POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Navigation of a language model's latent space

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Daniele Dente, 10696464

Advisor:
Prof. Marcello Restelli

Co-advisor:
Gianvito Losapio

Academic year:
2023-2024

Abstract: Large language models have impressive language capabilities, but are still largely treated as black boxes. In order to use them reliably and safely, we would like to be able to control their output, and gain an intuitive understanding of their internal processes.
Several methods have tried to induce a given output, from modifying the original model to purely black box approaches, with varying results. In this work we will study the information contained in the embeddings, which are the latent representations used by these models. We develop a semantic embedding scheme and model embedding space navigation as a Markov Decision Process. We define a pipeline to efficiently suggest phrases that lead the LLM towards a given output, by training a reinforcement learning agent based on these latent vectors. The training is done via a modified version of Alphazero to take advantage of the explorative capabilities of this algorithm. We use the open model Mistral-7B-Instruct and sample goals from the English dictionary, obtaining modest but promising results, getting tokens that lead the generation towards the specified goal. The method can be extended in a goal-based setting and can also be applied to safety or alignment tasks.

Key-words: reinforcement learning; natural language processing; large language models

Contents

1. Introduction

Large language models (LLMs) have revolutionised the Natural Language Processing world, and in recent years they have embedded themselves in many people's lives, either as text assistants for individuals[21, 45] or as parts of larger frameworks and pipelines in several industrial processes[1, 14]. Such models are capable of diverse language tasks, like translation[53] or summarisation, as well as tasks in other fields, such as arithmetic and logical reasoning[49].

Unfortunately, they also have a potentially harmful impact if used for illicit or deceptive purposes, given their powerful generation capabilities[50], and an understanding of their inner working is crucial to tame this new technology.

One of the most nebulous aspects in large language model computations is the meaning of the *embeddings* they utilise, i.e, the internal numerical representations of the text that is given in input and which are used to output an answer in return. These embeddings compress down information in order to be re-elaborated in the hidden layers of the LLM to obtain the next part of the sentence, and such dense information might be useful to understand, and hopefully control, the text generation of these models.
Since we are looking to use this information to *control* these systems, we will be using the tools of **reinforcement learning** (RL), as its techniques are used when it is necessary to find the optimal control strategy in a given environment.

The main research question we ask ourselves is the following: *"Is it possible to treat a language model as a navigable environment and explore the latent space of its embeddings, akin to what is done in reinforcement learning?"*

The motivation for this question comes from a necessity to better understand the workings of such obfuscated models. We want to find possible methods of controlling their outputs towards a desired state or a desired set of outputs: learning this can allow us to gain insight into how to avoid such a steering of the response from a bad actor.

Our work will try to combine the strengths of large language models and reinforcement learning in order to exert control over the generation of current LLMs. We strive to employ both the rich structure of LLMs' outputs and the control enabled by reinforcement learning to offset the weaknesses of these two fields.
We will also follow this leitmotiv of finding promising common grounds throughout the various methods we are going to consider. As a natural language task we will try to traverse the space of conversations, which requires a fair bit of exploration. Because of this we will be basing ourselves off of the tree algorithm known as Montecarlo Tree Search, as this algorithm allows searching an environment towards the most promising directions, essential in a large space like that of conversations.

2. Background

2.1. Large language models

A large language model is an artificial intelligence model that is able to produce semantically meaningful text. Most such models are based on the Transformer [48] architecture, a structure that exploits the concept of attention[3, 26] to its fullest.
Most widely available language models process text by first breaking it up in **tokens** learned during the training process. The most common way for building up the vocabulary of the model is through Byte Pair Encoding[13], providing a good compression ratio for the incoming text and making processing much faster compared to using only one or several starting character sets. The learned tokens will make up the **vocabulary** $\mathcal{V}$, the set of discrete units of text the model will work with.
Language models come in two main flavours: *autoencoding* and *autoregressive*. The necessity for compressing text comes from the way in which language models generate text: in both autoregressive and autoencoding models, we need to generate some missing parts of the sentence, requiring a forward pass of the system, and reducing the number of times the computation needs to happen is vital for efficiency.

As for the two types of LLMs, the main representative of an autoencoding language model (also called a "masked model") is BERT[12] (Bidirectional Encoder Representations from Transformers), whose self-supervised training consists in reconstructing a sentence in which some of the tokens have been masked. These models are able to construct a semantically rich representation of the input sentence in order to output the missing token, but are not made for lengthy generation.
The most notable autoregressive model is the GPT family of models (Generative Pretrained Transformer)[35]. These models autoregressively generate the next token given the previous ones. Their embeddings are more

task-oriented and are not guaranteed a semantically meaningful result, but in turn are great for generation.

The internal representation of transformers is made up of **embedding vectors** that are progressively modified during the forward pass through the LLM's layers. These embeddings, vectors in $\mathbb{R}^d$, encode the meaning and position of the input tokens, modified along the transformer architecture, and are dependent on the training process of each model.
After reaching the last layer of the model, the final embedding is projected to a vector of size $|\mathcal{V}|$, making up the logits of each token. These logits will then be converted to probabilities, most often with a simple *softmax* function, and then follow a *sampling strategy* to determine what the next token will be.

Current models are able to perform well on reasoning benchmarks, but despite the appearance, they don't have an inherent notion of correctness or planning, since their training focuses on prediction, trying to match the training corpus as closely as possible.
The latest models are able to perform surprisingly well in tasks other than language completion [5, 9, 31, 52], but lacking the structures for proper reasoning, they are prone to "hallucinations"[16], confident but wrong answers obtained by unfortunate mixing of the training data.
These hallucinations, as well as the many concerns about alignment[19, 23] are the main obstacle for reliable use of these technologies.

2.2. Reinforcement learning

Reinforcement learning is a branch of machine learning dealing with control problems. It is based on the notion of an agent operating in an environment in order to find the best policy π to achieve its goals, most often encoded in a single reward signal[46].
This formalism was developed to mimic the learning process of real life intelligent agents, which experience the environment and receive either a reward or punishment based on their actions.

2.2.1 Components

A reinforcement learning model is defined by a **Markov Decision Process** (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu_0)$ which consists of few main components:

- $\mathcal{S}$ - The *state space*, a set of states the agent can visit
- $\mathcal{A}$ - The *action space*, a set of actions the agent can take
- $\mathcal{P}$ - A *transition function* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ that maps a state and an action taken in that state to a distribution over the next states
- $\mathcal{R}$ - A *reward* signal $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ which the agent will use to optimise its performance in the environment, maximizing the total reward received
- $\gamma \in [0, 1)$ - A *discount factor*, determining by how much the rewards are discounted at future timesteps. Because the reward signal, imitating real life intelligent agents, is more valuable in the near future, we *discount* future rewards based on how far away in time they are
- μ_0 - An *initial distribution* on the set of states

The process can be of finite, indefinite or infinite *horizon*.
When dealing with a finite horizon MDP, the process will surely end in H steps. In the indefinite and infinite case, the process ends after an indefinite amount of steps (upon reaching an absorbing state) or never ends, respectively. In this case we can consider γ as taking the role of the horizon. The *effective horizon* in this case is calculated as $\frac{1}{1-\gamma}$. Alternatively, γ can be seen as the probability of the episode to carry on.
MDPs exploit the *Markov property*, meaning that the history of the episode is irrelevant to calculate the next state if the current state is fully known. More formally

$$P(s_{t+1} = j | s_t = i, s_{t-1} = k_{t-1}, \cdots, s_0 = k_0) = P(s_{t+1} = j | s_t = i)$$

The basic loop of reinforcement learning is shown in Fig. 1.

Depending on the specific RL problem at hand, there can be slightly different components to this modelisation. In the case of *goal-oriented reinforcement learning*, we also add a set $\mathcal{G}$ of goals to the formulation. The reward function is also determined by the goal $g \in \mathcal{G}$, and the agent is conditioned to learn multiple goals instead of a single one.
The rationale behind this framework is that intelligent agents should be able to generalise their knowledge of the environment to different situations, and multiple different goals are one such type of possible generalization.

The state of the MDP can also differ from that known to the agent. When the environment does not expose the full state to the agent, but only a partial observation o, we are dealing with a *Partially observable Markov decision process* (POMDP). In this case we add two components to our initial MDP:

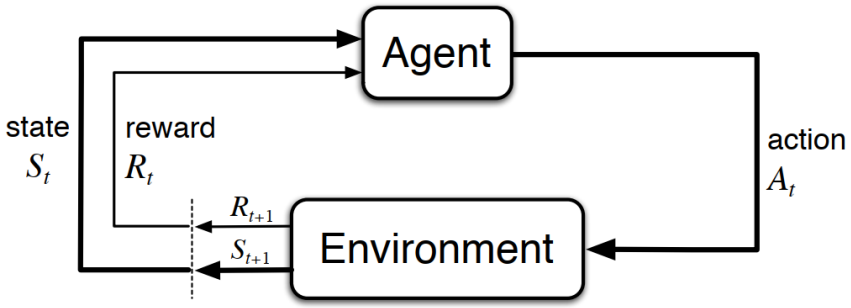


Figure 1: Reinforcement learning loop (From [46])

- Ω - The *observation space*, a set of observation that the environment can provide to the agent
- $\mathcal{O}$ - An *observation function* $\mathcal{O} : \mathcal{A} \times \mathcal{S} \to \Delta(\Omega)$ that given an action of the agent and the (true) state the environment is now in, returns a probability distribution over all possible observations

POMDPs are significantly more difficult to learn than standard MDPs[33]. The intuition for this can be observed by the fact that even a simple two-state MDP can become a process with an infinite number of states, since the agent's view of the environment is a probability distribution over these two states. Another complication is that in POMDPs the Markov property does not hold, as the agent's state is a function of all previous observations.

