For instance, when describing the goal `!reach(Obj)` in the prompt, a natural language description like "the agent wants to reach a situation where `Obj` is perceived in its surroundings". Similarly, the admissible belief `there_is(Obj, Dir)` could be described as "the agent can perceive the presence of `Obj` in direction `Dir`". Likewise, this applies to every admissible/current goal, belief, or action.

To achieve feature *(ii)*, we recommend using a syntax that is as structured as possible, where the boundary between one generated plan and the next one is easily identifiable, and each section of the plan (event, context, operations) is clearly delimited. This could be achieved by using the AgentSpeak(L) syntax directly, or, if the LLM is not good at it (likely to happen for small LLMs) by using a more famous syntax, such as Prolog or YAML. So, for instance, one may ask the LLM to generate plans matching the following template:

```
EVENT:
  verb: event
CONDITIONS:
  - condition
  - other_conditon
  - ...
OPERATIONS:
  - verb: operation1
  - verb: operation2
  - ...
```

where *verb* is a keyword among `execute`, `achieve`, `test`, `add`, `remove`, and `update`. This format can be parsed by programming languages, is human-readable and debuggable, and can be generated by the LLM, as it aligns with well-known syntaxes (e.g., YAML).

It is worth mentioning that a description of the expected response syntax should be included in the prompt to let the LLM know what is expected from it. This is acknowledged as the most common way to control the shape of the LLM response, and it is commonly referred to as *structured output* [39]. In our perspective, the "structured output instructions" are yet another information to be included in the prompt, along with the other items discussed above.

4.2 Writing Generative Agent Specifications

When BDI agents are generative and can generate their own plans, one may wonder how the role of the programmer changes, to account for such new capabilities. In other words, do agents still need programmers to write their plans? Given the current state of technology, we argue that programmers are still needed to write the initial plans of the agent, and they are also required to provide hints about the intended semantics of the goals, beliefs, and actions they write. The latter point in particular (writing hints) is the main focus of this

section. Namely, we aim to show how developers can provide additional domain knowledge to the LLM in an easily maintainable way, alongside the agent's specification, addressing **(RQ3)**. This focuses on transferring knowledge to the LLM to improve plan generation.

In the general case, it is not guaranteed that LLMs will be able to understand the intended meaning of the goals, beliefs, and actions they are asked to generate plans for/with. This is because there is no guarantee that BDI programmers would use intuitive and self-explanatory names for them, or, that the LLM is trained on the BDI programming language of choice. For this reason, we designed the encoding procedure (cf. Section 4.1.1) to include some natural-language description of each aspect of the agent's internal state.

Yet, descriptions cannot be automatically generated – as automatic procedures may be inaccurate or incomplete w.r.t. the programmer's intent – so programmers need to write them directly. Descriptions should be written in natural language and explain the meaning of the *admissible* and *actual* goals, beliefs, and actions each agent has.

Accordingly, our PGP-equipped BDI agent requires one final assumption: agent programs should be written in a way that *(i)* natural-language descriptions for admissible and actual mental abstractions are part of the agent's code, and *(ii)* those descriptions are used to generate the prompt for the PGP. To this end, the AgentSpeak(L) framework, and consequently BDI programming languages, should be extended with ad-hoc syntactical constructs for tagging cognitive abstractions with their intended meaning. Furthermore, to minimize the programmer's burden, these constructs should be optional, and allow for templating and reusing descriptions.

Consider for instance the `obstacle(Dir)` beliefs in Figure 1c. When generating an LLM prompt for them, the PGP should *(i)* describe the general meaning beliefs of the form `obstacle(Dir)` – namely: "an obstacle is perceived in direction `Dir`" – and *(ii)* describe the specific meaning of each belief of this sort that the agent currently has—namely: "there is an obstacle in direction north". Yet, there is no need for the programmer to write all these descriptions by hand, as they can be automatically generated out of templates.

We prototype these ideas in JaKtA [3], an AgentSpeak(L)-based programming language implemented as a Kotlin Domain-Specific Language (DSL)—therefore easily extensible with additional syntactical constructs. In JaKtA, the new entries are (cf. Section C in [10]): *(i)* the `.meaning{...}` blocks, which can be used to tag *actual* goals, beliefs, or actions with their intended meaning, via a post-fix syntax; and the *(ii)* the `admissible{...}` blocks, which can be used to tag *admissible* goals or beliefs with their intended meaning, via a pre-fix syntax. In both cases, the meaning is expressed as a string, attained via string interpolation, which in turn may include the functor, the arguments, and the presence/absence of negation of the tagged goal, belief, or action. The strings are then exploited by the PGP to generate the prompt for the LLM.

