

# Langue de Chat

January 2023

## 1 Introduction

We introduce Langue de Chat: an AI-based chat-bot that helps users retrieve scientific documents. Langue de Chat is entirely based on a reinforcement learning (RL) agent, trained on a rule-based user simulator. The choice of a reinforcement learning approach over supervised learning or rule-based methods exempts from the need of expert generated examples: policies are learned automatically from experience, allowing the conversational agent to explore trajectories that possibly do not exist in previously observed data, which increases the agent’s autonomy.

This project takes inspiration from the works of researchers at MIULab (Machine Intelligence & Understanding Laboratory National Taiwan University) [1, 2], which developed a similar architecture for a goal-oriented chat-bot for movie ticket booking and movie seeking. We claim to have improved from many points of view these works. Moreover, we applied these borrowed tools to a much more real case scenario.

## 2 Methodologies

**Overview.** Figure 1 represents the project architecture. The following sections analyse each component of the training cycle, to which we restricted our interest.

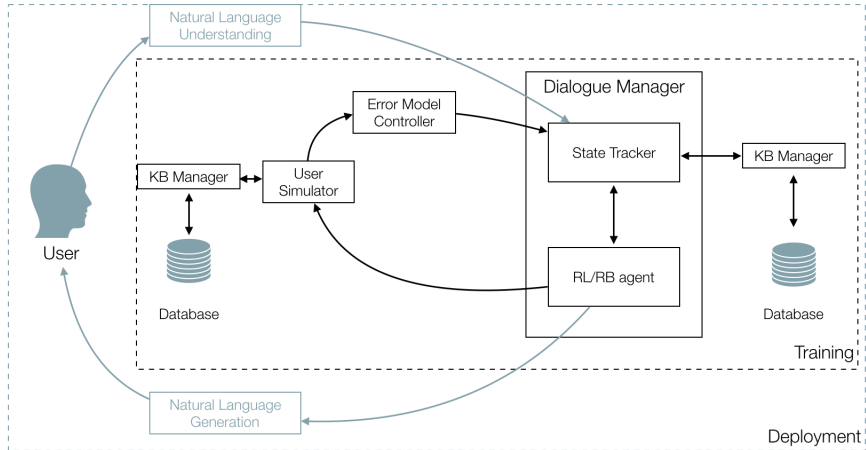


Figure 1: Overview

**Knowledge Base Manager.** The KBM role is to supply documents from the knowledge base to match the constraints set by the current state of the conversation. It is also used to count the matching items for each constraint, which is needed for the state of the RL Agent. The KB Manager can be queried also by the US, after conversation ends, to check if the goal proposed by the agent effectively matches its needs.

**User Simulator.** The US is a goal-oriented rule-based agent. Its goal is selected randomly from a user goal database at the beginning of each conversation and consists of a set of **informs**, which will serve as constraints for the agent, and a set of **requests** that the agent will be questioned about. The US keeps track of the history of the conversation in its internal state variables, which include the proposals made by the agent, its pending requests and the unexpressed constraints. At each step, the US rule-based policy receives in input the last agent action and outputs the next user action, taking into account the internal state. The US action is either a set of informs and requests, a thanking or a rejection of the agent proposals.

A reward is output after each step of the conversation, which ends when the US is satisfied (i.e. it has no more pending requests nor unexpressed constraints), or if the conversation reaches a maximum number of rounds (patience of the user), as shown in Figure 2.

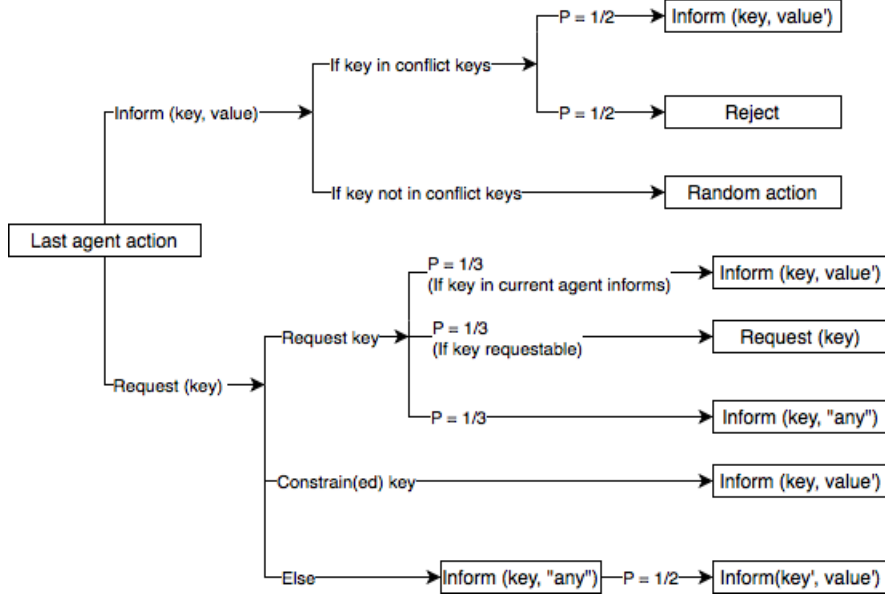


Figure 2: User Simulator rule based policy

**Error Model Controller.** The EMC is used to corrupt the US action, simulating errors in NLU, which may take place when interacting with real users. This is done to train the RL agent to detect and correct these errors. The error is infused with a given probability, and may be a switch in the keys requested or informed by the user as well as the change of an informed value.