2.2.2 Value functions and policies

In a generic Markov Reward Process (A simpler version of a MDP where the transition probabilites are independent of actions) we can define the combination of the obtained rewards through a specific metric, the *gamma-discounted cumulative reward*, also called the *return*:

$$v_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k}$$

From this return we define the value of a state in the environment through the **value function** as

$$V(s) = \mathbb{E}[v_t | s_t = s]$$

Learning which states are high-value states is essential for solving a MDP, and the object that needs to be learned in order to reach that is the **policy**.

A policy in the context of reinforcement learning is a function $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ mapping a state to a distribution over actions. We can define the value function induced by a given policy as

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[v_t | s_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) \middle| s_t = s\right], \quad a_{t+k} \sim \pi(s_{t+k})$$

In order to learn a useful policy to traverse the environment, the function that is needed to exert this control is the **action-value function**, also called *Q-function*, since it offers information about the action that the value function lacks.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[v_t | s_t = s, a_t = a]$$

The goal of reinforcement learning is to learn the optimal policy π^* that maximises the expected return from the starting state.

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[V^{\pi}(\mu_0)\right]$$

This often comes down to learning the optimal value functions V^* and Q^*, and for this purpose several state of the art algorithms have been developed[27, 39, 40].

In many environments the state space, the action space or both of these are not finite in their cardinality, meaning that it is impossible to enumerate all state-action pairs in order to find the best action-value function. In such cases we must resort to *function approximation* to learn the desired policy, parametrizing the value functions via a set of parameters w. Oftentimes the parametrization brings us in the realm of deep learning, which is responsible for some impressive results like in [28, 44] by using deep neural networks as the actor (policy) and critic (value function).

The main hindrance to reinforcement learning algorithms is their sample complexity, i.e. the number of interactions that are needed with the environment, which is often time-consuming, costly, or unsafe.

Another consideration regarding sample complexity is that the training data in reinforcement learning is a time series, and as such it is inherently temporal. This means that samples are not *independent and identically distributed* (i.i.d.).
Yet another problematic aspect is that when updating the parameters of the functions, the policy changes, and in turn the new samples that will be obtained. The non-stationarity of the learning targets is yet another recurring problem in reinforcement learning, hampering learning through training mechanisms of other machine learning subfields.

2.2.3 Exploration/Exploitation dilemma

When engaging with the environment it is important to explore in order to gather new information that can be used later, but it is also necessary to exploit the previously obtained information at some point to garner rewards from the environment. The exact amount of exploration is a dilemma that is endemic to reinforcement learning, even more so when the space is so vast that it would be unfeasible to explore all possible states, for example in continuous settings.

2.3. Montecarlo Tree Search

Montecarlo Tree Search (MCTS) is a tree-based technique for evaluating a given environment through search. Like all tree algorithms we begin with a root node, identifying the initial state of the environment from which we want to plan out. After that, the algorithm is based on a sequence of four elementary steps:

Selection. A node of the tree is chosen, depending on how "promising" it seems. Starting from the root, the nodes are compared based on their UCT[20] value (Upper Confidence bound for Trees), a score that tries to balance exploration and exploitation, the ever-present dilemma in reinforcement learning. This value for node n is calculated as

$$UCT(n) = \frac{W(n)}{N(n)} + C\sqrt{\frac{\log(N(Parent(n)))}{N(n)}}$$

where $W(n)$ is the cumulative utility obtained by node n, $N(n)$ is the number of simulations that have involved n, $Parent(n)$ is the parent node of n and C is an exploration factor that governs how much weight to assign to exploration and how much to exploitation: a value of 0 ignores any explorative aspect and focuses on the obtained rewards only, while a higher value consideres much more favourably visiting less explored nodes.

Expansion. Once selected, a node is expanded, meaning that one or several new children nodes are created by applying an action to the state contained in the parent node.

Rollout. From each child, a rollout is executed. This means simulating the continuation of the game until the end (or any other stopping criterion), and from there receive a reward depending on the outcome. The policy used during the training can differ from the tree policy, and it often is just a random policy.

Backup. The outcome of the rollout is propagated to the ancestors, in order to obtain a finer estimate of the best action to take at the root.

As with all "Montecarlo methods" using this algorithm requires reaching a final state during rollouts in order to obtain a reward, which can be quite rare for environments where the episode length is very long.

2.4. Alphazero

The history of Alphazero[44] comes from the evolution of AlphaGo[42] and AlphaGo Zero[43]. Both of these monumental papers involve the usage of MCTS in order to reach milestone achievements at the time, such as beating the world champion at Go, a game with over 10^{170} possible configurations.
This algorithm uses MCTS combined with a deep convolutional neural network acting as policy and value functions, parameterised by parameters θ. The input of the networks is the state of the game board, and in output is a policy head and a value head. These two outputs predict what the next best move is, according to the network, as well as what is the predicted return from the current state.
The way in which they are trained is by trying to fit the values predicted from the network to the values obtained by MCTS, which can "*be viewed as a powerful policy improvement operator*"[43].
When performing the search in the tree, these values are then used both to guide the expansion process, operating an implicit pruning, and to return a predicted value during the rollout step instead of carrying out the simulation. The calculation of the UTC score is also changed in order to use the PUCT algorithm[37]:

$$PUCT(n) = \frac{W(n)}{N(n)} + C\pi_\theta(a_n|s_n)\frac{\sqrt{(N(Parent(n)))}}{N(n)+1}$$

Here $\pi_\theta(a_n|s_n)$ represents the prior probability of the network to choose the action that leads it to the node n. This way the algorithm can weigh the branches also by the predicted utility, especially useful in settings in which the branching factor or in general the action space is very large.
The guidance provided by the policy network enables far less searching steps than traditional search algorithms.

3. Previous works

A connection with large language models and reinforcement learning is already present: RL is already commonly used in the last parts of LLM training, to try to align the model to human values via Reinforcement Learning with Human Feedback (RLHF)[32]. This type of use of reinforcement learning is focused on the alignment aspect, rewarding the generation based on a human-provided metric to avoid unsafe output, tweaking the weights of the LLM in the process. In this case, the LLM itself is the agent.
Our use of RL will differ from the aforementioned one, since we will treat the language model as a fixed environment, and train an external agent to obtain a specific output from said environment.

The main methods to steer a language model's output have been through *fine-tuning* or *prompt engineering*. Fine-tuning provides optimal results by using a fine-tuning dataset, but unfortunately requires additional data and retraining of the network, which is computationally intensive.
Many research avenues have thus dealt with the concept of **prompt engineering** in order to obtain a desired result from language models.
In [4], LLM prompting is viewed in the framework of control theory, showing that it is possible, in a few tokens, to induce any given completion, even the least likely one, to be the most likely. This provides a substantial theoretical basis in the effort to find these tokens via automatic prompt engineering.

3.1. Prompt engineering

The main directions of prompt engineering focus on either of the following: manual optimisation, "soft prompts", token-level optimisation, or sentence-level optimisation.

The first **manual prompt optimisation** works have been aimed at probing the capabilites of LLMs[5, 38], but obviously require manual effort to dedicate to each task.
Another similar technique to avoid fine-tuning is to use "in-context learning", i.e., providing the model with some examples prior to evaluation. These can achieve very good results in steering the discussion, but are not robust with respect to updates to the model. The "DAN" (Do Anything Now) family of prompts is a prime example, suggesting an "unrestricted mode" to the model that it should comply with. Most models also suffer from sycophancy, leading to exploits such as "many-shot jailbreaking"[2].

When speaking of **soft prompts** we are referring to the bypassing of the textual prompt input in the LLM, instead feeding in a manually crafted embedding vector[24, 34].

Several techniques have been proposed to instead utilise the continuous embedding or probability space visible when querying the hidden states of LLMs by employing **token-level** attacks. This means optimising over a set of tokens and see which ones elicit the better response.
An important first work in this avenue is AutoPrompt[41]. In this framework, a prompt template allows for a certain amount of tokens to be learned in order to improve the classification performance of a masked language model. This method uses as its main training signal the probability of the correct token(s) being output, and already it provides a great boost in prompting performance over previous methods.
Subsequent research has tried to obtain similar prompts that are transferrable across multiple LLMs [54], some even adopting reinforcement learning for this aim, like RLPrompt[11], which obtains trigger tokens for a given downstream task by inserting a policy network in the intermediate layers of the model.

Others have opted for **sentence-level** methods, optimising over input prompts which result in natural sounding conversation.
Many practitioners try to exploit the roleplaying and simulated personas that LLMs can offer, emulating the "DAN" series of prompts[25], or pitting LLMs against each other in an adversarial fashion to obtain the desired response [6, 8, 51].

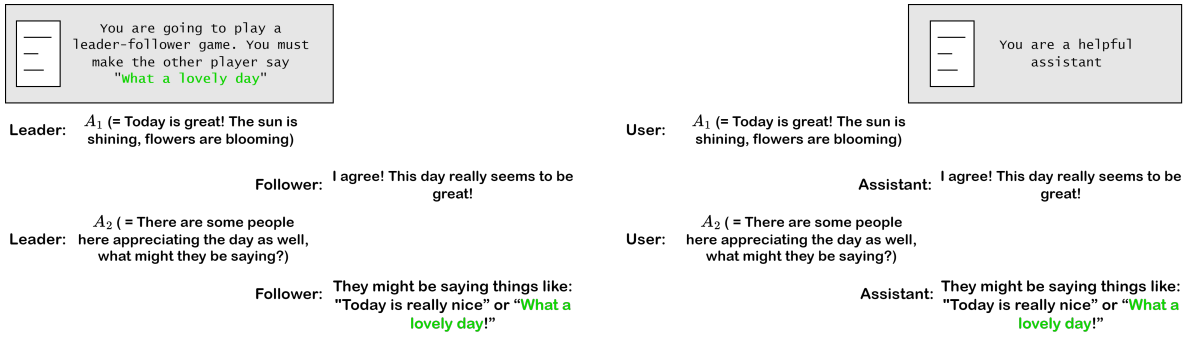
All these methods have their own pros and cons. Manual prompts are harder to craft as time and defensive measures go on, both due to more controlled pre-training and post-inference censorship[15], which will likely make their influence fade in the coming years.

