NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ALEXANDRE TEIXEIRA LOBATO CORREIA

BSc in Computer Science

CONVERSATIONAL INTERACTIVE RECOMMENDATION

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CONVERSATIONAL INTERACTIVE RECOMMENDATION

ALEXANDRE TEIXEIRA LOBATO CORREIA

BSc in Computer Science

Adviser: João Miguel da Costa Magalhães
Associate Professor with Habilitation, NOVA School of Science and Technology

Co-adviser: David Fernandes Semedo
Assistant Professor, NOVA School of Science and Technology

Examination Committee

Chair: Carla Maria Gonçalves Ferreira
Associate Professor, NOVA School of Science and Technology

Adviser: João Miguel da Costa Magalhães
Associate Professor with Habilitation, NOVA School of Science and Technology

Member: Carlos Santiago
Investigator, Instituto Superior Técnico

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2022

Conversational Interactive Recommendation

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.2) [22].

Acknowledgements

I want to thank:

My advisor, Professor João Magalhães, and my co-advisor, David Semedo, for all their support, expertise and motivation;

My colleagues at NOVASearch investigation group, specially to Diogo Silva, for all their help and brainstorming sessions;

NOVA University Lisbon, specially to NOVA School of Science and Technology faculty, not only for these amazing five years of continuous and great learning, but also to the friends I made through the entire process, which also gave a huge help during some difficult moments across these five years;

Farfetch for taking a partnership with NOVA School of Science and Technology and letting me work on the iFetch project (ref. LISBOA-01-0247-FEDER-045920);

And most important, my whole family for all the support, not only during my thesis development or during the course, but during my entire life.

Abstract

When we search for something online, sometimes we know what we want, but we do not know how to exactly describe it. To address this challenge, this thesis presents a framework based on reinforcement learning, which improves conversational assistants in order to get closer to the user's desired product by asking a sequence of clarifying questions and giving recommendations. These clarifying questions are chosen before each interaction, in real-time, and are capable of narrowing down the search space, helping the user finding what he wants.

In order for the assistant to decide which clarifying question to use, its neural network needs some information about the recommendation conversation. After the selection of the question, the assistant generates the question using a template, which might need some recommended products. Having generated the question, the assistant presents it to the user and gets his answer, from which extracts some information to update the recommendation conversation history.

This framework can be served as a module inside projects and lets the recommendation conversation to evolve in various directions: minimizing the number of interactions, minimizing the repetitiveness of used questions and minimizing the repetitiveness of used questions according to their feedback times.

Keywords: Conversational Assistant, Conversational Agent, Task-Oriented Conversational Agent, Risk-aware Conversational Agent, Reinforcement Learning, Deep Reinforcement Learning

Resumo

Quando pesquisamos algo online, às vezes sabemos o que queremos, mas não sabemos como descrevê-lo exatamente. Para enfrentar este desafio, esta tese introduz uma framework baseada em aprendizagem por reforço, que melhora os assistentes de conversação para se aproximarem do produto desejado pelo utilizador, através de uma sequência de perguntas esclarecedoras e dando recomendações. Estas perguntas esclarecedoras são escolhidas antes de cada interação, em tempo real, e são capazes de diminuir o espaço de procura, ajudando o utilizador a encontrar o que deseja.

Para que o assistente decida qual pergunta esclarecedora usar, a sua rede neural precisa de algumas informações sobre a conversa de recomendação. Após a seleção da pergunta, o assistente gera a pergunta usando um template, que pode precisar de alguns produtos para recomendar. Gerada a pergunta, o assistente apresenta-a ao utilizador e obtém a resposta, da qual extrai algumas informações para atualizar o histórico da conversa de recomendação.

Esta framework pode ser usada como um módulo em projetos e permite que a conversa de recomendação evolua de várias formas: minimizando o número de interações, minimizando a repetitividade das perguntas usadas e minimizando a repetitividade das perguntas usadas de acordo com os tempos de feedback.

Palavras-chave: Assistentes Conversacionais, Agentes Conversacionais, Agente Conversacional Orientado a Tarefas, Agente Conversacional com Noção do Risco, Aprendizagem por Reforço, Aprendizagem por Reforço Profundo

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Context and Motivation

Conversational agents are an emerging technology and are slowly replacing traditional search engines [11, 41]. This is essentially because of their capabilities to communicate in natural language, providing a more natural way of interacting with a machine as shown in Figure 1.1, and their ability to fine-tune the search results by maintaining the context within turns of a conversation.

When a user searches for something on search engines or online shopping, he sometimes knows what he wants or may encounter something interesting during the search/shopping process. To simplify the development scenario, like the authors of [24, 25] did, this thesis focuses on the scenario where the user knows what he is looking for or wishes to find. This scenario restriction still comes with some challenges regarding the way the user is able to describe the target product for these systems to accurately find it, as can be observed in Figure 1.1, mainly because:

- Although the user knows what he wants, his description of the product (i.e. his input) might be ambiguous or too generic, since:
  - He might not know which attributes the system uses to describe products;
  - He might not know what values each attribute is associated with;
  - He might not know the slight differences between similar existing products;
- The system might have a large amount of products to search through.

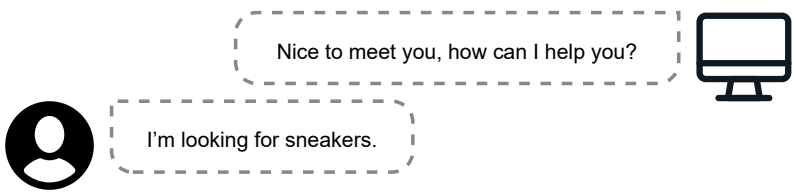


Figure 1.1: Conversation Sample

1.2 Proposed Framework

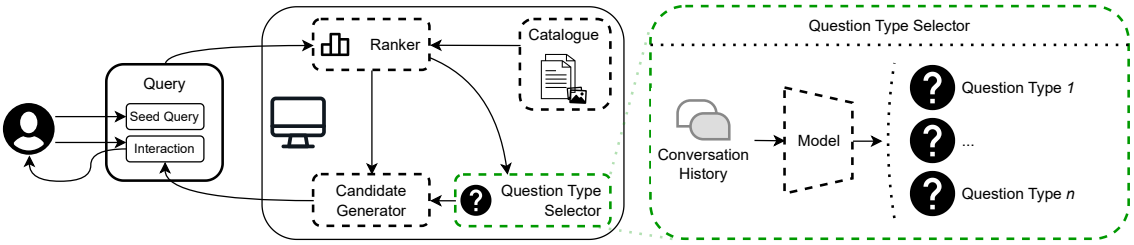


Figure 1.2: Framework Architecture

This thesis focuses on developing a framework that improves conversational assistants in order to get closer to the user's target product by generating a sequence of clarifying questions and giving recommendations. This way, the assistant has the capability of deciding which question type is most relevant to narrow down the search space towards the user's target product, which in the case of this thesis can be one of the three available question types:

- *Binary* attribute question
- *Relative* attribute question
- *Comparative* attribute question

Additionally, this framework allows the assistant to be driven in different ways in order to achieve different behaviors according to the desired requirements, which for the use case of this thesis there are three behaviours defined:

- Conversations with *few interactions*;
- Conversations with *few question types repetitiveness*;
- Conversations with *few question types repetitiveness according to the time the user takes to answer each of them*.

To support such feature, as shown in Figure 1.2, the model of the Question Type Selector will decide which question type to select, conditioning the information expected to be gathered from the user's answer and may be accompanied with one or more product recommendations, in order to generate the clarifying question.

Having the clarifying question generated, the system presents it to the user and gets his answer. This interaction between the assistant and the user is simulated, being performed by comparing the specified attribute value(s) with the one of the product the user wishes to find (represented as a random product from the catalogue).

After the feedback of the user, the system computes the top results according to the restrictions provided by the user and produces a reward to indicate to the agent if he is

coming closer to the user's desired product or not. This reward can be shaped in various forms to give the assistant the ability to have the multiple behaviours defined above.