5 Experimental Evaluation

Here we assess our approach empirically, by evaluating the performance of the PGP in generating plans for BDI agents, by indirectly assessing the quality of the generated plans.

Experimental Setup We implement the reactive PGP approach as described in Section 3 in a JaKtA agent [3], whose specification is enriched with natural-language descriptions as discussed in Section 4.2. The experiment is publicly available[1] and archived on Zenodo [9]. The agent has to reach home within a simulated 2D grid world environment where it can perceive obstacles, with no pre-programmed knowledge. The setup mirrors our running example throughout the paper. We conduct the experiment across multiple popular and publicly available LLM models (specifically: `gpt-4.1`, `gemini pro 2.5`, `deepseek V3`, and `claude 3.7 thinking`), iterating each experiment 5 times per model to account for the stochastic nature of LLM responses, resulting in a total of 20 experimental runs.

Metrics To assess the quality of generated plans we combine automatic and manual evaluation procedures, as our goal is to assess hard properties of the generated plans (e.g., syntactical correctness or functionality, which can be automatically tested), but also soft properties (e.g., relevance, usefulness, readability, etc.) which are hard to formalize and automatically infer. To evaluate the quality of generated plans, we define the following metrics:

- *Plan Count* (**PC**): Total number of generated plans.
- *Context Complexity* (**CC**): Average number of beliefs per plan context.
- *Plan Body Complexity* (**PBC**): Average number of operations per plan body.
- *Generalization Count* (**GC**): Number of generated plans that use variables (general) rather than constants (specific).
- *Redundancy Amount* (**RA**): Amount of generated plans which are useless (e.g., not executable at runtime or subsumed by more general plans).
- *Novel Goal Count* (**NGC**): Number of newly-invented goals.
- *Novel Belief Count* (**NBC**): Number of newly-invented beliefs.
- *Goal Semantic Misalignment* (**GSM**): Number of semantically-misaligned uses of admissible goals.
- *Belief Semantic Misalignment* (**BSM**): Number of semantically-misaligned uses of admissible beliefs.
- *Task Success Rate* (**TSR**): Percentage of experiments where the agent successfully achieves the `!reach(home)` goal.

Among these metrics, **GC** and **TSR** should be maximised (higher values are better), while **RA**, **GSM**, **BSM**, should be minimised (lower values are better). The remaining metrics (**PC**, **CC**, **PBC**, **NGC**, and **NBC**) have no clear optimization direction, yet they provide valuable comparative insights, with both extremely low and high values potentially indicating issues.

Additionally, to establish a comprehensive evaluation metric for plan quality, we implement a **Plan-Reference Alignment Score (PRAS)** comparing generated plans against expert-designed reference plans. For this purpose, we exploit an LLM explicitly as a judge [27], tasked with evaluating the quality of abstraction, generalizability, and adherence to BDI principles in the generated plans compared to reference implementations—see Section D in [10]. This LLM-as-judge methodology allows us to systematically rate plan quality using criteria that would be difficult to formalise through traditional metrics alone. **PRAS** tends to 1 when the generated plans are similar to the reference ones, to 0 when they are completely different.

Results Table 1 and [10] (Section E) summarise the results of our experiments across different LLM models in terms of the defined metrics, whereas Figure 2 provides a visual representation of the **PRAS** scores for each model. In [10] (Section B) we provide an example of a generated plan. Task Success Rate (**TSR**), our primary effectiveness indicator, varied considerably across models (Table 1). GPT-4 demonstrated perfect performance with 100% **TSR**, producing consistently usable plans that successfully completed the task in all experimental runs, thereby matching the optimal baseline. Claude

[1] github.com/pslab-unibo/experiment-2025-ECAI-generative-bdi-agents

Table 1: Experimental results across different LLM models. These number are averaged over 5 runs per model.