**Dialogue Manager.** The dialogue manager is made of the state tracker and the agent. At each step, the DM receives in input the last user action, and outputs the agent action. When using the RL agent, this requires getting the current state from the ST first. The agent training procedure is based on the experience stored during simulated conversations. The experience replay buffer pool is initialized using a Rule-Based agent (RB agent), before training starts, in the so-called warm-up period. This is done to suggest intelligent actions to the RL agent, thus speeding up its training.

**State Tracker.** The state tracker provides an interface between the user and the agent. Its goal is to keep track of the state of the conversation: this includes the current user informs and requests, the current agent informs, the last agent and user actions, and the count of matching documents in the KB given the current constraints. These information are one-hot encoded and fed to the RL agent Q-network. An additional role of the ST is to complete the RL agent informs with matching values from the KB.

**RL Agent.** The RL agent is a Deep Q-Network, trained using experience replay. Given an input state, the network outputs a Q-value, namely a score quantifying how convenient it is to perform a certain action in that precise state. The action with the highest Q-value is chosen as the next agent's action with probability  $1 - \epsilon$ , otherwise a random action is picked by the agent.  $\epsilon$  is an hyper-parameter used to modulate the RL agent propensity to exploration, and usually exponentially decreases with the ongoing of the training.

**RB Agent.** The RB agent is used to initialize the replay buffer pool. The RB agent keeps track of the conversation history in its internal state variables, including the current user informs and requests, and the agent current proposals. At each round of the conversation, the agent is fed with the user action and outputs its own action based on a rule-based policy, as shown in Figure 3.

### 3 Results

We tested our architecture using a list of 9194 documents sampled from IEEE Xplore database, generated by querying the IEEE API using the most popular AI topics as keywords. The Q-network of the RL agent was an MLP with 80 hidden neurons, with ReLU as the activation function of the hidden layer. We used Adam optimizer with learning rate  $\eta = 1e - 3$  and MSE loss function, and set the RL discount factor to  $\gamma = 0.9$ . The exploration hyper-parameter was set to  $\epsilon = 0.1$ , with exponential decay  $d = 0.9$ . We used

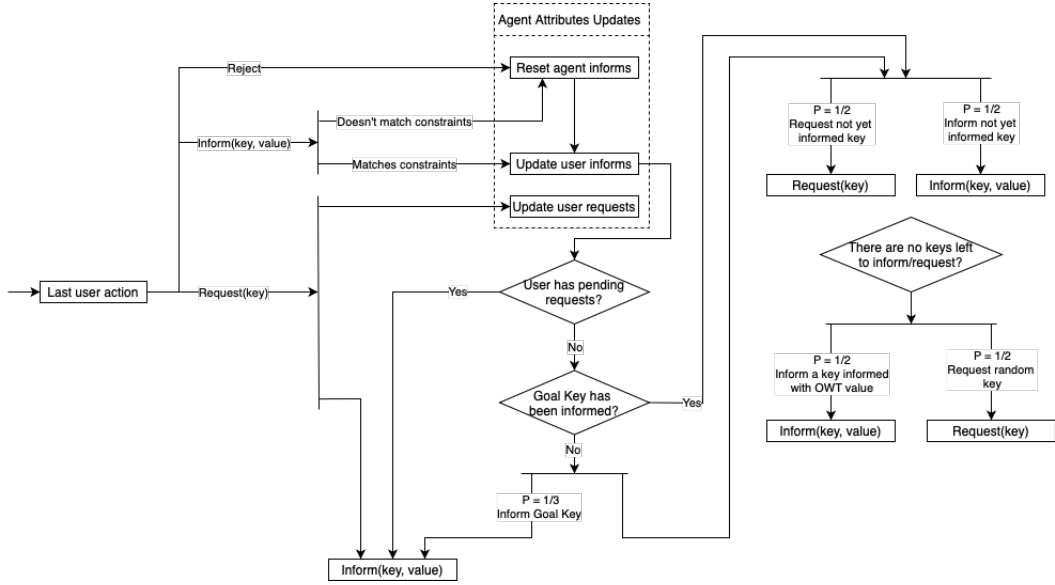


Figure 3: Rule Based Agent policy

random batches of size 256 for experience replay. The model was warmed up until collecting  $2^{13}$  experience tuples, and then trained for 50 epochs, each consisting of 100 episodes. The EMC error probability was set to 0.02 both for warm-up and training, and the US patience to 15. Figures 4 and 5 show the results that we got in terms of average reward and success rate. As we can see, after only  $50 \times 100 = 5000$  episode, the RL agent outperforms the RB agent, whose results are already quite satisfactory.