Leveraging reinforcement learning, which is incorporated in the Question Type Selector of Figure 1.2, the assistant learns how to minimize the number of questions while still making meaningful questions.

1.3 Research Challenges and Objective

With the goal of helping the user finding his desired product as soon as possible by making a sequence of questions, this thesis focuses on the development of the Question Type Selector, which will require the development of two main components:

Model to decide which question type to use in the next interaction of the conversation with the user. This is based on deep learning, trained using reinforcement learning, and chooses the next question type to use by mapping the conversation history with the available question types as shown in green on Figure 1.2, i.e. classifies the conversation history into one of the available question types;

Ranker to perform the ordering of the products according to the user preferences, which is used both for the products accompanying the clarifying questions and the list of recommended products.

The hypothesis considered to address the goal is that by leveraging on reinforcement learning, it is possible to replace heuristic-based models with a more principled, smart and scalable model, that maximizes a given reward function. That reward function is designed to account for the number of interactions taken and the time each question type takes to have feedback from the user. This introduces the following challenges:

State Composition how the state of the Question Type Selector should be composed for the system to make good decisions;

Reward Function the schema of the reward function for the assistant to get feedback on its decisions according to the expected behaviour;

Architecture the architecture of the neural network used in the Question Type Selector;

Actions which of the existing methods are useful to get feedback from the user - the question types;

Dataset the training process requires a dataset including both the interactions between the user and the assistant, and a catalogue for the assistant to search through.

1.4 Contributions

The result of the work performed in this thesis contributes to:

- **iFetch project, a multimodal conversational agent for the online fashion marketplace**: with the goal of having a virtual assistant that can mimic the Farfetch's in-store experience, being the assistant able to offer a more natural and personalized experience than the traditional conversational agents, Farfetch is developing its own virtual assistant.

  With this project several challenges arise as depicted in Figure 1.3: a simple and intuitive interface for the user to interact with the assistant (3D Virtual Store), understanding the user input in natural language (State-Tracker and its pre-processing) and either generation of text using natural language (Decoder) or a list of recommended products (Recommendation). When recommending products, if the user already has a clear idea of what he wishes to find, there might be the need for the assistant to ask clarifying questions, each of which might be accompanied by one or more recommended products.

  The work of this thesis focuses on efficiently retrieving the user's desired product by asking a minimal sequence of questions, thus implementing the clarifying questions of the Recommendation module.

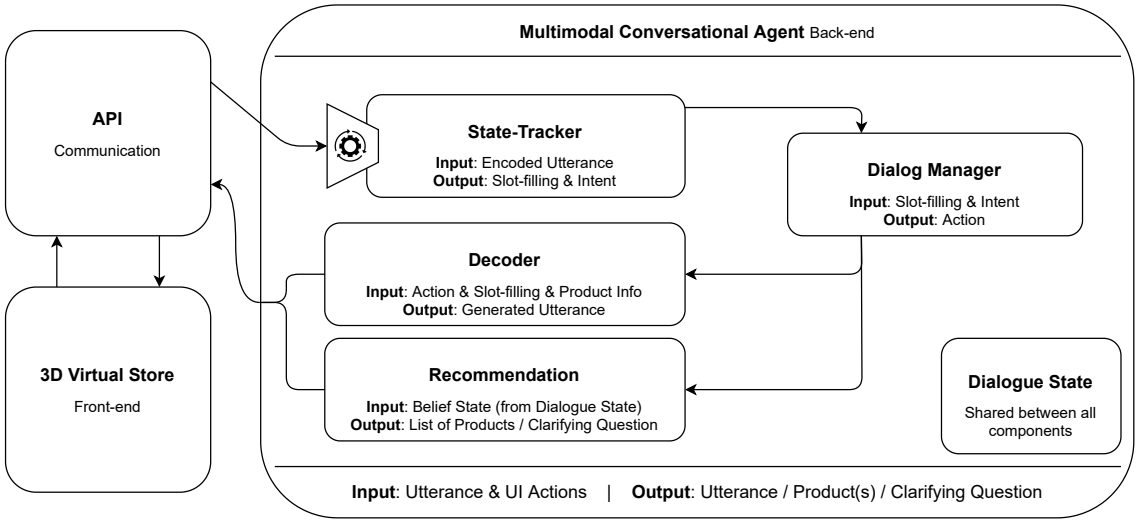


Figure 1.3: iFetch Prototype Architecture

- **Target-Product Oriented Conversational Agents**: by enabling the agents to ask sequences of clarifying questions conditioned by various behaviours according to different requirements;

- **System for Interactions Simulation**: whenever the dataset does not include the interactions between the user and the assistant, the system developed is capable of simulating those interactions and extract the needed information for the agent.

1.5 Document Structure

This document has the following structure:

- **Chapter 1 (Introduction)**: introduces the context and motivation for this thesis, and explains the architecture and the goal of the developed framework;
- **Chapter 2 (Related Work)**: presents the related and state of the art methods and technologies, as well as the required background;
- **Chapter 3 (Conversational Interactive Recommendation)**: describes how the interaction between the assistant and the user is performed, focusing on the available question types;
- **Chapter 4 (Selection of Clarifying Questions)**: specifies how the assistant learns and selects the next question to ask, and how the next attribute is selected for usage in each clarifying question;
- **Chapter 5 (Experiments and Results)**: presents and analyses all the experiments performed and discusses the problems faced during the development of this thesis;
- **Chapter 6 (Conclusions and Future Work)**: concludes and reviews the works of this thesis and discusses ways of improving and extending it.

2

Related Work

This chapter presents the related and state of the art methods and technologies which were used in the development of this thesis as well as the necessary theoretical background.

2.1 Task-Oriented Conversational Agents

Conversational agents or dialogue systems are designed to interact with a human via multiple interaction channels like text or speech, which are all translated into a representation that a machine can understand and process, as shown in Figure 2.1 [10].

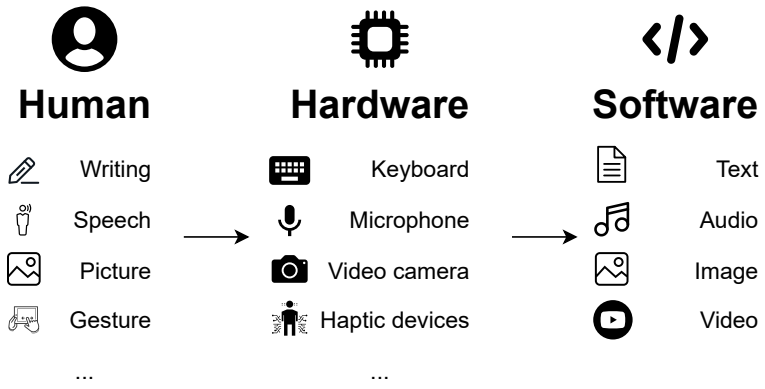


Figure 2.1: Conversational Agent Pipeline, adapted from [10]

When combined with recommendation systems - with the goal of predicting the rating the user would give to an item - it originates a conversational recommendation system, which aims to understand the user's needs through dialogue in order to rate the items of the system according to the user preferences, so that it can suggest relevant items to him.

Since these systems seek to help the user in getting its desired product, this type of conversational agents are called task-oriented conversational agents, and can support two different types of domain constraints [5]:

- **Open-Domain**: supports tasks on a variety of domain areas;
  Examples: Google Assistant, Siri and Alexa

- **Domain-Specific**: focused on tasks on a specific domain.
  Examples: recipes, movies and fashion

On both of the domain constraints, these systems require a large dataset to train, must be able to answer some types of requests and have a limited number of types of answers, and so, these agents tend to be hard to materialize. Developing them in a domain specific manner helps in getting it to work properly.

As shown in Figure 2.1, these agents usually have some type of graphical support, being it either text and/or audio and/or image and/or video, then requiring one or more processing modalities. Also, there might be the need of even more processing modalities when the type of content of the user query does not match the dataset content type [32, 10].