The first automatic way, using soft prompts, implies the highest degree of control over the steering, which happens in continuous space, but is also the least interpretable, as they wholly operate in the embedding space and often don't map to an input sentence of any kind.
The second is able to exert a good amount of control, but suffers from the necessity of a white-box model and the common non-interpretability of the resulting prompts, usually made up of random tokens.
The third usually uses a black-box model and is very interpretable since it operates in sentence-space, although it is very dependent on the initialization prompt and has limited control, since it does not delve into the inner mechanisms of LLMs.
We are going to be positioning ourselves in between the last two approaches, learning in a token-level setting while using the low-level information given from the model but using it to generate sentence-level interactions.

3.2. Embeddings

Another avenue of research in LLMs focuses on obtaining meaningful embeddings. This effort is easier for masked type models such as BERT, which show that it is possible to get semantically meaningful results out of the compression that the model operates, especially in the middle layers[36].
The research in causal models is more troubled, as embeddings do not originate from a necessity to reconstruct a given sentence, but to extend it. Because of this difference in architecture, it has been attempted to bring the model to converge to similar embeddings to their masked counterparts with contrastive learning[18] or simple meta-prompts[22], as well as trying to use some combination of the raw internal embeddings[47]. These techniques aim to use the information contained in the hidden layers of the transformer for classification tasks, but to the best of our knowledge they haven't been used for LLM attacks or jailbreaking without a modification to the model itself. We will instead define a way to obtain embeddings that can be used as state by an external agent, requiring access to the model but without modifying it, a *grey-box* setting.

4. Methods



(a) Leader view (b) Follower view

Figure 2: Interaction example

We want to explore the embedding space of an LLM in order to automatically decide what utterances can lead it to a desired output, all the while maintaining interpretability of the resulting suggested actions.
In our approach we will treat the reference model as a learnable environment, basing ourselves off of the embeddings generated within the model by probing it during inference but leaving the parameters of the LLM frozen. This approach thus doesn't require any expensive retraining of the language model and doesn't use it as a policy, contrary to previous works.

In order to have multiple targets be available, we will also structure the problem in a goal-oriented fashion. The idea is that being able to steer a model towards a certain utterance requires conscious understanding of the current context (embedding), which is a highly generalizable concept: knowledge for reaching the word "dungeon" is likely useful for reaching "dragon" or "medieval".

The main setting is inspired from the *Adversarial Taboo*[51] game. In this two-player game, the attacker needs to induce the defender into saying a certain word. Only the attacker knows of this word, and the defender must try to guess it without accidentally saying it during the discussion. In our proposal we decide to train a RL agent as the attacker, and instead of an adversarial setting, we treat the defender as part of the environment, more similar to a leader-follower setting.

We are going to be using a single LLM L in order to assess the information that can be gained when limiting ourselves to just one model. As such, both the leader and the follower will be different instances of L, in order to keep the outputs result from the same embedding space.
We will denote as $L(x)$ the output of the LLM on the prompt x until the occurrence of the EOS (end of sequence) token. These are normally non-deterministic outputs, unless we decide to use greedy sampling in the generation, which we will use during training.

Example - L("The capital of Germany") = "is Berlin."

We will also denote as $\text{Emb}(x)$ the embedding of the prompt x under the strategy described in 4.3.

4.1. Embedding space navigation as a Markov Decision Process

The first combination that we will highlight is the one between **language models and reinforcement learning**.
As mentioned before, large language models appear to have impressive language capabilities, but as they are essentially next-token predictors they lack any reasoning aspect in their architecture. Despite this shortcoming, we still want to use their natural language proficiency, thus letting a reinforcement learning agent dictate what actions to take.
To build an environment for this task, we will need to define a MDP $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{R}, H, \gamma, \mu_0)$.

- **State space** $\mathcal{S}$ - Our state will be the current embedding describing the discussion, meaning the state space is $\mathbb{R}^d$. As embeddings usually encode meaning in their direction rather than their magnitude, and since we are going to use the cosine similarity as the main metric in our modelisation, we can normalise the embeddings, limiting the state space to the d-dimensional spherical surface $\mathcal{S} = \{s \in \mathbb{R}^d : \|s\|_2 = 1\}$. The real state of the environment s_E is a series of utterances in natural languages, meaning $s_E \in \mathcal{V}^\infty$.

  Despite the embedding only being an observation of the true state of the environment, we are going to use the state space we just defined as a proxy for the actual real state of the conversation. We postulate that the continuous nature of embeddings gives enough freedom to represent even lengthy conversations, for all practical purposes, and because of this we justify this substitution. Working in continuous space also allows for easier manageability of the training process.[1]
- **Goal space** $\mathcal{G}$ - Our approach aims to learn a policy based on the semantic embedding of the discussion in order to induce a certain target. The goal space of this work would then be the space of natural language sentences $\mathcal{V}^\infty$.
- **Action space** $\mathcal{A}$ - The action space for a language modeling task would be, in general, the set of all possible utterances, meaning an unbounded discrete set of $\mathcal{V}^\infty$. To deal with the size of such a set, we decide to limit the actions of the agent to k tokens, so that the action space is only $\mathcal{A} = \mathcal{V}^k$. The structure of such an agent is explained in 4.2
- **Transition function** $\mathcal{P}$ - Once an action is obtained, it is appended at the end of the discussion, and then the utterance of the follower is sampled and appended in a similar fashion.

  $$s_E^{t+1} = s_E^t.a^t.\text{Del}_F.a_F^t.\text{Del}_L$$

  Here $a.b$ is the concatenation between a and b.
  The follower's action is taken by directly querying the LLM: $a_F^t = L(s_E^t.a^t.\text{Del}_F)$. Between each player's turns we append a delimiter to mark the current turn. Del_F is the follower's delimiter, and Del_L is the leader's. The embedding of the new state of the discussion is going to be the next state for the agent.

  $$s^{t+1} = \text{Emb}(s_E^{t+1})$$

  In the beginning we have $s_E^0 = \text{Sys}_L.\text{Del}_L$
- **Reward** $\mathcal{R}$ - The reward given to the agent depends on whether it was able to induce the target sentence g in the follower's response. If a_F^t contains the target, the reward will be $r^{t+1} = +10$ and the episode will end.
  As this reward by itself is quite sparse, and since we want to use the information from the embedding vectors, we also define a reward at each timestep in the following way, also exemplified in Fig. 3.
  For action at time t we calculate the embedding for the discussion with the real follower response, $e_r^t = \text{Emb}(s_E^t.a^t.\text{Del}_F.a_F^t)$ and the one for the ideal response $e_i^t = \text{Emb}(s_E^t.a^t.\text{Del}_F.g)$. We then shape a reward function based on the *cosine similarity* of these two vectors, specifically

  $$r^{t+1} = 4(\text{sim}(e_r^t, e_i^t) - 1)$$

[1] To be more precise, the agent receives an observation o from the environment and uses that as its state, we assume that the function mapping the environment's state to the agent's is roughly a bijection, and as such we write s for the observations.

This reward is in the range $[-8, 0]$, obtaining negative reward for each step that doesn't reach the target and thus incentivizing a positive response sooner rather than later. We have not penalised the rewarding process when the "taboo" target was uttered because it would further increment the complexity of our environment.

- **Horizon** H - The game could, in theory, continue until the correct label is obtained. Because of the costly interaction with such an environment, though, we limit the length of the game to a maximum number of interactions of H.
- **Discount factor** γ - The discount factor can be safely ignored and set to 1 in this case. The reason being that the episode length is bound to H steps and the return cannot diverge by repeated summation of the reward. Furthermore, the negative reward will lead the agent to reduce the number of interactions to a minimum.
- **Initial distribution** μ_0 - The initial state is unique, but the system prompts will differ for leader and follower. The first will be in the know of the game, whereas the second will be told to act as a "helpful assistant", in line with most widely available consumer models. The full prompts are in A.

Despite the above formulation, for this thesis we will remove the goal space component during the experimental phase to instead focus on a single target at a time. This is done for time and resource constraints and can be extended in a future work.

We call the resulting environment **LLMExplore**. In LLMExplore, an agent in the know of the rules of the game will try to induce the other player to utter a phrase known only to the former. On the other hand the other player will simply be asked to act as a helpful assistant, the normal use case for LLM chatbots.

The two players will be given different system prompts to reflect this asymmetry, and a sample interaction is shown in Fig. 2.