Model	PC	CC	PBC	GC ↑	RA ↓	NGC	NBC	GSM ↓	BSM ↓	TSR ↑
Claude	4.80	1.86	1.59	2.80	0.00	0.80	0.20	0.00	0.00	60%
Deepseek	5.00	1.52	1.96	1.00	0.40	1.20	1.60	0.20	1.00	40%
GPT 4	2.00	1.90	1.50	2.00	0.00	1.00	0.00	1.00	0.00	100%
Gemini Pro	3.80	2.00	1.71	3.80	0.00	0.20	0.80	0.00	0.00	20%
Optimal	3.00	1.00	1.30	3.00	0.00	0.00	0.00	0.00	0.00	100%

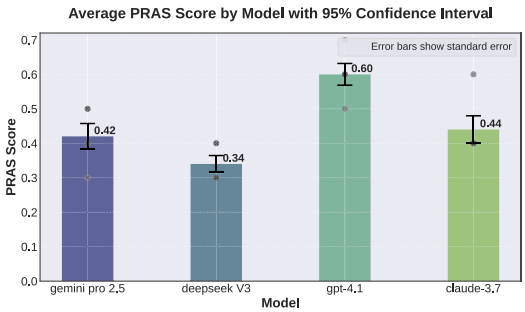


Figure 2: **PRAS** across different LLM models.

achieved moderate success (60% **TSR**), while Deepseek (40%) and Gemini Pro (20%) struggled with task completion.

Another key indicator of plan quality, namely the **PRAS** score, indicated that GPT-4 generated plans most closely resembling expert-designed reference plans (0.60), suggesting a high degree of reusability (also due to its high **TSR**). Conversely, the other models exhibited lower **PRAS** scores (< 0.50), which may indicate a divergence from reference implementations. These lower scores suggest differences in plan structure and strategy compared to expert-designed solutions, potentially reflecting model-specific reasoning patterns or limitations in interpreting the semantic constraints of the BDI architecture. Despite generally positive alignment scores, plans generated by most LLMs diverged significantly from the baseline in terms of structure and complexity, as evident from the results. Complexity analysis showed that LLMs generally produced more elaborate solutions than the optimal baseline (**PC**=3.00, **CC**=1.00, **PBC**=1.30). Most models generated higher plan counts (Deepseek: 5.00, Claude: 4.80), context complexity (Gemini Pro highest at 2.00), and plan body complexity (Deepseek highest at 1.96), showing a tendency toward richer conditions and longer plans, than the minimal optimal strategy.

Interestingly, however, Gemini Pro displayed the highest Generalization Count (3.80) among LLMs, surpassing even the optimal **GC** (3.00), indicating a tendency to create abstract, variable-using plans. However, this high generalization did not translate into effectiveness, given its low **TSR**. Conversely, Deepseek showed a significant Redundancy Amount (0.40), likely lowering its **TSR**, while GPT-4 and Claude produced no redundant plans (**RA** = 0.00).

For cognitive construct invention, Deepseek was most prolific (**NGC** = 1.20, **NBC** = 1.60), followed by GPT-4 (**NGC** = 1.00, **NBC** = 0), while Claude and Gemini Pro showed minimal invention. This highlights LLMs' abstraction capabilities, though it can lead to unnecessary complexity, as seen in Deepseek's redundant plans.

Finally, the semantic alignment metrics assess the correct usage of pre-defined admissible goals and beliefs, leveraging the provided natural language descriptions. Lower scores are better, indicating less misalignment. Claude and Gemini Pro achieved perfect scores (**GSM/BSM** = 0.00). However, GPT-4 showed misalignment in goal usage (**GSM** = 1.00), and Deepseek in belief usage (**BSM** = 1.00), suggesting these models, despite their capabilities, occasionally failed to correctly interpret or apply the semantics of existing agent knowledge based on the hints provided.

Summary Addressing the research question *can LLMs generate (re)usable BDI plans?*, our experiments show that LLMs can indeed generate usable plans, but the quality and effectiveness (namely, reusability) of these plans vary significantly across models. While GPT-4 achieved perfect task completion (100% **TSR**) with a reasonable alignment with human-generated plan (**PRAS** = 0.60), Gemini Pro demonstrated the highest Generalization Count (3.80) but paradoxically showed poor task success (20% **TSR**). This suggests that plan reusability does not automatically translate to effectiveness. The tendency of LLMs to generate more complex plans than necessary (higher **PC**, **CC**, **PBC** than optimal) and occasionally invent new abstractions presents both opportunities and challenges. Overall, LLMs can generate (re)usable BDI plans, but their quality and effectiveness are highly model-dependent, with tradeoffs between generalization, complexity, and utility.

6 Conclusion

In this paper, we explore integrating GenAI into the AgentSpeak(L) agent architecture to exploit LLMs for generating new plans at runtime, based on the agent's current knowledge. Our goal is to assess if and how LLMs can generate plans for BDI agents, potentially reducing reliance on human programmers or first-principle planners.

We encapsulate the LLM within a PGP functionality, for reactive or on-demand plan generation, ensuring BDI architecture compatibility. We define the PGP interface, propose implementation guidelines, and evaluate a prototype to validate our approach.