Having some possibilities on the modalities of the interactions, [32] analyses how the generation of a response including image description impacts the performance of the system, which by itself is a hard task since it involves the extraction of visual features and domain knowledge. Comparing text responses, the systems performs well in all the scenarios except when it needs to use it due to the difficulty of this task, which requires a good transformation from images to their text description. But when comparing image responses, the addition of a description of the images improves the performance of the selection by the user.

With these base concepts about task-oriented conversational agents, let's start by discussing the different state of the art types of feedback this kind of assistants can eventually support, in Sections 2.2 to 2.4. Some of these will be used in the conversation simulation presented in Chapter 3 and by the automated assistant presented in Chapter 4.

2.2 Like/dislike Feedback

An interesting type of feedback is like/dislike feedback, consisting in letting the user to tell which products are similar (like) or dissimilar (dislike) to the one he is looking for, without being specific to what is missing in a particular product, which might also bring some more useful information.

2.2.1 Binary Classifier

A way to approach this is how [19] does, which trains a scoring function, Equation (2.1), to tell the system the rating of each product. For this, the system iteratively gets feedback from the user in the form of relevant or irrelevant product, organizing them in two sets: relevant images $\mathcal{R}$ and irrelevant images $\overline{\mathcal{R}}$. These two sets are then used to train an SVM classifier, which will learn to label products' images as positive examples (relevant) or negative examples (irrelevant), being the system able to create a ranking of the products according to all the user restrictions. An example of this approach is shown in Figure 2.2.

$$S^b(x_j) = w_b x_j^T + b \tag{2.1}$$

where w_b, b are the SVM parameters and x_j are the visual attributes extracted from the product image.

Figure 2.2: Like/dislike Example

2.2.2 Discussion

As used for the binary relevance feedback of [24, 25], this approach has a poor performance alone and mixed with other feedback strategies in the context of conversational image search. A possible reason for this is that the classifier is trained only using the products marked as liked and disliked, which are just a few comparing to the size of the dataset, leading to having an underfitted model situation.

Other approaches have been proposed in this regard:

- [2] allows for users to provide sequential feedback ("likes" and/or "dislikes") on search results to adapt the recommendation;
- [3] guides the interactive search in an Ask-and-Confirm fashion, where AI actively searches for discriminative details missing in the current query - composed by images of objects - and users only need to confirm AI's proposal.

2.3 Relative Feedback

Another interesting type of feedback is the relative feedback, which enables the user to make comparisons between products, allowing him to exactly tell what is missing in the mentioned product and not restricting his feedback to the form "A is relevant, B is not" or "A is more relevant than B". This also enables a more natural way for the user to interact, describe and compare products.

2.3.1 Category-Dependent

[39, 20, 13, 33] first solved this type of feedback by learning a model for each of the categories, enabling the user to make comparisons between product categories. This can be performed using binary classifiers and similarities.

More specifically, [13, 33] make use of prepositions and adjectives for a better contextual modeling of products, allowing a more natural way of comparing and relating them.

2.3.2 Category-Independent

Although the proposed methods in Section 2.3.1 are able to make some comparisons between products, the user is restricted to comparisons between the category of the object. [28, 19, 29] tackle this problem by making attributes comparisons category-independent, allowing the user to freely make comparisons between any products and categories by learning a multi-dimensional space where the axes represent the relative strength of the corresponding attribute. An example with two dimensions is shown in Figure 2.3.

Figure 2.3: Relative Attributes Example

For this, the system first learns the relative feedback, which is accomplished by annotating pairs of images (i, j) such that image i has a stronger presence of the given attribute than j or both i and j images present the equivalent strengths of the attribute. This annotation process cannot be done by asking annotators to give the score which would reflect how much of the attribute is present, since they might have different sensitivity and calibrations for the strengths.

To work around this issue, these annotated pairs are grouped into two sets: O_m where image i has a stronger presence of attribute m than image j, and E_m where both images have an equivalent level of presence of attribute m. Using this relative information between images, to learn an attribute's ranking function it is useful to use the large-margin ranking (Equation (2.2)), which consists in learning a ranking function that depends on the features extracted x_i or x_j from the image, the learned parameters w_m^T and the use of slack variables $\mathcal{E}_{i,j}$ and $\mathcal{Y}_{i,j}$ to handle the ranking variations. These computed ranking functions can then be used to map each image into a m-dimensional space, where each dimension corresponds to the relative rank prediction of attribute m.

In the m-dimensional space, if the images have different levels of presence of an attribute (from O_m set), they are represented at least an unit away from each other. Otherwise, if they appear to have similar strengths (from E_m set), they tend to be close to each other.

With each interaction with the user the system needs to update the score of each image. For this, the system adds the new constraint provided by the user to all the gathered constraints and analyses how the relative ranking changes in order to increase or decrease the score.

$$\begin{aligned} \text{minimize} \quad & \left(\frac{1}{2}\|w_m^T\|_2^2 + C\left(\sum \mathcal{E}_{i,j}^2 + \sum \mathcal{Y}_{i,j}^2\right)\right) \\ \text{s.t.} \quad & w_m^T x_i \geq w_m^T x_j + 1 - \mathcal{E}_{i,j}, \forall (i,j) \in O_m \\ & |w_m^T x_i - w_m^T x_j| \leq \mathcal{Y}_{i,j}, \forall (i,j) \in E_m \\ & \mathcal{E}_{i,j} \geq 0, \mathcal{Y}_{i,j} \geq 0 \end{aligned} \tag{2.2}$$

where C is a constant penalty, which measures the trade-off between maximizing the margin and satisfying the pairwise constraints.

2.3.3 Optimization

From this m-dimensional space, [18] worked on a mechanism to improve the efficiency of the attribute selection strategy, by teaching the system to choose an image and an attribute that will maximize its information gain, minimizing the number of interactions with the user.

To accomplish this, the system makes use of the relative strengths learned in [19, 29] to build a binary search tree per attribute. Every node of each tree represents a pivot, which can be used to compare the target product of the user with. Each pivot has two siblings, on its left the images that have a lower ranking than the pivot, and on its right the images which present a higher ranking than the pivot. A tree example is depicted in Figure 2.4.

When interacting with the user, the system chooses the next image-attribute pair question by considering the feedback gotten so far to compute the probability of the next user action as well as which attribute will most increase the information gain.

Figure 2.4: Relative Attributes Optimization Tree Example

2.3.4 Discussion

As proposed by [29], this m-dimensional space can represent products as m-dimensional arrays, which can be used for several tasks:

- Zero-Shot Learning from Relationships: each product has a category, since the product is related to this space, the category can also be related to the same space, thus allowing to make relative relations between categories by using attribute comparisons;
- Describing Images in Relative Terms: generating a description by comparing the image representation with other images or categories representations;
- Relative Relevance Feedback in Image Search: allowing the user to compare and describe its image relatively to others by using high-level properties. This enables the system to discard images that fall in the irrelevant portion of the relative attributes scales;
- Relative Feedback to Discriminative Classifiers: having a classifier to determine to which category an image fits into and also a supervisor which can tell the classifier if the classification was correct or not, the classifier can use the feedback from the supervisor to improve its training since all the images are represented in this space and can be compared with each other.

from which, as used for the free-form relative attribute feedback and suggested questions presented in [24, 25], and for the use case of this thesis, the Relative Relevance Feedback in Image Search is the most appropriate to adapt because the user knows how his target product is, but is unable to describe it properly by his own words. This way, allowing the system to question the user about how his product relates to one or more from the dataset, would allow both the user to tell more effectively how his target product is and the system to know which products to discard.

Table 2.1 summarizes the differences between the approaches considered for implementing this kind of feedback. Between the first two strategies, the category-independent one is better as it allows a deeper comparison between products, but incorporating the optimization is not helpful when the dataset gets big enough because of the memory it requires to decide the next attribute to use.