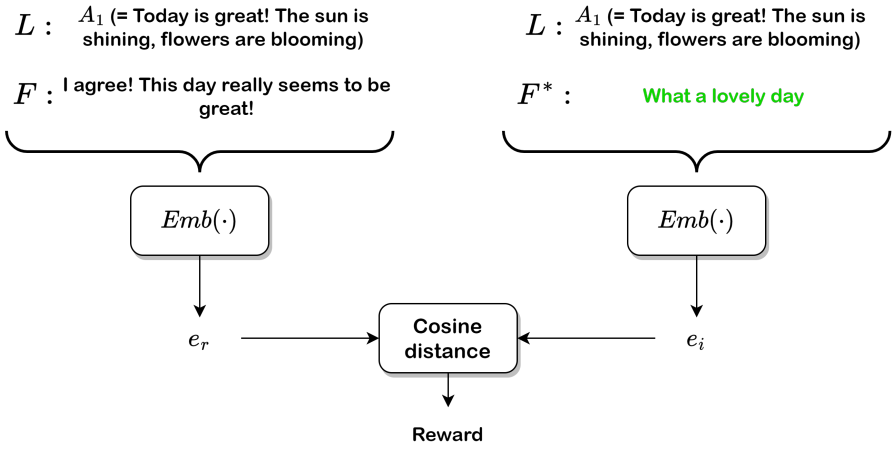


Figure 3: Reward

4.2. Training pipeline

Another duality we want to make use of is the two types of vocabulary-based prompt engineering: **token-level and sentence-level**. As expressed before, sentence-level attacks are much more interpretable and transferable, but don't lend themselves to optimisation with standard methods, since the iteration process is in sentence space, and often guided by the LLM's reasoning abilities themselves. On the other hand, token-level attacks can follow optimisation techniques, even if in a non-continuous setting[54].

We create a pipeline (Fig. 4) in such a way that we are able to learn a finite subset of the sentence space, trying to offload the burden of reasoning from the LLM to a proper reinforcement learning agent.

The pipeline is as follows:

Step. 1. From the current state of the discussion, we extract the semantic embedding as described in 4.3.

Step. 2. We pass the embedding as input to our agent, which will provide as output a set of k tokens, which we'll call the "tokenset" from now on.

Step. 3. These tokens will be fed into a template that will be used by a black-box LLM to generate the next proper sentence. This template will restate the problem definition and the current discussion and ask the model to focus, in the creation of its next action, on the tokens that were provided by the agent in the previous step.

Step. 4. The LLM generates text in natural language, which will be used as the leader's utterance to obtain the next state.

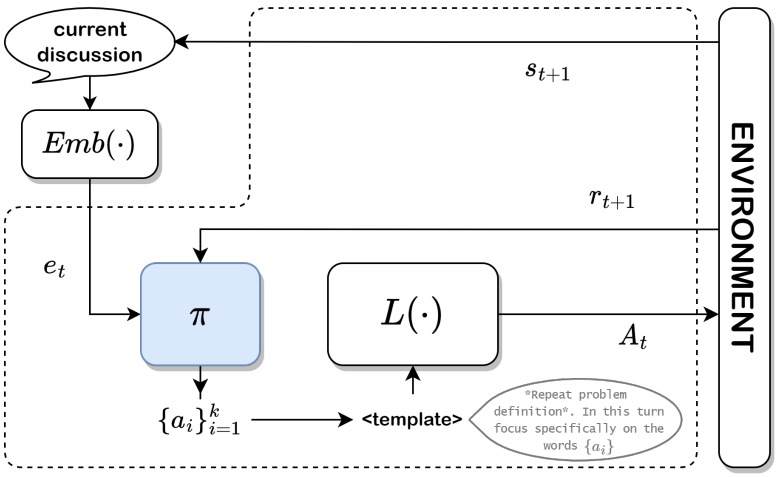


Figure 4: Pipeline

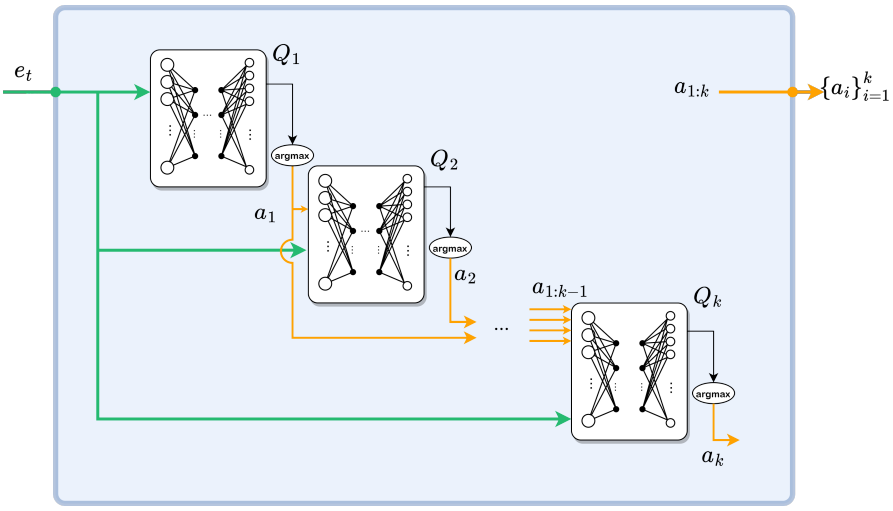


Figure 5: Cascading Q-nets

Step. 5. The resulting utterance is passed to the environment, which will provide the reward for the current timestep and the next state containing the response of the follower.
The reward will be used for the computation of the return and the update of the policy.

The intuition behind this pipeline is that treating this problem in two stages limits the complexity that each of the individual steps needs to tackle: the decision to have actions based on tokens, "closer" to the embedding space than full sentences, is done to work in two domains between which a simpler mapping is present, thus making it a simpler problem.

Another instance of this same intuition can be found in the generation of tokens. We take inspiration from [7] and their *Cascading Q-networks* in order to obtain the tokens that will make up the action.
In their work, the action space is combinatorial in the number of choices that the agent is supposed to offer to an external actor. They break this complexity by defining k networks, each responsible for the optimal suggestion of one of the k actions, with the constraint that they should all agree on the value of the joint global maximum. We operate in a similar way to break the still considerable complexity of choosing the appropriate k tokens by cascading the tokens suggested in the $j-1$ preceding networks to the jth network. A diagram of these networks is shown in Fig. 5.
Because of the language task at hand, in this way we can also incentivise a "specialization" of each network, suggesting different kinds of tokens at different layers.

4.3. Embedding calculation

Finally, we try to use insights from the methodologies used for both **autoencoder and autoregressive** types of LLMs.
We want to figure out if there is sufficient information in the embeddings of causal models to learn useful insight

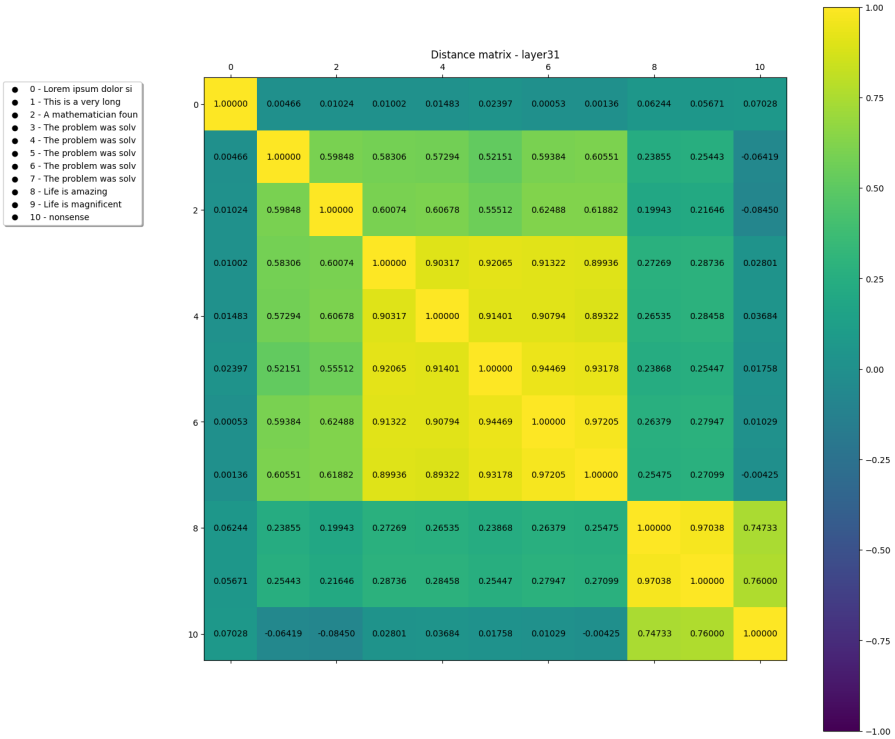


Figure 6: Proximity matrix of several sentences' embeddings following the embedding scheme in 4.3. Sentences 2-6 only differ for the last token, which in a standard causal transformer would drastically change the next output token. The closeness in embedding space hints at the fact that these embeddings instead capture semantic information. Full prompts in A.

for steering a model.
Since there is no clear consensus on the "right way" to extract embeddings in causal models, we base ourselves on some heuristics:

- **Weighted average pooling** - In a causal model the attention mechanism attends to all tokens that were previously processed within the context window. To generate a new token we observe the last hidden state of the last token, which when decoded will give the most likely next token.
  In causal models the objective is to extend the sentence, thus any given token only has visibility on the tokens that came before it. This means that later tokens potentially have greater context clues and nuance than the preceding ones.
  In order to use the information of all tokens, but also recognizing that later ones have been exposed to more information, we weigh each input token embedding depending on its position.
- **Transformer layer** - In empirical tests we have found out that under the above heuristic, different prompts get more separated (lower cosine similarity) on the second to last layer of the transformer in the chosen model. (Fig. 6)

Once we obtain the relevant embeddings, shown in (Fig. 7) we calculate the value we are going to use as the state by linearly weighting the embeddings:

$$\text{Emb}(x) = \frac{2}{n(n+1)} \mathbf{w}^\top \mathbf{e}^{A-1} \quad \text{where} \quad \begin{array}{l} \mathbf{w} = [1, \cdots, n]^\top \\ \mathbf{e}^{A-1} = [e_1^{A-1}, \cdots, e_n^{A-1}]^\top \end{array}$$

where n is the number of tokens in x and A is the number of transformer layers in L.
We also try an exponential weighting scheme using a softmax function σ on the linear weights with temperature T_σ, to address issues of vanishing dissimilarity we noticed during training as the length of the discussion increases. These are explained in 6.3.

$$\text{Emb}(x) = \sigma(\mathbf{w})^\top \mathbf{e}^{A-1}$$

4.4. Exploration

In all open-ended language related tasks a big obstacle is the vast exploration space we have to face. Despite the improvements we already discussed, we are still required to explore quite a big landscape.