Addressing **(RQ1)**, we define our PGP to use structured prompts detailing current and generally *admissible* goals and beliefs, available actions and plans, and the operational semantics of the BDI architecture. For **(RQ2)**, structured prompts paired with parseable outputs ensure reliable integration of generated plans into the agent's library, balancing expressiveness with formal rigor. We further speculate that the knowledge transfer can be improved, by providing additional semantics to the LLM via natural language descriptions of the agent's goals, beliefs, and actions.

Consequently, answering **(RQ3)**, automatic plan generation shifts the agent specification from exhaustive plan encoding to providing the basic (procedural) knowledge necessary for the problem domain, potentially leaving to the LLM the task of handling corner cases and unexpected situations. The agent operates like a classic BDI agent, with the possibility of the PGP to be triggered explicitly (on-demand) or implicitly (reactively) to handle goals with no matching plans.

For **(RQ4)**, our methodology is promising for generating reusable plans with variables, despite the variability in LLMs performance.

Future work includes exploring runtime validation and verification mechanisms to address hallucinations or inaccuracies in LLM-generated constructs. Studying prompt robustness under varying conditions and developer inaccuracies could enhance the methodology's practicality. Ablation studies may clarify factors influencing prompt effectiveness **(RQ1)** and knowledge transfer **(RQ2)**. Testing scenarios with concurrent goals, dynamic environments, and partial observability will validate scalability and generalization **(RQ4)**. Finally, incorporating plan repair, failure learning, and adaptive refinement further enhances agent autonomy and long-term performance.

This research bridges traditional BDI architectures and emerging GenAI technologies, laying the groundwork for more autonomous, adaptable, and explainable cognitive agents.

Acknowledgements

This work was partially supported by: PNRR – M4C2 – Investment 1.3, Partenariato Esteso PE00000013 – "FAIR—Future Artificial Intelligence Research" – Spoke 8 "Pervasive AI" and "WOOD4.0 - Woodworking Machines for Industry 4.0", (CUP E69J22007520009) Emilia-Romagna regional project, call 2022, art. 6 L.R. N. 14/2014.

References

[1] C. Adam and B. Gaudou. BDI agents in social simulations: a survey. *The Knowledge Engineering Review*, 2016. doi: 10.1017/S0269888916000096.

[2] V. Aregbede et al. Affordance-based goal imagination for embodied ai agents. In *International Conference on Development and Learning (ICDL)*, 2024. doi: 10.1109/ICDL61372.2024.10644764.

[3] M. Baiardi, S. Burattini, G. Ciatto, and D. Pianini. Blending BDI agents with object-oriented and functional programming with JaKtA. *Springer Nature Computer Science (SNCS)*, 2024. doi: 10.1007/s42979-024-03244-y.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, Oxford, England, 2007.

[5] M. Bratman et al. *Intention, plans, and practical reason*, volume 10. Harvard University Press, Cambridge, MA, 1987.

[6] S. Bubeck et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *ArXiv*, 2023. doi: 10.48550/ARXIV.2303.12712.

[7] R. C. Cardoso, L. A. Dennis, and M. Fisher. Plan library reconfigurability in BDI agents. In *Engineering Multi-Agent Systems (EMAS)*, 2019. doi: 10.1007/978-3-030-51417-4_10.

[8] G. Ciatto, A. Agiollo, M. Magnini, and A. Omicini. Large language models as oracles for instantiating ontologies with domain-specific knowledge. *Knowledge-Based Systems*, 2025. doi: 10.1016/j.knosys.2024.112940.

[9] G. Ciatto, G. Aguzzi, R. Battistini, M. Baiardi, S. Burattini, and A. Ricci. Experiments for the paper "Exploiting GenAI for Plan Generation in BDI Agents", 2025. URL https://doi.org/10.5281/zenodo.16744409.

[10] G. Ciatto, G. Aguzzi, R. Battistini, M. Baiardi, S. Burattini, and A. Ricci. Supplementary material for the paper "Exploiting GenAI for Plan Generation in BDI Agents", 2025. URL https://doi.org/10.5281/zenodo.16600196.

[11] L. de Silva, S. Sardiña, and L. Padgham. First principles planning in BDI systems. In *Autonomous Agents and Multiagent Systems (AAMAS)*, 2009. doi: 10.5555/1558109.1558167.

[12] M. D'Inverno et al. The dMARS Architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 2004. doi: 10.1023/B:AGNT.0000019688.11109.19.