Relative Feedback	Category Comparison	Attribute Comparison	Attribute Efficiency	Memory Bootleneck
Category-Dependent	Y	N	N	N
Category-Independent	Y	Y	N	N
Optimization	Y	Y	Y	Y

Table 2.1: Relative Feedback Comparison

2.4 Sketch Feedback

Another type of interaction, used in the framework presented by [24, 25], is the sketch feedback, which allows the user to draw the shape of the product he has in mind.

2.4.1 Approach

For getting products similar to the sketch, the image is first passed through a CGAN that makes its conversion to a colored image, as described in [16] and exemplified in Figure 2.5, and then this image is used to extract the products' features which will be used for getting its probability of being of each of the available categories. These probabilities are used for ranking the products.

For simulating the user sketches, it is useful to use the technique described in [43] in order to convert an image into a sketch with its edges, which is then used as described for the real sketches.

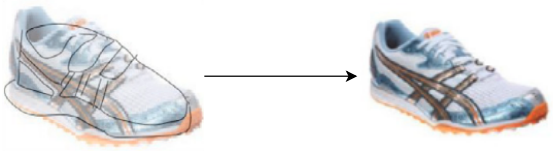


Figure 2.5: Sketch Example

2.4.2 Discussion

Although this in an interesting approach, for the context of this thesis this is not very suitable since the framework is aimed to be able to work in multiple domains like fashion and movies. For the example of the movies domain, this would require a lot of info regarding each movie for it to work properly, since a possible solution for this to work is to store some scenes of the movie for getting comparisons with the user's sketches. Hence, this interaction type requires some time from the user, is not scalable with the size of the dataset and depends in the aptness the user has for transforming its mental image of the product into a drawing.

2.5 Automating Selection

Having explored how we can treat the feedback from the interactions with the user, there is the need to choose which one is better in each time step of the conversation with the user.

2.5.1 Reinforcement Learning Background

For this task, it is useful to incorporate artificial intelligence inside the system, specially the area of deep reinforcement learning. This is a variant of reinforcement learning, which in

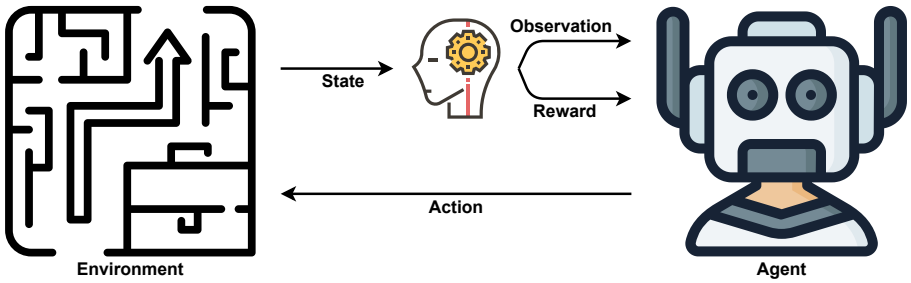


Figure 2.6: Reinforcement Learning Pipeline

the recent years has shown a great success in learning what works best in each interaction with the environment in order to maximize the cumulative reward, as shown in [25, 4].

Reinforcement learning, as illustrated in Figure 2.6 and explained in [31, 15, 1, 23], consists of an agent that decides the action to take next according to the state of the environment. After the change of the environment, it is processed (by sensors, for example) in order for the system to observe the new state of the environment and also, but not always, to get some positive or negative reward depending on what happened in the environment after the action. This feedback from the state is then used for improving the algorithm that makes the decisions. The rewards can only be computed after each interaction with the environment since for certain problems the dataset can only evolve as the agent interacts with the environment, not being able to get it all a priori.

These problems can be formulated as a Markov Decision Process, which consists in a tuple with some values:

- States (S): all the possible states of the environment;
- Actions (A): all the possible actions the agent can take;
- Probabilities ($P(s'|s,a)$): the probability of transitioning from state s to state s' using action a;
- Rewards ($R(s,a)$): the reward obtained in each possible transition;
- Initial States (S_0): all the possible initial states of the environment;
- Discount Factor (γ): the discount factor for future rewards, which can help the agent to learn that getting the reward sooner is better;
- Planning Horizon (H): the maximum number of steps the agent can take.

but for doing so, they must follow the Markov property which forces the current state to only depend on the previous state. This property is very useful to formulate reinforcement learning problems because it allows to only store the previous state's information, not needing to keep a lot of information for each action taken.

When the agent interacts with the environment, it is doing a sequence of actions, also called a plan. Following a plan without paying attention to the rewards and risks involved

in each action can end in bad results. So, it is important to take actions according to a policy π which accounts these rewards and risks. Although having a policy is important, it is also interesting to have some way to compare and evaluate policies, since one can lead to better results than others:

- State-Value function: gives the expected reward from state s of the environment by following policy π, as shown in Equation (2.3a);
- Action-Value function: gives the expected reward from state s of the environment by following action a of policy π, as shown in Equation (2.3b);
- Action-Advantage function: tells if taking action a from state s is better than simply following the action given by the policy π, as shown in Equation (2.3c).

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)\,(r_{s,a,s'} + \gamma v_\pi(s')) \tag{2.3a}$$

$$q_\pi(s,a) = \sum_{s'} P(s'|s,a)\,(r_{s,a,s'} + \gamma v_\pi(s')) \tag{2.3b}$$

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.3c}$$

With these three functions it is possible to reach an optimal policy, but the process to get there is time and resource consuming.

2.5.2 Deep Reinforcement Learning

When these problems become big enough, getting the rewards of each action from each possible state of the environment is computationally very expensive and almost impossible, since there are tons of possible combinations. This impossibilitates the computation of the Markov Decision Process specification. To overcome these issues, as described in [8, 7], the base concepts of reinforcement learning can be adapted in order for the reward functions specified in Equation (2.3) to be simulated by a neural network, knowing that:

- Feedback is delayed since some of the agent actions do not get any feedback;
- Evaluation of an action is only relative to others;
- Feedback is only gathered from samples generated by the interaction with the environment;
- Sometimes the environment observation cannot by directly used to make decisions.

As the agent interacts with the environment to get training examples, if he just makes use of what he already knows ends up in not exploring new actions which may be a better option. Having a trade-off between exploration and exploitation is essential.

In order to tackle this type of problems, the functions specified in Equation (2.3) can be adjusted to use temporal-difference learning:

- State-Value function: uses the reward of the current state combined with the rewards predicted for future states, accounting the difference between the total predicted value and the real value, as shown in Equation (2.4a);

- Action-Value function: there are two possibilities of computing this reward prediction

  - SARSA (State-Action-Reward-State-Action): estimates future reward by following the action a given by the policy π, as shown in Equation (2.4b);
  - Q-learning: estimates future reward by using the next best action a, as shown in Equation (2.4c).

$$v_{t+1}(s_t) = v_t(s_t) + \alpha_t \left(R_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t)\right) \tag{2.4a}$$

$$q_{t+1}^{SARSA}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t \left(R_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}) - q_t(s_t, a_t)\right) \tag{2.4b}$$

$$q_{t+1}^{Q-learning}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t \left(R_{t+1} + \gamma \arg\max_a(q_t(st+1, a)) - q_t(s_t, a_t)\right) \tag{2.4c}$$

Since the neural network estimates the values of these functions, these learning algorithms (SARSA and Q-learning) cannot be used as is, as they require some tables with data to be updated. This way, to be able to train a neural network with these algorithms, they need to be adapted one more time: SARSA in Equation (2.5a) and Q-learning in Equation (2.5b).

$$y_i^{SARSA}(a_t) = R_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}) \tag{2.5a}$$

$$y_i^{Q-learning}(a_t) = R_{t+1} + \gamma \arg\max_a \left(q_t(s_{t+1}, a)\right) \tag{2.5b}$$

2.5.3 Discussion

This machine learning technique to train a machine to perform tasks is used in a lot of contexts, being some of them:

- Autonomous Driving [17]: focuses on how this technique is used for making decisions on driving environments;
- Image Search [24, 25]: focuses on a context similar to this thesis, but also considering other kinds of contents (public figures and scenes);
- Object Localization [4]: focuses on an algorithm to extract objects from images;
- Visual Dialog [6]: focuses on an assistant capable of having a conversation with a human about visual assets.