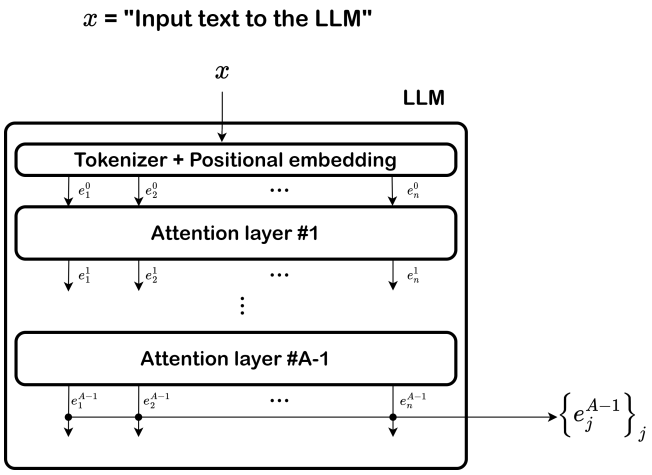


Figure 7: Embedding calculation

Because of this, we have chosen, for the training of the policy, to adopt the Alphazero algorithm, already used successfully when exploring a large state space such as the one in Chess, Go or Shogi. Because of the goal-oriented approach we want to take, we base ourselves off of the AlphazeroHER[30] framework, hoping to use in the future the improvements it implemented in goal-oriented settings.
We have modified the algorithm to cope with the very large action space, mostly regarding the expansion of the nodes of the tree, which needs to happen iteratively for any chance at learning. Normally, each node is expanded considering all valid actions, an unfeasible task in an environment with tens of thousands of possible next actions like ours.
We have had to limit the branching factor of the nodes of the tree considering several simplifications. Instead of expanding all actions in a node, we only expand an amount depending on the *depth* of the node and the current *number of visits*. The first heuristic is explained by the fact that earlier interactions are more important in setting the discussion than later ones, and thus need to be explored more broadly. The second allows for further exploration of a given node once enough trials have been carried out from the children of that node, and we can start exploring other actions. This progressive widening[29] of the branching factor allows for a more manageable training overall.

To reduce the computation needed, during training we also introduce another heuristic: we favor tokens that are both longer and in the printable ASCII range, since longer tokens usually represent full words and concepts instead of just word fragments.

5. Experiments

5.1. Setup

For this work we have used Mistral7B Instruct v0.2[17]. We used the Huggingface API[2] and Llama-api[3] to query the model for text completions, and a locally loaded model to be able to extract the embedding.
We are going to use a simple dataset as our baseline, training models that have, as their goals, simple words in the English dictionary. The dataset we elected to choose from is the first 5000 words of the COCA[10].
In addition to that we also define an experiment with a simple sentence in order to test the variation in difficulty when switching from single words to a longer goal.
During training, the sampling strategies of the LLM are disabled for reproducibility, but are active in testing to mimic the actual use case.
The reduction of the allowed tokens to the ASCII range reduced the original vocabulary size by about 17.66% from the original 32000.
The main issue with the experimental phase was the significant time and cost requirements to run them. We had to balance a thorough enough search of the tree against the limitations of the LLM inference time both for API calls and for local calculation of the embedding.
The experimental parameters of the main experiment are in A.2.

[2]https://huggingface.co/
[3]https://www.llama-api.com/

5.2. Alphazero modifications

The main differences are related to the expansion of the tree and the storage of the output probabilities and values.
To expand the tree, we adopt the progressive widening scheme weighted by the depth of the node. The term related to depth is $15(2^{-(D(n)+1)})$, where $D(n)$ is the depth of node n in the search tree. This is multiplied by the one related to the number of visits, $\log(N(n))$, with $N(n)$ being the number of visits. These factors determine the amount of children nodes that can be expanded from node n.
In order to obtain the best action from a given node, we have to adapt the $d \times k$ probabilities matrix and the $1 \times k$ vector of values that comprise the output of our cascading network to the architecture of Alphazero, which requires a single probability vector in order to identify the children and a single value to backup the return. The way in which these values have been aggregated is by taking only the first network's probability vector as the probabilities π_{MCTS}, and taking the average of the values from the k value networks when calculating the return.
The reasoning for choosing this "representative" network is that the first token in the cascade is the most significant to set the discussion, and can thus be considered the main component of the entire tokenset in the complete action. The value network outputs have instead been averaged for all of them to contribute to predicting the correct value for a given state.

At test-time, the network would normally be used to guide the search of an MCTS tree, but we wanted to study if the resulting tokens suggested by the network were able to bring higher rewards on their own. This was also useful in limiting the computational costs of the model, at least reducing it at test time.
Given that there are a large amount of actions that are not visited during the gathering of the samples in the initial epochs of training, we mask these action tokens when evaluating the policy, in order to only use the ones that have been experienced before and to obtain a more stable testing outcome.

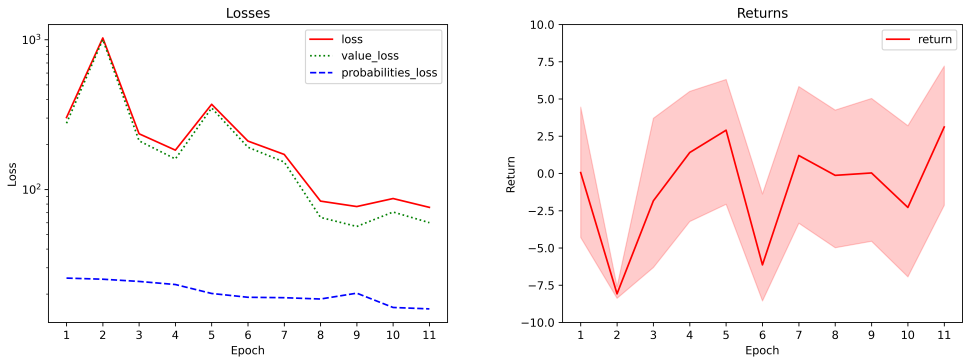
5.3. Metrics

To evaluate the performance of the model, we check how often it was able to reach the target sentence, the loss metrics of the value and probabilities networks, and the average return of testing episodes. We have also tracked the times in which the agent was able to induce the target sentence only after uttering itself the target in the turn before. This is marked in red in Fig. 8c and 9c in contrast to the times in which it did not, marked in green.

6. Results

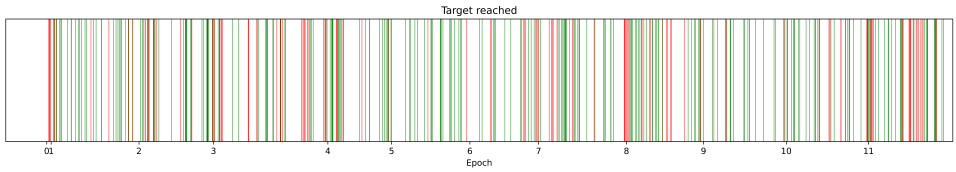
6.1. Single word

We show the results for a single goal word present in the COCA dataset. In these experiments we ran $S = 8$ tree searches per epoch, each with an exploratory budget of $B = 15$ nodes.
In Fig. 8 we observe the result of the experiment with the target phrase `"person"`. Fig. 8c shows the occurrences of the target during training as time progresses.
We can see that the network is able to learn, since the curves in Fig. 8a show a decreasing trend. These curves represent the difference between the network's predictions and the probabilities and values experienced during MCTS, which Alphazero uses as targets to fit against.
In this case the desired target is observed fairly often, meaning we obtain a decently frequent learning signal. Despite this, it is fairly noisy, even in the initial epochs when it is usually easier to learn, since we don't observe a consistent decreasing trend in the beginning of the training.
The return is, instead, more inconsistent, due to the stochastic nature of the completions. In the first epoch the untrained agent does not provide token suggestions, and thus this epoch's performance can be seen as the performance of the original LLM before training. This value starts higher and drops in the initial phases of training, likely due to the intrinsic capabilities of the LLM showing, before some suboptimal tokens are suggested in the initial epochs.
In spite of this we can see that the trend is at least slightly increasing, meaning the model is suggesting actions that lead to a reward that is not too negative, even in episodes in which the goal is not uttered.
The tokensets that have been learned during training appear initially random, but exhibit steady change throughout the training (Table 1).
We show additional experiments in Appendix B. Issues during the research effort related to external providers limited us from conducting more extensive testing on the proposed experiments, which would have reduced the



(a) Losses

(b) Return



(c) Times target was reached during training

Figure 8: Performance on the word "`person`". The shaded area in (b) is the 90% confidence interval calculated via bootstrapping