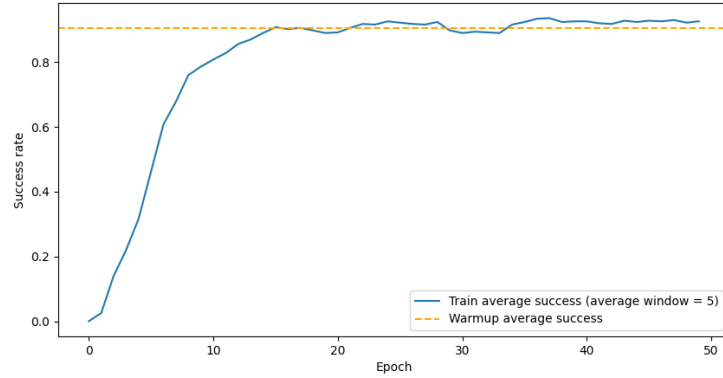


Figure 4: Training average success rate

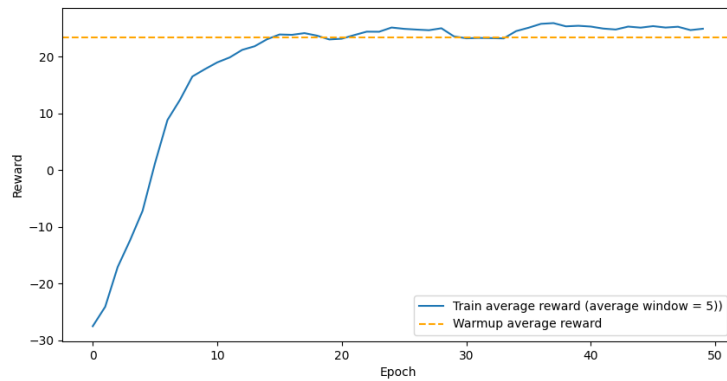


Figure 5: Training average reward

## A A framework from the Artificial Intelligence Course

For this project, we developed three different agents, two of which are rule based, while the other is a learning agent, following the guidelines of the Artificial Intelligence Fundamentals course.

The environment type largely determines the agent design: in this case, not only the environment is multi-agent, but we could almost argue that the interaction of an agent with the environment boils down to the interaction with the other agent that populates the environment. Every rational agent has the power of acting and perceiving: here, this means nothing but the ability of expressing and processing actions during the conversation, resulting in an effective exchange of information.

The two agents are cooperative, in the sense that a conversation can only end in win-win or lose-lose situations. Despite this, their goals are defined independently: the user goal consists of a set of actions that the US must take during the conversation, while the RL agent aims to maximize its reward, namely the performance measure driving the learning process.

At each round of the conversation, each agent possibly has access to a new piece of information: the RL agent can discover something about the user goal and the user simulator can get some information about the current agent’s proposal. In this sense, we can state that each agent operates in a partially observable environment. Now, the best way to handle partial observability is for a rational agent to keep track of all previously observed information, and this is why both the US and the RL agent continuously update their internal state with the history of the conversation.

In the first set of simulated dialogues, the RL agent is replaced by a rule based agent. In this phase, the conversation is the result of deterministic and stochastic rules decided by the programmers. Even though such agent can be programmed to be possibly unsurpassed on a precise task, it remains very fragile when it comes to deal with new situations. To sum up, the need for a learning agent finds an explanation in the willingness to develop a flexible and autonomous conversational agent, while the rule based agent remains fundamental for speeding up the first phase of the learning process.

## B Improvements over the original papers

Our work is highly inspired to those from Li et al. [1] and Li et al. [2]. The architectural schema is in fact the same; still, we included in our implementation several improvements.

First of all, we redefined the state of the agent. In the original paper this included both an absolute and one-hot representation of the round of the conversation, given the patience of the user. We believe that this has to be changed: the ST does not know a priori the patience of the user, and thus should not be able to represent the current round in a one-hot fashion. Also, this set a maximum number of rounds to the conversation, which is not necessarily appropriate when dealing with users with different patience. In the original paper, the round representation is needed because each conversation ends as a consequence of a **done** action, by the user or the agent. Thus, the round representation allows the agent to choose a done action when the conversation is about ending. Still, we think that this is not appropriate either, as the chat should end only when the user wants to end it, either because it is satisfied, or because its patience has exhausted. Thus, in our implementation, we dropped both the **done** action and the state round representation, thus allowing Languge de Chat to deal with not known and variable patience users. We also included in the state the current user requests, which were not encoded in the original papers. We believe that this is a valuable information for the agent to choose its best action.