All of these recent works, specially [17], show and prove that the Q-learning algorithm works better than SARSA to train the agent's neural network, since Q-learning ignores the policy and chooses the best possible action at next step, thus checking the overall best move.

Also, by a deeper analysis to the work that is most similar to this thesis [24, 25], it is possible to see that the reinforcement learning agent ends up in choosing human-initiated feedback in the first interactions with the user, while mostly using machine-based feedback in later interactions. A possible reason for this is that in the earlier interactions it is more useful to let the user to freely express how his desired product is, and in the later interactions it is better for the system to choose what information is missing, as the user might give repeated feedback, not helping the system to narrow down the search space.

2.6 Models Architectures

Having discussed how the assistant will learn what action to follow on each interaction with the user, it is important to also discuss some model architectures that can be used for the model responsible for selecting the feedback type used in each interaction with the user.

2.6.1 Very Deep Convolutional Networks - VGGs

The most commonly used architecture, used by [25, 4], is based on convolutional neural networks. This architecture is composed of multiple convolutional layers, each of which generates F filter maps from the data. Each of these filters extracts features from each of the input channels of the data (e.g. images in color have three channels: red, green and blue). Each value in each of the F maps is extracted by applying a kernel with dimensions height H and width W, having each of the kernels $H \times W$ weights.

Since the kernel weights are shared for each of the F filter maps, these layers end up having much less connections between neurons to learn, thus being faster and if the input suffers any simple modification (like translation), the result of these filters reflects the same modification.

When this type of layer is applied, the goal is to extract the maximum amount of information so, the tendency is to use large kernel sizes. However, the same results can be achieved using less weights by stacking smaller layers on top of each other. As introduced in [34] and explained by [37], the number of weights used in each of the kernels can be further reduced by using a pooling layer - which aggregates the results of the input map into another smaller map via the application of some operation like maximum or average to some map cells - after some convolutional layers, as shown in Figure 2.7.

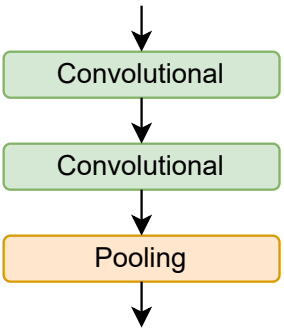


Figure 2.7: VGG Base Block Architecture

2.6.2 Residual Networks - ResNets

When neural networks get deeper, they become difficult - due to the vanishing gradient problem and its accuracy degradation - and take some time to train. To overcome these issues, as explained in [9] and introduced by [14], residual blocks were added to the architecture, which makes use of skip connections between layers, as shown in Figure 2.8.

The creation of this shortcut or skip connection enables the gradients to directly pass to a layer forward in the architecture and also ensures that the layers between the skip connection learn an identity function since their output is concatenated with the input of the block.

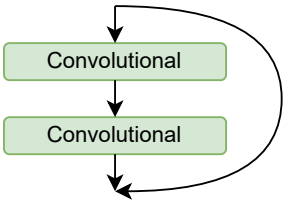


Figure 2.8: ResNet Base Block Architecture

2.6.3 Vision Transformers - ViT

Unlike the architectures presented above, transformers use the mechanism of self-attention, enabling them to take more attention on some parts of the input depending on the input instead of looking always the same way on the input.

As presented in [42] and explained by [38], a transformer is able to take more attention to certain parts of the input than others by using a self-attention mechanism called scaled dot-product attention, shown in Figure 2.9. This scaled dot-product uses three matrices (for the query, key and value), from which the attention is computed between the query and key matrices, to be then used to extract the relevant information from the input, which will be concatenated with the information extracted from previous iterations.

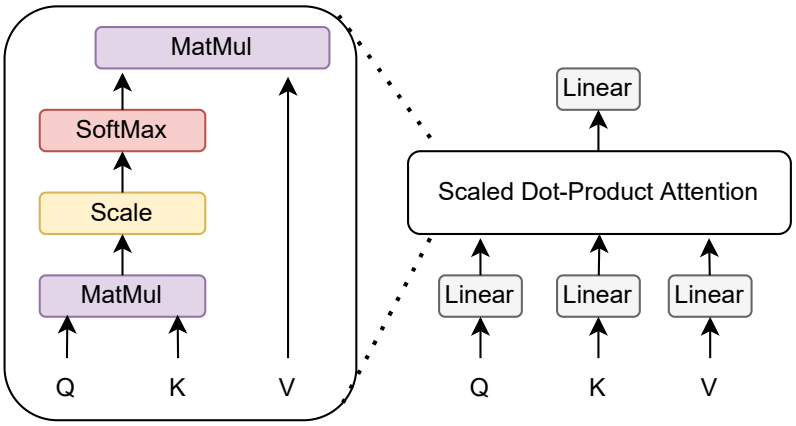


Figure 2.9: ViT Base Block Architecture

2.6.4 Discussion

Between these presented architectures, the Vision Transformers (Section 2.6.3), may be the most efficient to learn relationships between input data and the output decision, as they are the most recent ones.

2.7 Controlling the Risk

Making the assistant intelligent in a way that he is able to decide which question to ask next is a step forward to help the user getting his target product as soon as possible, but this question selection strategy does not take into account that keeping asking questions can become overwhelming or be unnecessary at all.

2.7.1 Approaches

To address this issue, [30] proposes a theoretical definition of a framework which is able to make decisions between the many actions the user and the agent can do and shows that

this kind of framework is needed for improving the conversational information retrieval in general.

Later, [35] implements a base framework for risk control, based on this theoretical definition, which decides if it is to answer or not the user question based on the prediction of the uncertainty of the benefit of answering the user's question.

Based on this base framework, [40] proposes an extended framework which aims to help the agent to decide whether to answer the user's question or to ask a clarifying question based on the risk involved. As shown in Figure 2.10, this framework is composed by three components: two rerankers, one for ranking the questions and another for ranking the answers, and a decision making model (represented in bold in Figure 2.10), which is where this framework focuses its attention on.

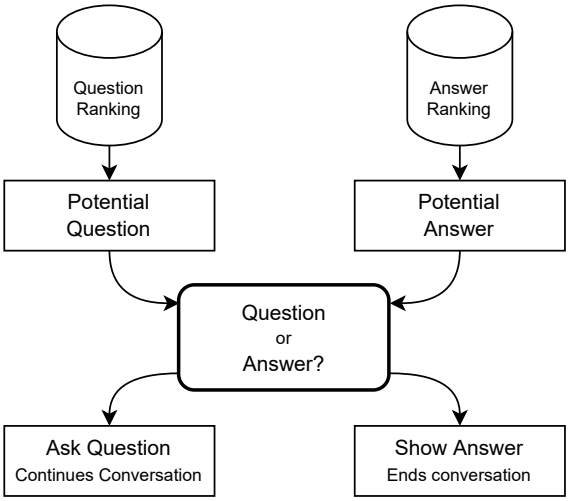


Figure 2.10: Controlling the Risk

For the risk decision module to decide when to answer or ask the selected question, it starts by encoding the initial user query q, the context of the whole interaction with the user h, the top k clarifying questions $\{cq_1, ..., cq_k\}$ and the top k answers $\{ans_1, ..., ans_k\}$ by passing them through a BERT-based model - Bidirectional Encoder Representations from Transformers, a transformer-based model for natural language processing - and getting the [CLS] token vector, which represents an embedding of the whole input of the model. Then, these features are concatenated with the scores of the top k clarifying questions $s_{cq}^{1:k}$ and the top k answers $s_{ans}^{1:k}$, which composes the state that is fed into the neural network that will predict the rewards of answering or asking the question $y = (r_{cq}, r_{ans})$.