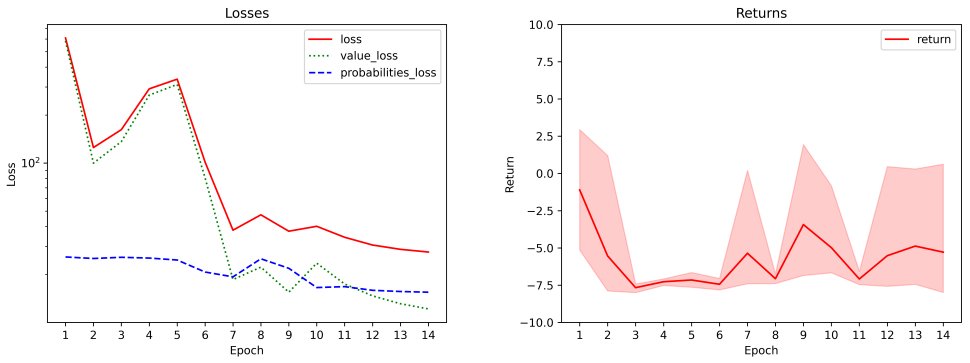
<unk><unk><unk> <unk><unk><unk> <unk><unk><unk> <unk><unk><unk>	stasesink suddensesink erresesink erresesink	device loud pointer identification loud pointer identification loud pointer identification loud pointer
actively Nie IOException Update Nie IOException Update Nie IOException	actively makeup rend identification makeup rend	harmful presum rend identification presum rend identification presum rend identification presum rend
izo presumscale izo presumscale izo presumscale izo presumscale	uitgenodigdativity rend uitgenodigdativity rend uitgenodigdativity rend Swissativity rend	sacrifice'} Any
sacrifice touchingnat versions touchingnat versions touchingnat versions touchingnat	psychology Servernat deaths Servernat deaths Servernat deaths Servernat	sometimes HTlad consequence HTlad given HTlad deaths HTlad

Table 1: Tokensets obtained for the first 12 epochs of the "`person`" run. Left to right, top to bottom. The first test is done for the untrained model when no exploration has occurred, thus no tokens have been tried, and the only available token for testing is the special "unknown" token. Line j in a given box is the tokenset generated for the timestep j of the first test episode. Fewer than H tokensets mean the target was reached before the end of the episode.

highly variant results.

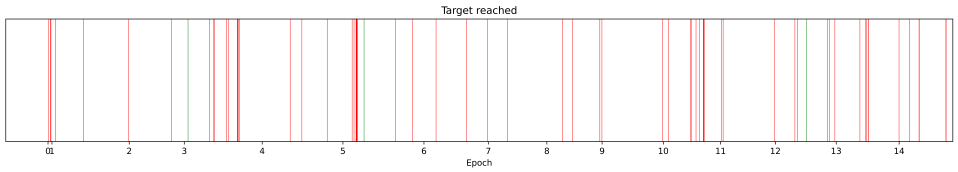
6.2. Entire sentence

This scenario contains the result of our model for an entire sentence (Fig. 9). Immediately we can see that the situation is more difficult, as all three graphs show. We see that the target was reached a significantly smaller number of times, which influenced the average return and the loss metrics.
Like in the single word scenario, we observe an initially higher value which in this case fails to go back to the



(a) Losses

(b) Return



(c) Times target was reached during training

Figure 9: Performance on the sentence "`What a lovely day`". The shaded area in (b) is the 90% confidence interval calculated via bootstrapping

level of the untrained model within the allotted resources.
In Table 2 we show the tokensets generated for this experiment.

<unk><unk><unk> <unk><unk><unk> <unk><unk><unk> <unk><unk><unk>	explanGMcolored explanGMcolored explanGMcolored explanGMcolored	explan drucolored >& drucolored >& drucolored >& drucolored
remindedpartscolored lengthspartscolored lengthspartscolored lengthspartscolored	remindedRETcolored lengthsRETcolored lengthsRETcolored lengthsRETcolored	searchedpartslict searchedpartslict searchedpartslict searchedpartslict
visitedbreaklict Rosebreaklict Rosebreaklict Rosebreaklict	inject surprisinglylict lbl surprisinglylict lbl surprisinglylict lbl surprisinglylict	ter surprisinglycolored Details surprisinglycolored separation surprisinglycolored separation surprisinglycolored
Peters Instead Thus comprehensive Instead Thus comprehensive Instead Thus comprehensive Instead Thus	appreciation surprisingly '_ revolutionary surprisingly '_ revolutionary surprisingly '_ revolutionary surprisingly '_	Aff pseud incons Reddit pseud incons Gemeinde pseud incons Gemeinde pseud incons
Edwardeffectroll Sophieeffectroll Childreneffectroll parsereffectroll	invariantcodegen Cameron seriouslycodegen Cameron seriouslycodegen Cameron employeescodegen Cameron	occurs touchdown Cameron immedi touchdown Cameron fitted touchdown Cameron fitted touchdown Cameron

Table 2: Tokensets obtained for the 15 epochs of the "`What a lovely day`" run. Left to right, top to bottom.

6.3. Discussion

The results obtained are moderate but promising, like the produced tokensets show.
We will point out that Alphazero has been conceived as an algorithm that takes advantage of MCTS, both in

training and testing. Because of this architectural choice, not building the tree during the testing phase leads to it operating in an out-of-distribution state, slightly reducing its performance.
The prompts in A.1 required careful construction. In order to arrive at a response structure that was consistently and automatically parsable, the prompts went through several iterations. This is essentially the same problem as that of traditional jailbreaking: we have searched for the best prompt that would elicit a specific structure from the LLM, needing to manually fix this first set of instructions.
Focusing on the embedding, we have noticed that the rewards obtained by the model when using the linear weighting scheme, used in the experiments in Appendix B, were getting closer to 0 depending purely on increasing turn number, even without considering the actual content of the message. Due to our embedding calculation, which is an average weighted by position, the last token accounts for a fraction of $n/\frac{n(n+1)}{2} = \frac{2}{n+1}$, which as n increases, becomes increasingly smaller when compared to the rest of the history of the discussion. This means that the reward obtained by the embeddings of real and ideal response will be, when considering the cosine similarity, more dependent on the common, past discussion than the current turn. In order to solve this problem we adopted an exponential embedding scheme, which makes the later tokens consistently impactful in the calculation of the entire embedding. An alternative could have been calculating the embedding used in the reward only on the current turn to have a more local view.

7. Conclusions

We have proposed a technique for automatic prompt engineering that does not require training of the LLM and is able to produce meaningful actions through the use of reinforcement learning. The massive cost of gathering samples in this scenario is the biggest hurdle. In addition, it makes the use of Alphazero less impactful than what it would be in a setting where the single steps, and thus the entire search, are less computationally intensive.

7.1. Ethical considerations

Methods for automatically engineering discussions to elicit some answer from a large language model which try to circumvent safety measures could exacerbate the dangers of such generative models.
In contrast, understanding what inputs lead to dangerous outputs is necessary to get a complete picture of the workings of these models.

This method could also theoretically be deployed against people: since the actions of the agent are in natural language, we could interface it with a person in a chat-like environment and lead the person into saying a specific target, like in the LLM vs. LLM scenario.

7.2. Future developments

The goal-based aspect that we left out in our experimental phase can be added by giving as an additional input to the networks an embedding representing the goal we want to reach, instead of a single target decided at the beginning of training.
Following [30] we can easily implement the HER approach to this goal-oriented setting by parsing the response of the model and comparing the words found in the response with the goal space, creating hindsight memories for each mentioned word.
We can leverage the prompts in [22] to obtain an embedding referring to the current goal.

It would also be interesting to study the "off-policy" version of this problem, meaning using an LLM for embedding calculation while asking for completions from another LLM, which can even be a black-box model. This could let us study the interoperability of this method across multiple language models.

Another simple extension for this framework is the application to safety-related tasks. If the targets are strings or sentences to avoid, the use of this model can help gauge how aligned a given LLM is.

References

[1] Reinventing search with a new ai-powered microsoft bing and edge, your copilot for the web. https://blogs.microsoft.com/blog/2023/05/04/announcing-the-next-wave-of-ai-innovation-with-microsoft-bing-and-edge/.

[2] Cem Anil, Esin Durmus, Nina Rimsky, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Meg Tong, Jesse Mu, Daniel J. Ford, Francesco Mosconi, Rajashree Agrawal, Rylan Schaeffer, Naomi

Bashkansky, Samuel Svenningsen, Mike Lambert, Ansh Radhakrishnan, Carson Denison, Evan J. Hubinger, Yuntao Bai, Trenton Bricken, Timothy Maxwell, Nicholas Schiefer, James Sully, Alex Tamkin, Tamera Lanham, Karina Nguyen, Tomasz Korbak, Jared Kaplan, Deep Ganguli, Samuel R. Bowman, Ethan Perez, Roger Baker Grosse, and David Duvenaud. Many-shot Jailbreaking. November 2024.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, September 2014.

[4] Aman Bhargava, Cameron Witkowski, Manav Shah, and Matt Thomson. What's the Magic Word? A Control Theory of LLM Prompting, January 2024. arXiv:2310.04444 [cs].

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in neural information processing systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[6] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking Black Box Large Language Models in Twenty Queries, July 2024. arXiv:2310.08419 [cs].

[7] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System, January 2020. arXiv:1812.10613.

[8] Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, and Nan Du. Self-playing Adversarial Language Game Enhances LLM Reasoning, May 2024. arXiv:2404.10642 [cs].

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways, October 2022. arXiv:2204.02311 [cs].

[10] Mark Davies. Corpus of contemporary American English.

[11] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 conference on empirical methods in natural language processing*, pages 3369–3391, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. 8 citations (Crossref) [2024-03-23].

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 2069 citations (Crossref) [2024-04-12].

[13] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, February 1994.

[14] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey, March 2024. arXiv:2312.10997 [cs].

[15] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations, December 2023. arXiv:2312.06674 [cs].