[13] D. Driess et al. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning (ICML*, 2023. doi: 10.5555/3618408.3618748.

[14] R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971. doi: 10.1016/0004-3702(71)90010-5.

[15] A. Gatti, V. Mascardi, and A. Ferrando. ChatBDI: Think BDI, talk LLM. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2025. doi: 10.5555/3709347.3743930.

[16] I. Georgievski and M. Aiello. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 2015. doi: 10.1016/J.ARTINT.2015.02.002.

[17] R. Hazra, P. Z. D. Martires, and L. D. Raedt. SayCanPay: Heuristic planning with large language models using learnable domain knowledge. In *AAAI Conference on Artificial Intelligence*, 2024. doi: 10.1609/AAAI.V38I18.29991.

[18] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. C. Meyer. Agent programming with declarative goals. In *Agent Theories Architectures and Languages (ATAL)*, 2000. doi: 10.1007/3-540-44631-1_16.

[19] X. Huang et al. Understanding the planning of LLM agents: A survey. *ArXiv*, 2024. doi: 10.48550/arXiv.2402.02716.

[20] J. F. Hübner and R. H. Bordini. Agent-based simulation using BDI programming in Jason. In *Multi-Agent Systems - Simulation and Applications*. 2009. doi: 10.1201/9781420070248.CH15.

[21] J. F. Hübner, R. H. Bordini, and M. J. Wooldridge. Programming declarative goals using plan patterns. In *Declarative Agent Languages and Technologies (DALT)*, 2006. doi: 10.1007/11961536_9.

[22] A. Y. Ichida, F. Meneguzzi, and R. C. Cardoso. BDI agents in natural language environments. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2024. doi: 10.5555/3635637.3662942.

[23] F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 1992. doi: 10.1109/64.180407.

[24] M. Jang et al. A structured prompting based on Belief-Desire-Intention model for proactive and explainable task planning. In *International Conference on Human-Agent Interaction (HAI)*, 2023. doi: 10.1145/3623809.3623930.

[25] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In *International Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[26] A. Kong et al. Better zero-shot reasoning with role-play prompting. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2024. doi: 10.18653/v1/2024.naacl-long.228.

[27] H. Li et al. LLMs-as-judges: A comprehensive survey on LLM-based evaluation methods. *ArXiv*, 2024. doi: 10.48550/ARXIV.2412.05579.

[28] F. Meneguzzi and L. de Silva. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, 2015. doi: 10.1017/S0269888913000337.

[29] F. R. Meneguzzi and M. Luck. Leveraging new plans in AgentSpeak(PL). In *Declarative Agent Languages and Technologies (DALT)*, 2008. doi: 10.1007/978-3-540-93920-7_8.

[30] S. Murugesan. The rise of agentic AI: implications, concerns, and the path forward. *IEEE Intelligent Systems*, 2025. doi: 10.1109/MIS.2025.3544940.

[31] J. S. Park et al. Generative agents: Interactive simulacra of human behavior. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2023. doi: 10.1145/3586183.3606763.

[32] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*. Springer, 1996. doi: 10.1007/BFb0031845.

[33] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *International Conference on Multi Agent Systems (ICMAS)*, 1995.

[34] A. S. Rao and M. P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 1998. doi: 10.1093/logcom/8.3.293.

[35] A. Ricci, S. Mariani, F. Zambonelli, S. Burattini, and C. Castelfranchi. The cognitive hourglass: Agent abstractions in the large models era. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2024. doi: 10.5555/3635637.3663262.

[36] S. Schmid, M. Freund, and A. Harth. Adaptive planning on the Web: Using LLMs and affordances for Web agents. In *Knowledge Graphs and Semantic Web (KGSWC)*, 2024. doi: 10.1007/978-3-031-81221-7_7.

[37] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: language agents with verbal reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[38] T. R. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024.

[39] X. Tang et al. Struc-bench: Are large language models good at generating complex structured tabular data? In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2024. doi: 10.18653/v1/2024.naacl-short.2.

[40] G. Wang et al. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024, 2024.

[41] L. Wang et al. A survey on large language model based autonomous agents. *Frontiers in Computer Science*, 2024. doi: 10.1007/S11704-024-40231-1.

[42] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & procedural goals in intelligent agent systems. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M. Williams, editors, *International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, 2002.

[43] H. Yang, S. Yue, and Y. He. Auto-gpt for online decision making: Benchmarks and additional opinions. *ArXiv*, 2023. doi: 10.48550/ARXIV.2306.02224.

[44] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.