With the result from the risk decision module, the agent then presents either the question or the recommendation to the user, depending on what has the higher expected reward.

2.7.2 Discussion

Table 2.2 summarizes the differences between the approaches presented to consider the risk. The main difference between the base and the extended framework is that the base one only determines if it is beneficial to ask or not the question, not doing anything if the question is not suitable to use, while the extended one also considers the risk between answering to the user and asking further questions to him.

Controlling the Risk	Generates Question	Generates Answer
Base Framework	Y	N
Extended Framework	Y	Y

Table 2.2: Controlling the Risk Comparison

2.8 Catalogues

With every technical detail revised, it is now time to look at the testing environment, which is mainly defined by the catalogue being used. For this, let's analyse the available catalogues and their information.

2.8.1 Available Options

As shown above, this kind of systems may have multiple types of feedback from the user, and each of them may require different types of information from each product. So, it is useful that the catalogue incorporates that information.

There are several catalogues that are usually used for testing these recommendation systems, presented in Table 2.3.

	Description	Size	Binary Attributes	Relative Attributes
Pubfig [20]	Public figures	~700 images	11	11
Scenes [26, 28]	Outdoor scene recognition	~2k images	6	6
Shoes [19]	Attribute discovery	~14k products	10	10
Deep Fashion [21]	Fashion	~800k images	1000	0

Table 2.3: Catalogues Comparison

2.8.2 Chosen Option

From the feedback types explored and discussed before in Sections 2.2 to 2.4, for the development of this thesis the most appropriate one is the one containing shoes. This dataset is particularly useful by having a good amount of attributes, being them binary and relative, which is optimal for supporting the feedback types used.

3

Conversational Interactive Recommendation

In this chapter we present how the user and the assistant interact with each other, focusing on the available question types. This includes what information is required from the catalogue, how the conversation simulation is performed, how the entire system is logically organized for the learning process and how each product is estimated to be a candidate for recommendation.

3.1 Problem Statement

For simulating a conversation with some interactions between the assistant and the user, the system illustrated in Figure 1.2 was developed. This simulation requires some information from the catalogue, which must be composed of several products

$$\mathcal{D} = \{(p_1, c_1, b_1, r_1, e_1), ..., (p_i, c_i, b_i, r_i, e_i)\} \tag{3.1}$$

where $p_i \in R^{h \times w}$ is the photo, c_i is the category, b_i is the set of binary attributes, r_i is the set of relative attributes, and $e_i \in R^n$ is the embedding (a n-dimensional representation) of the product i. All of these characteristics related to product i, as exemplified in Figure 3.1, are defined as

$$\mathbf{c_i} = [c_{i,1}, ..., c_{i,j}] \quad (3.2) \qquad \mathbf{b_i} = \{b_{i,1}, ..., b_{i,j}\} \quad (3.3) \qquad \mathbf{r_i} = \{r_{i,1}, ..., r_{i,j}\} \quad (3.4)$$

where

Categories each individual category $c_{i,j} \in \{0, 1\}$ in Equation (3.2) indicates if category j is present in product i, conditioned by $\sum_{j=1}^{\#categories} c_{i,j} = 1$, i.e. only one category is associated with a product. But, since only one category can be associated with a product, this is implemented as being a binary attribute with multiple possible values: $c_i \in \{1, ..., \#categories\}$;

Binary Attributes each individual binary attribute $b_{i,j} \in \{0, 1\}$ in Equation (3.3) indicates if product i has attribute j.

Relative Attributes each individual relative attribute $r_{i,j} \in \mathcal{R}^1$ in Equation (3.4) indicates how much of the property j the product i has.

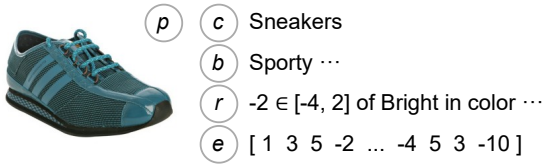


Figure 3.1: Product Sample

3.1.1 Conversation Flow

With this information, it is possible to understand in greater detail how the conversation operates, Figure 3.2 presenting the main interactions between the user and the assistant.

Before the start of the conversation, the user simulator picks a product from the catalogue to serve as his target product. With that product in the "mind" of the user, he then reaches the assistant and tells him some initial details regarding the product (seed query).

With this initial information regarding the user's target product, the assistant can now start the conversation itself. For each of the interactions, the assistant chooses the next question type to use, generating either a text-only question or a question with text followed by one or more recommended products, depending on the question type being used.

Having the question generated, the assistant presents it to the user and lets him either answer the question or give its preferences over the recommended product(s) by attribute comparisons, text-only and text followed by products respectively.

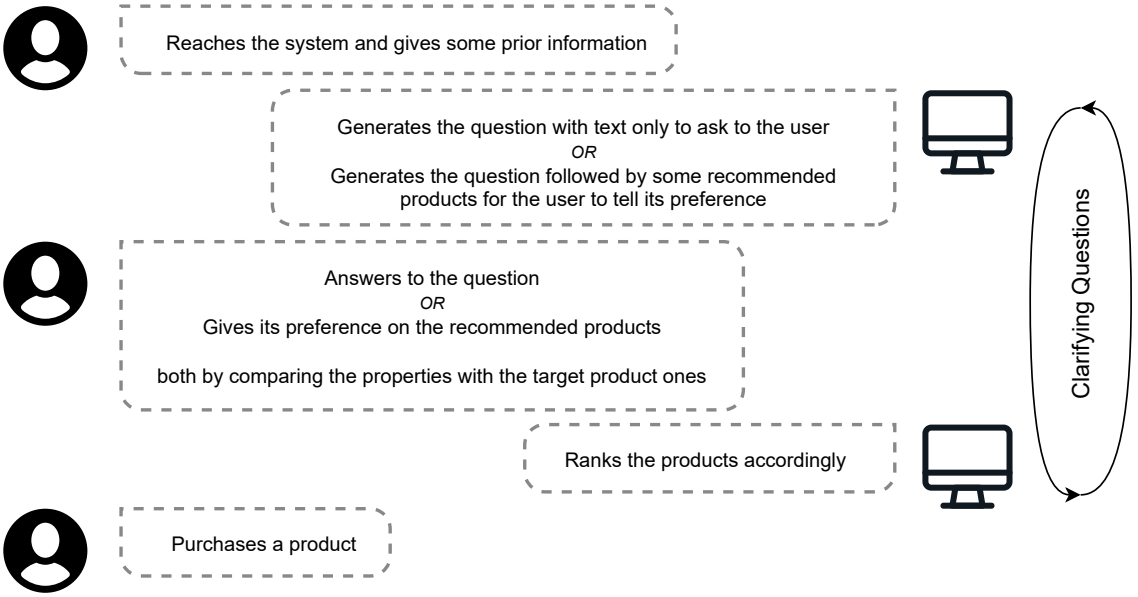


Figure 3.2: Conversation Flow

The assistant then gathers the user's answer and processes it in order to update the products' ranking, which is used to collect the final recommendation. This final recommendation is used by the user to know, at the end of each interaction, whether he got his target product, so that he can signal the assistant he already got his product. However, if the user never ends up seeing his product being recommended at the end of each interaction, the assistant ends the conversation after a pre-defined number of interactions.

3.1.2 Implementation Details

Knowing the conversation flow, further details on how the conversation simulation works are presented in Algorithm 1, where the user's target product is specified in Line 4 and the initial details of the target product in Line 6.

Related to the conversation itself, there may be multiple interactions between the user and the assistant as defined in Line 7. For each of them, the assistant first generates the question, then the user answers it and finally the assistant processes that information, respectively in Lines 8 to 10.