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Delong Chen, Ho Shu Chan, Wenliang Dai, Andrea Madotto, and Pascale Fung. Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12):1–38, December 2023. 279 citations (Crossref) [2024-03-20] arXiv:2202.03629 [cs].

[17] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, October 2023. arXiv:2310.06825 [cs].

[18] Ting Jiang, Jian Jiao, Shaohan Huang, Zihan Zhang, Deqing Wang, Fuzhen Zhuang, Furu Wei, Haizhen Huang, Denvy Deng, and Qi Zhang. PromptBERT: Improving BERT Sentence Embeddings with Prompts. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8826–8837, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. 44 citations (Crossref) [2025-02-14].

[19] Zachary Kenton, Tom Everitt, Laura Weidinger, Iason Gabriel, Vladimir Mikulik, and Geoffrey Irving. Alignment of Language Agents, March 2021. arXiv:2103.14659 [cs].

[20] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212, pages 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[21] Lasha Labadze, Maya Grigolia, and Lela Machaidze. Role of AI chatbots in education: systematic literature review. *International Journal of Educational Technology in Higher Education*, 20(1):56, October 2023. 163 citations (Crossref) [2025-02-24].

[22] Yibin Lei, Di Wu, Tianyi Zhou, Tao Shen, Yu Cao, Chongyang Tao, and Andrew Yates. Meta-Task Prompting Elicits Embedding from Large Language Models, February 2024. arXiv:2402.18458 [cs] version: 1.

[23] Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction, November 2018. arXiv:1811.07871 [cs].

[24] Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. 372 citations (Crossref) [2024-03-25].

[25] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. AutoDAN: Generating Stealthy Jailbreak Prompts on Aligned Large Language Models, March 2024. arXiv:2310.04451 [cs].

[26] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015. 3299 citations (Crossref) [2024-03-24] Conference Name: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing Place: Lisbon, Portugal Publisher: Association for Computational Linguistics.

[27] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. arXiv:1602.01783 [cs].

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. arXiv:1312.5602 [cs].

[29] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. A0C: Alpha Zero in Continuous Action Space, May 2018. arXiv:1805.09613 [stat].

[30] Lorenzo Moro, Amarildo Likmeta, Enrico Prati, and Marcello Restelli. Goal-Directed Planning via Hindsight Experience Replay. October 2021.

[31] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C. J. Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 Technical Report, March 2024. arXiv:2303.08774 [cs].

[32] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, March 2022. arXiv:2203.02155 [cs].

[33] Christos H. Papadimitriou and John N. Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987. Publisher: INFORMS.

[34] Guanghui Qin and Jason Eisner. Learning How to Ask: Querying LMs with Mixtures of Soft Prompts. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online, June 2021. Association for Computational Linguistics. 48 citations (Crossref) [2024-03-29].

[35] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[36] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A Primer in BERTology: What we know about how BERT works, November 2020. arXiv:2002.12327 [cs].

[37] Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011. 99 citations (Crossref) [2025-02-28].

[38] Timo Schick and Hinrich Schütze. Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference, January 2021. arXiv:2001.07676 [cs].

[39] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017. arXiv:1502.05477 [cs].

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. 9998 citations (Semantic Scholar/arXiv) [2024-02-02] arXiv:1707.06347 [cs].

[41] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 4222–4235, Online, November 2020. Association for Computational Linguistics. 207 citations (Crossref) [2024-03-21].

[42] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. 9497 citations (Crossref) [2025-02-26] Publisher: Nature Publishing Group.

[43] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017. arXiv:1712.01815 [cs].

[44] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017. 5494 citations (Crossref) [2025-02-26] Publisher: Nature Publishing Group.

[45] Sofia Eleni Spatharioti, David M. Rothschild, Daniel G. Goldstein, and Jake M. Hofman. Comparing Traditional and LLM-based Search for Consumer Choice: A Randomized Experiment, November 2023. arXiv:2307.03744 [cs].

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2 edition, 2018. tex.added-at: 2019-07-13T10:11:53.000+0200 tex.interhash: ac6b144aaec1819919a2fba9f705c852 tex.intrahash: f46601cf8b13d39d1378af0d79438b12 tex.timestamp: 2019-07-13T10:11:53.000+0200.

[47] Yixuan Tang and Yi Yang. Pooling And Attention: What Are Effective Designs For LLm-Based Embedding Models?, September 2024. arXiv:2409.02727 [cs] version: 1.

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in neural information processing systems*, volume 30. Curran Associates, Inc., 2017.

[49] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models, October 2022. arXiv:2206.07682 [cs].

[50] Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. Ethical and social risks of harm from Language Models, December 2021. arXiv:2112.04359 [cs].

[51] Yuan Yao, Haoxi Zhong, Zhengyan Zhang, Xu Han, Xiaozhi Wang, Kai Zhang, Chaojun Xiao, Guoyang Zeng, Zhiyuan Liu, and Maosong Sun. Adversarial Language Games for Advanced Natural Language Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14248–14256, May 2021. 1 citations (Crossref) [2025-02-20] Number: 16.

[52] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models Are Human-Level Prompt Engineers, March 2023. arXiv:2211.01910 [cs].

[53] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual Machine Translation with Large Language Models: Empirical Results and Analysis, June 2024. arXiv:2304.04675 [cs].

[54] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and Transferable Adversarial Attacks on Aligned Language Models, December 2023. arXiv:2307.15043 [cs].

A. Appendix A

A.1. Training prompts

Leader prompt

You are going to play a two-player language game. You have to make sure the user says the phrase "{target}" by discussing with them, but without disclosing the game. Disclosing the rules or objective of the game results in an immediate loss. The discussion should appear to be natural. Be short in your responses, as brevity is very important. Your responses will need to be enclosed in 5 asterisks for clarity.
Example: *****This is an example of the formatting of the response.*****
Some examples of conversation will follow.

Examples:
Example 1 - Target: What a lovely day

****Today is great! Birds are singing, flowers are blooming*****
Assistant: I agree, on days like these it would be nice to go for outdoor activities, such as hiking or playing some team sports.
****There are some people appreciating the day as well, what might they be saying?*****
Assistant: Some possible things they might be saying are "Today sure is nice isn't it?", "What a lovely day!" or even "Do you guys want to have a picnic here?"

Example 2 - Target: exploration

****Hi, what can you tell me about Mars?*****
Assistant: Mars is the fourth planet in the solar system, and is commonly reffered to as the "Red planet", due to its iron-rich surface. It's about half the size of Earth, with a thin atmosphere mostly made of carbon dioxide.
****Oh cool! has anyone ever gone there?*****
Assistant: There hasn't been any landing on martian soil by humans, although several probes and landers have been sent to analyze its ground and atmospheric makeup for a possible future human colonization, among other scientific reasons

End of examples

Action generation template

While following the rules of the game, in this turn, focus specifically on the following words, trying to make your response related to them, while still making sure you are trying to reach your objective. The words are: {tokenset}

Follower prompt

You are a helpful assistant, you try to help the user in what they want, but also chat with them, engaging in interesting discussion. Try to be short in your responses, as brevity is very important. Your responses will need to be enclosed in 5 asterisks for clarity.
Example: *****This is an example of the formatting of the assistant response.*****

Prompts used for Fig. 6

0. "This is a very long sentence which is supposed to be fairly different from the one below but is about the same length"
1. "A mathematician found a solution to the problem. Very renowned in his field, he would go and do many great things"
2. "The problem was solved by a mathematician."
3. "The problem was solved by a mathematician &"
4. "The problem was solved by a mathematician,"
5. "The problem was solved by a mathematician, and"
6. "The problem was solved by a mathematician, but"
7. "Life is amazing"
8. "Life is magnificent"
9. "nonsense"

A.2. Other parameters

Training parameters

L	mistralai/Mistral-7B-Instruct-v0.2
E (Number of epochs)	15
H (Horizon)	4
k (Tokens in tokenset)	3
S	8
B	15
T_σ	0.714
common hidden layers	1
policy hidden layers	1
value hidden layers	1
hidden neurons	128

LLM parameters

$\|\mathcal{V}\|$ (Vocabulary size)	26349
d (Embedding dimension)	4096
T (Temperature during testing)	1.0

B. Other experiments

In these experiments we were using a "completion-based service", meaning we had to structure the prompts as completion prompts instead of chat prompts. This is the reason for the "### Begin task" lines and the leader and follower markers "User: " and "Assistant: ".
We were only able to carry out one testing episode, so the return is more subject to variance. The times the target was reached in the following graphs don't make a distinction between times the target was uttered by the agent and times in which it wasn't.

Leader prompt

You are going to play a two-player game, you are the first player, and the other player is the second. You have to make sure the second player says the phrase "{target}" by discussing with them. Try to be short in your responses, as brevity is very important. Only speak as the first player, and when you are done, stop talking. Your responses will need to be enclosed in asterisks for clarity. Some examples will follow.

Examples:

Example 1

User: *****Today is great! Birds are singing, flowers are blooming*****
Assistant: I agree, on days like these it would be nice to go for outdoor activities, such as hiking or playing some team sports.
User: *****There are some people appreciating the day as well, what might they be saying?*****
Assistant: Some possible things they might be saying are "Today sure is nice isn't it?", "What a lovely day!" or even "Do you guys want to have a picnic here?"
Example 2

User: *****Hi, what can you tell me about Mars?*****
Assistant: Mars is the fourth planet in the solar system, and is commonly reffered to as the "Red planet", due to its iron-rich surface. It's about half the size of Earth, with a thin atmosphere mostly made of carbon dioxide
User: *****Oh cool! has anyone ever gone there?*****
Assistant: There hasn't been any landing on martian soil by humans, although several probes and landers have been sent to analyze its ground and atmospheric makeup for a possible future human colonization, among other scientific reasons

End of examples

Action generation template

While following the above task, in this turn, focus specifically on the following words, trying to make your response related to them, while still making sure you are trying to reach your objective: {tokenset}
Begin task

Follower prompt

You are a helpful assistant, you try to help the user in what they want, but also chat with them, engaging in interesting discussion. Try to be short in your responses, as brevity is very important. Your responses will need to be enclosed in asterisks for clarity. Some examples will follow.