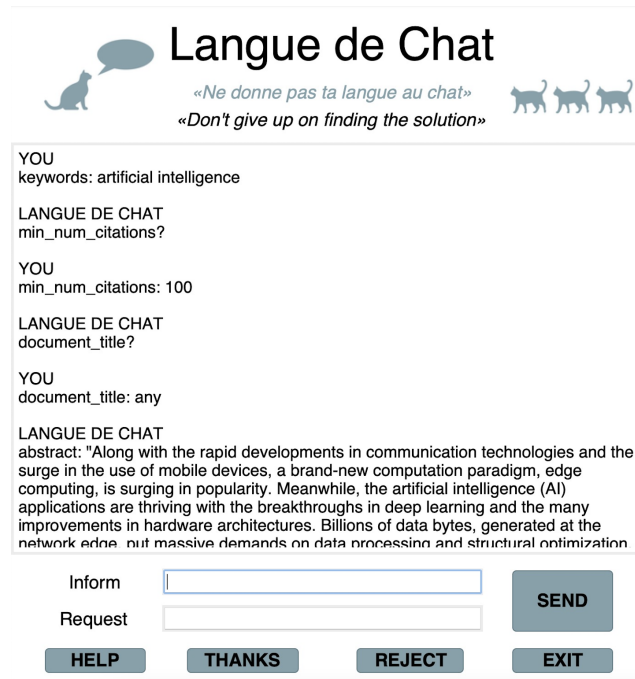
We also introduced an intelligent Rule Based agent for warming up the RL agent. This was not done in the original papers, where the warm up was left to a much more "stupid" rule based agent. Even if this is not crucial, having much better experience early can drastically speed up the training.

We also introduced the agent action **WITHDRAW CURRENT PROPOSALS**. This action is undertaken when the user informs something for which the agent cannot find a matching document. This can happen when the agent proposals "compromises" the agent. In this case, if the agent had previously proposed something, it withdraws it, and looks for new documents matching the updated user constraints. In the original papers, the agent would instead undertake a **NO MATCH AVAILABLE** action, which would cause a **REJECT** user action, causing the reset of the current agent proposals. Our solution let the agent save a round of the conversation, and we believe is much more natural and user friendly.

Many other minor improvements were made, including the possibility of define and deal with constraints between the keys involved in the dialogue (e.g. a minimum publication year or a minimum number of

citations), which was not expected in the original papers, as well as an enlargement of the EMC infusion error methods.

## C Open source code and Graphical User Interface



**Langue de Chat**  
 «Ne donne pas ta langue au chat»  
 «Don't give up on finding the solution»

YOU  
 keywords: artificial intelligence

LANGUE DE CHAT  
 min\_num\_citations?

YOU  
 min\_num\_citations: 100

LANGUE DE CHAT  
 document\_title?

YOU  
 document\_title: any

LANGUE DE CHAT  
 abstract: "Along with the rapid developments in communication technologies and the surge in the use of mobile devices, a brand-new computation paradigm, edge computing, is surging in popularity. Meanwhile, the artificial intelligence (AI) applications are thriving with the breakthroughs in deep learning and the many improvements in hardware architectures. Billions of data bytes, generated at the network edge, put massive demands on data processing and structural optimization

Inform

Request

**SEND**

**HELP** **THANKS** **REJECT** **EXIT**

Figure 6: Langue de Chat GUI

The code for this project is fully open source and available on GitHub. Apart from the scripts used to build the environment and train the RL, you can find an interactive GUI, shown in figure 6. You can run it with the command `python app.py` to test Langue de Chat yourself. Natural language units have not been implemented, so the dialogue is constrained to dictionary-based communication. We warmly suggest to check the **HELP** menu before starting!

### Team Contributions

Gabriele Marino: Dialogue Manager, with particular focus on the reinforcement learning agent and the definition and management of the environment through the State Tracker.

Andrea Roncoli: Warm up policy for RL Agent with implementation of Rule Based Agent, generation and management of knowledge base through Knowledge Base Manager.

Bianca Ziliotto: Implementation of the User Simulator, a goal oriented rule-based agent able to support the training of the RL Agent, and of the Error Model Controller, with the prospect of future possible implementations of NLU/NLG.

## References

- [1] Xuijun Li et al. "A User Simulator for Task-Completion Dialogues". In: *arXiv preprint arXiv:1612.05688* (2016).
- [2] Xuijun Li et al. "End-to-End Task-Completion Neural Dialogue Systems". In: *Proceedings of The 8th International Joint Conference on Natural Language Processing*. 2017.