The end of the conversation is triggered either when the limit number of interactions is reached in Line 7 or the user sees his product being recommended in Line 13 and signals the assistant that it happened in Line 14. After getting the recommended products in Line 11, the assistant computes some statistics in Line 12: available products, time each question takes to be answered and the number of products found after that interaction.

Algorithm 1: Interactions Simulation

Data: $\mathcal{D}$, Q - question types, #repetitions, #interactions, P - size of list of products, Sq - seed query
Result: List of P products

```
1  assistant ← Assistant()
2  user ← User()
3  dialogue ← []
4  target_product ← random_product(D)
5  for each repetition do
6      dialogue.append(assistant.seed_query(Sq, target_product))
7      for each interaction do
8          dialogue.append(assistant.question(D, Q))
9          dialogue.append(user.answer(target_product, dialogue))
10         assistant.process_answer(D, dialogue)
11         top_products ← assistant.recommendation(P)
12         assistant.compute_statistics(target_product, top_products)
13         if target_product ∈ top_products then
14             user.purchase()
15         end
16     end
17 end
```

3.1.3 Learning Architecture

Having explored how the entire system works, it is now possible to understand deeper how the assistant will learn to choose the question type to use in each interaction with the user, in real-time, during the conversation.

As discussed in Section 2.5, similar works like [24, 25] tackle similar problems using deep reinforcement learning, which is based on trial and error. As such, the agent keeps learning by continuous interaction with the environment and getting feedback after each action taken. In Figure 3.3 it is possible to check how we integrated the reinforcement pipeline on this thesis problem context:

Agent incorporates the Question Type Selector, which is where the neural network model is placed, and so, this module is responsible for deciding which action to follow, i.e. the question type to follow. This decision is made based on the expected rewards of taking each of the available actions, which are computed according to the current state of the conversation;

Environment is responsible both for simulating the conversation between the user and the assistant, and for processing the user's answer in order to compute the expected reward of taking the action decided by the agent, so that the agent can improve its decisions;

System also processes the user's answer, but for getting an observation, which is used for updating the state of the conversation.

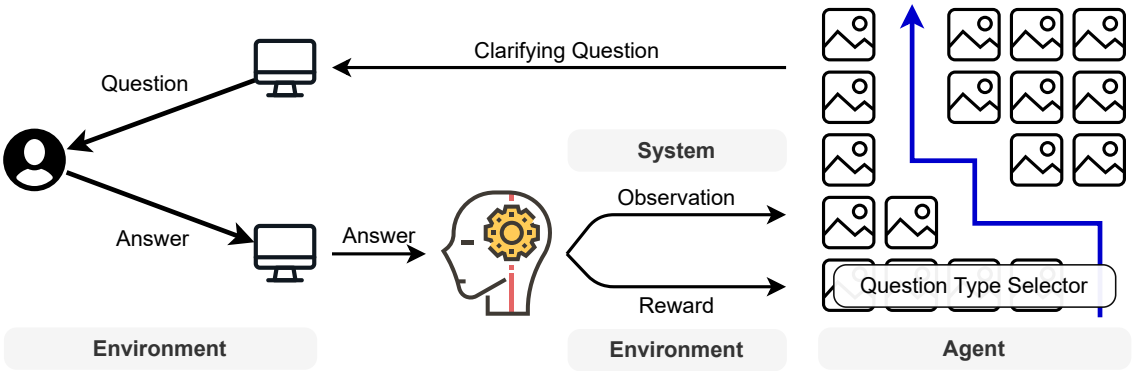


Figure 3.3: Reinforcement Learning Flow

The main delimiter interactions that make the conversation possible - seed query and purchase - are presented in Section 3.2, and the interactive clarification dialogue is defined by the question types presented in Section 3.3.

3.2 User-Assistant Interactions

As presented in Section 3.1 and more specifically in Algorithm 1, the dialogue between the user and the assistant always starts with the user giving a seed query, and always finishes with the user signalling he got or found his desired product, or the number of interactions reached the specified limit. Between these two main interactions there is another main interaction, interactive clarification dialogue, which is supported by the question types.
Main interactions:

- **Seed Query** (category possibly with attribute(s) keyword): the user is able to tell some information about its target product before the actual conversation begins;
- **Interactive Clarification Dialogue**: binary questions, relative questions and comparative questions;
- **Purchase**: for the user to tell the assistant he found his target product or something similar.

Where the interactive clarification dialogue part supports three different types of questions:

- **Binary Attribute Question** (no products are shown): the assistant asks the user whether the *desired product* has the *target attribute* a_i;
- **Relative Attribute Question** (only one product is shown): the assistant presents *one reference product* p_j and *indicates an attribute* a_i, then, the user relates the *target product* to the *reference product* according to the *indicated attribute*;
- **Comparative Attribute Question** (multiple products are shown): the assistant presents a *list of products* and *indicates an attribute* a_i, then, the user *selects the product* p_j with the most similar attribute to the *target product*.

To start, let's understand how the two conversation delimiter interactions - seed query and purchase - are handled.

3.2.1 Seed Query

Once the user reaches the assistant, before starting the conversation, he gives some prior information to the assistant about its target product, similar to entering a store. These details, as specified in the only utterance before the conversation u_{seed_query} presented in Equation (3.5) and Line 6 of Algorithm 1, include the category $c_{u_{seed\_query}}$ and, optionally, one or two attributes $b_{u_{seed\_query}}$ of the user's target product tp. This is used by the system to update the initial products' ranking.

$$u_{seed\_query} = (c_{u_{seed\_query}}, b_{u_{seed\_query}}) \qquad \text{where } b_{u_{seed\_query}} \in \{\emptyset, tp_{b_0}, \{tp_{b_0}, tp_{b_1}\}\} \tag{3.5}$$

3.2.2 Purchase

The last interaction of a conversation is when the user tells the assistant he found his target product or something similar by sending a signal to it, as defined in the last interaction of the conversation shown in Line 14 of Algorithm 1, so the conversation does not need to carry on.

3.3 Question Types

Knowing the logic of the delimiter interactions of the conversation, let's look at how each of the question types - the actions of the reinforcement problem - used in the interactive clarification dialogue operate. The differences between the question types are summarized in Table 3.1.

Question Types	Binary	Relative	Comparative
Products	-	1	5
Answer	Yes *or* No	More, Less *or* Equally	1, 2, 3, 4 *or* 5

Table 3.1: Question Types Comparison

3.3.1 Binary Attribute Question

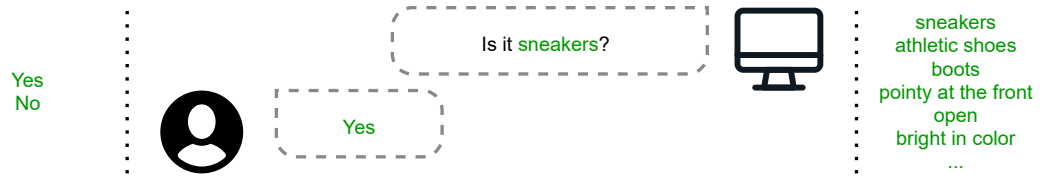


Figure 3.4: Binary Attribute Question Example

In this action, the assistant generates a clarifying question for the user to answer with **Yes** or **No**, depending on whether his target product has the asked attribute. An example of this action is illustrated in Figure 3.4. To accomplish this:

Question the assistant starts by choosing the next binary attribute to use $b_{binary\_question}$ - *sneakers* in Figure 3.4 - and outputs the clarifying question $u_{binary\_question}$ for the user to answer, as specified in Equation (3.6).

$$u_{binary\_question} = b_{binary\_question} \tag{3.6}$$

Answer the clarifying question is then presented to the user, who checks if his target product has the requested attribute $b_{binary\_question}$, giving as an answer Yes or No - *Yes* in Figure 3.4:

- Which is translated to the number corresponding to the assistant's numbering of the categories values, in the case of the categories (Equation (3.7)); or

$$u_{binary\_answer} = value \in \{1, ..., \#categories\} \tag{3.7}$$

- Which is associated directly to the attribute, in the case of a simple binary attribute (Equation (3.8)).

$$u_{binary\_answer} = value \in \{Yes, No\} \tag{3.8}$$

State Update the assistant obtains the user's response u_{binary_answer}, as shown in Equations (3.7) and (3.8), and checks the attribute $b_{binary\_question}$ for every product:

- Discarding the ones which do not have that specific value for that attribute, in the case of the categories (Equation (3.7)); or
- Discarding the ones which do not have that attribute, in the case of a simple binary attribute (Equation (3.8)).