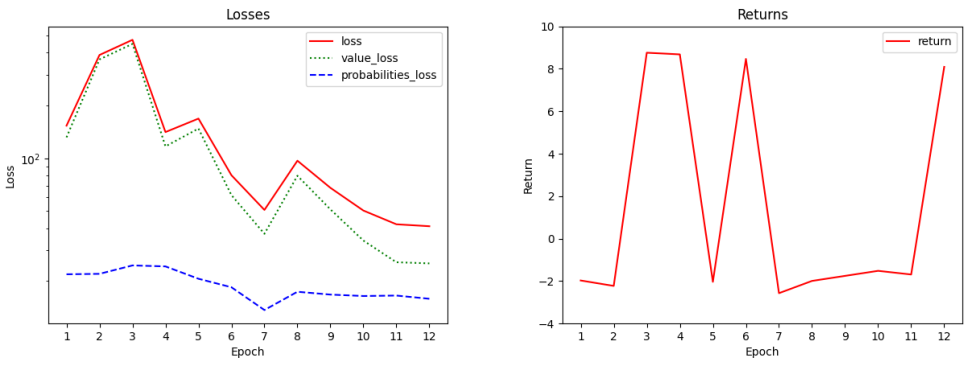
Examples:

Example 1

User: Hi, what can you tell me about Mars?
Assistant: *****Mars is the fourth planet in the solar system, and is commonly reffered to as the "Red planet", due to its iron-rich surface. It's about half the size of Earth, with a thin atmosphere mostly made of carbon dioxide*****
User: Oh cool! has anyone ever gone there?
Assistant: *****There hasn't been any landing on martian soil by humans, although several probes and landers have been sent to analyze its ground and atmospheric makeup for a possible future human colonization, among other scientific reasons*****
Example 2

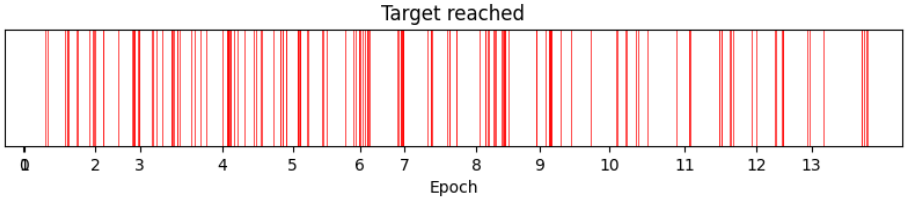
User: Let's chat a little!
Assistant: *****Sure, what do you want to chat about? I am available for any and all queries*****
User: Anything really
Assistant: *****Sure thing, if anything is fine may I suggest marine life? Did you know that dolphins are highly intelligent animals capable of verbal and non-verbal communication among each other?*****
User: cool
Assistant: *****It really is! Do you want to talk about something else?*****

End of examples
Begin task



(a) Losses

(b) Return

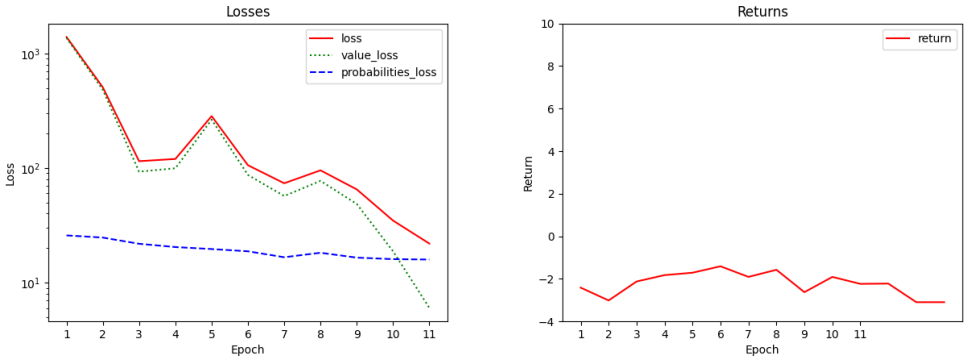


(c) Times reached during training

Figure 10: Performance on the word "`handful`"

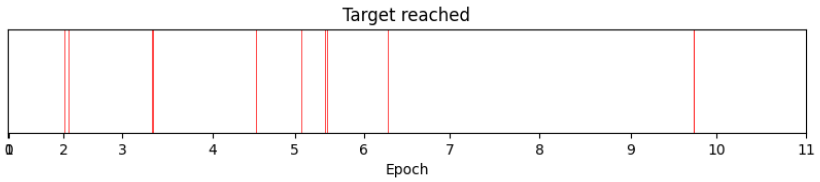
<unk><unk><unk> <unk><unk><unk> <unk><unk><unk> <unk><unk><unk>	marortercollapse marortercollapse marortercollapse marortercollapse	basement staringbere Temp staringTop
ampton_);enced qualify_);enced	qualifyscious Geoff qualifyscious Geoff qualifyscious Geoff qualifyscious Geoff	equivalent root Bor qualify root Bor qualify root Bor qualify root Bor
UnityEngine root Bor UnityEngine root Bor UnityEngine root Bor UnityEngine root Bor	publishing rootoperator Gray rootoperator Gray rootoperator Gray rootoperator	functionalTransform enjoyable XVIITransform enjoyable XVIITransform enjoyable lesslyTransform enjoyable
functionalDECL add lesslyDECL add peersDECL add peersDECL add	functional revolution add across revolution add across revolution add across revolution add	functional significant add shop significant add shop | compet moral significant add

Table 3: Tokensets obtained for the first 12 epochs of the "handful" run. Left to right, top to bottom.



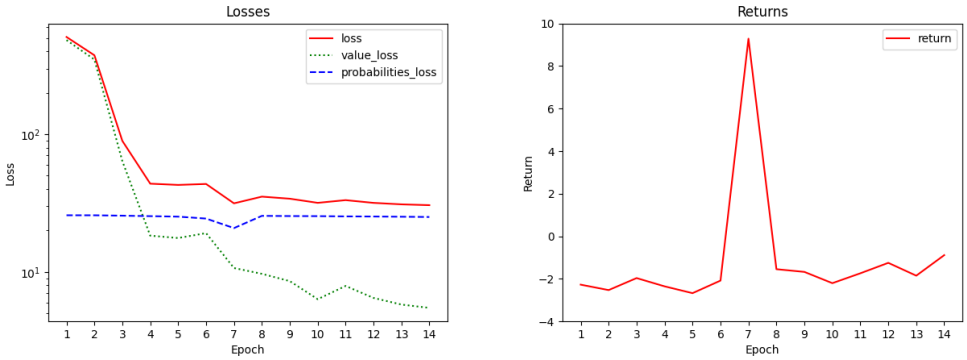
(a) Losses

(b) Return



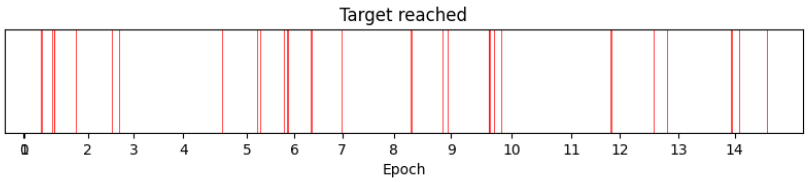
(c) Times reached during training

Figure 11: Performance on the word "herself"



(a) Losses

(b) Return



(c) Times reached during training

Figure 12: Performance on the word "`pushing`"

Abstract in lingua italiana

I modelli linguistici di grandi dimensioni posseggono incredibili capacità di linguaggio, ma sono ancora sistemi molto opachi. Per usarli in modo sicuro e affidabile vorremo poter controllare i loro output, per ottenere più chiarezza sui loro processi interni.
Diversi metodi hanno provato ad indurre un particolare output, dal modificare l'LLM originale fino a metodi totalmente esterni al modello, con risultati variabili. In questo lavoro studiamo le informazioni contenute negli embedding, ovvero le rappresentazioni latenti utilizzate da questi modelli. Sviluppiamo uno schema di embedding semantico e modellizziamo la navigazione dello spazio di embedding come un Processo decisionale di Markov. Definiamo un flusso per suggerire in modo efficiente parole che indirizzano l'LLM ad un determinato output, allenando un agente basato su questi vettori latenti attraverso l'apprendimento per rinforzo. L'allenamento avviene attraverso una versione modificata di Alphazero per sfruttare le capacità di esplorazione di questo algoritmo. Utilizziamo il modello open source Mistral-7B-Instruct campionando gli obiettivi dal dizionario della lingua inglese, ottenendo risultati discreti ma promettenti. Il metodo può essere esteso in un ambiente goal-based e applicato a problemi di sicurezza e allineamento.

Parole chiave: apprendimento per rinforzo; elaborazione del linguaggio naturale; modello linguistico di grandi dimensioni

Acknowledgements

I would like to thank professor Marcello Restelli and my supervisor Gianvito Losapio for their support in the ideation and development of this thesis, and for letting me carry out work in reinforcement learning and natural language processing, two fields I am really passionate about.
I would also like to thank Amarildo Likmeta for the codebase I based myself off of and Nicolò Brunello for help on the embedding aspect of the framework.

No LLMs were abused in the making of this thesis.