3.3.2 Relative Attribute Question

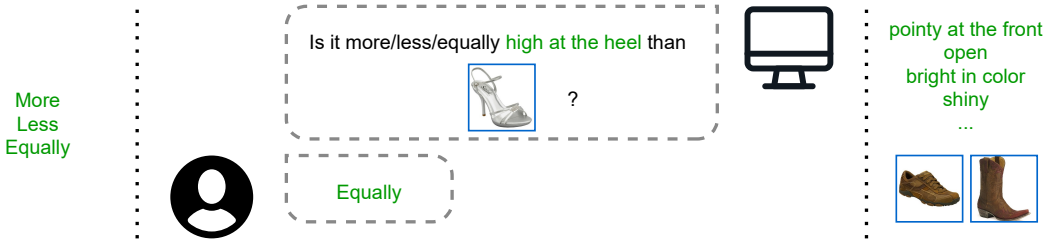


Figure 3.5: Relative Attribute Question Example

In this action, the assistant questions the user with an attribute and a reference product for him to answer with **More**, **Less** or **Equally**, depending on how his target product compares to the specified one. Figure 3.5 shows an example of this action. To achieve this:

Question the assistant first chooses the next relative attribute to use $r_{relative\_question}$ - *high at the heel* in Figure 3.5 - and picks the product with the relative strength scale (where each product is represented in a m-dimensional space, in which each axis represents the level of presence of the respective attribute, as defined in Section 2.3) closest to the average of the corresponding attribute strengths as being the reference product $p_{relative\_question}$ - *image* in Figure 3.5 - , producing the clarifying question $u_{relative\_question}$ for the user, as stated in Equation (3.9).

$$u_{relative\_question} = (r_{relative\_question}, p_{relative\_question}) \tag{3.9}$$

Answer the clarifying question is then presented to the user, who compares the value of the referenced product $p_{relative\_question}$ with the value of his target product tp and answers by telling if his target product has More, Less or Equally - *Equally* in Figure 3.5 - of the specified attribute $r_{relative\_question}$ than the mentioned product.

$$u_{relative\_answer} = value \in \{More, Less, Equally\} \tag{3.10}$$

State Update the assistant gathers the user's response $u_{relative\_answer}$, as specified in Equation (3.10), and applies that comparison to all available products using the referenced attribute, discarding the ones not satisfying the relation between the two products, i.e. not satisfying the restriction given to the referenced attribute.

3.3.3 Comparative Attribute Question

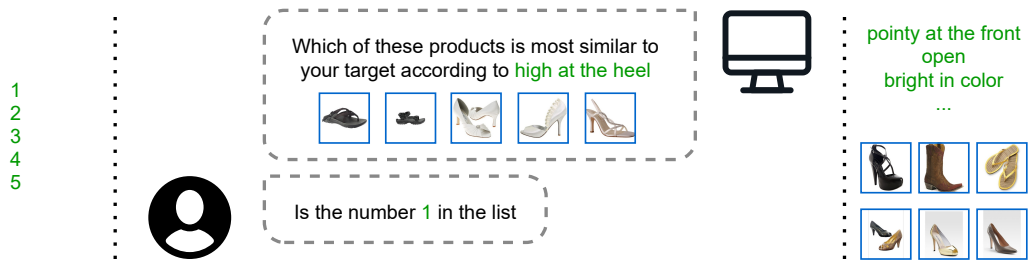


Figure 3.6: Comparative Attribute Question Example

With this action, the assistant questions the user with an attribute and a list of products for him to answer telling which of them is most similar to his target product. This action is exemplified in Figure 3.6. To perform this:

Question the assistant starts by choosing the next relative attribute to use $r_{comparative\_question}$ - *high at the heel* in Figure 3.6 - and then picks five products Equation (3.11) - *images* in Figure 3.6 - , which their values are the closest to the minimum, $\frac{1}{4}$, average, $\frac{3}{4}$ and maximum values within the relative strength scale of attribute $r_{comparative\_question}$, without product repetitions whenever there are enough available products.

$$products_{comparative\_question} = \bigcup_{i=0}^{4} p_i \tag{3.11}$$

Having both the attribute and the reference products picked, the assistant generates the clarifying question $u_{comparative\_question}$ as shown in Equation (3.12).

$$u_{comparative\_question} = (r_{comparative\_question}, products_{comparative\_question}) \tag{3.12}$$

Answer the clarifying question is then presented to the user, who compares the values of each of the referenced products $products_{comparative\_question}$ with the value of his target product tp and answers by telling which of those referenced products $p_{comparative\_answer}$ is the closest to the value of the specified attribute $r_{comparative\_question}$ of the user's target product tp.

$$u_{comparative\_answer} = p_{comparative\_answer} \in products_{comparative\_question} \tag{3.13}$$

State Update the assistant gets the user's response $u_{comparative\_answer}$, as presented in Equation (3.13), and applies the range of values restriction to all available products using the referenced attribute, discarding the ones which are far from the user selection.

3.4 Ranking Products

To conclude the conversation specification, a computation required after each interaction between the user and the assistant for knowing the relevant products is the products' ranking. This computation is performed even if there is no need to recommend products, since it is still needed for the computation of the conversation statistics as mentioned in Section 3.1.2.

When the conversation starts, this computation assigns a score of 1 to each product and, after each interaction with the user, the ranking is updated by multiplying the current ranking with the ranking of the current restriction, which is gathered from the answer processing of each of the question types. The score of each product can be interpreted as its probability of satisfying all the user restrictions. This is used for:

- Computing statistics in order to evaluate the performance of the conversation as specified in Sections 3.1.2 and 5.1.3, such as:
  - Percentage of products found per conversation interaction;
  - Number of available products per conversation interaction;
  - Time consumed by each question type for getting feedback from the user per conversation interaction.
- Giving the final recommendation (Section 3.2.2), by selecting the highest scored products from the products' ranking;
- Selecting products to be shown in some of the question types.

Having explored how the interaction between the user and the assistant is simulated, how each of the Question Types operate and how the assistant keeps track of the user restrictions, Chapter 4 describes in detail everything related to how the assistant learns to choose the next action (question type) within the conversation, as well as how the attributes are chosen for each clarifying question.

4

Selection of Clarifying Questions

In this chapter we explain in detail everything related to how the assistant selects and learns to select the next action to use on the next interaction of the conversation, as well as how the attributes are chosen for each clarifying question.

4.1 Question Type Selector

Although in this thesis we developed the entire system to support the interactions between the user and the assistant as shown in the previous chapters, this work focuses on the development of the selection of the question type to be used in each interaction, i.e. the action chosen by the Question Type Selector.

4.1.1 Architecture

For the neural network to work properly, it needs to receive some relevant information, so that it can take informed decisions. The architecture used for the model is based on the one used by [24, 25], which assumes that the assistant is able to know which product the user is looking for, enabling the authors of [24, 25] to compute the closest and furthest products from that target, additionally passing that information to the model.

Since our assistant operates separately from the user simulation, it is not able to know the target product, thus not being able to compute those closest and furthest products. This way, as can be seen in Figure 4.5, we adapted their model architecture to the context of the problem of this thesis by removing those closest and furthest products information from the history of the conversation:

State is the source of information of the model, represented in Figure 4.1, and expects the following information:

Top Products history containing the latest 3 top products lists - first 3 in Top Products size. Each of the top products list has 3 products - second 3 in Top Products size - , each of which is represented by an embedding with shape R^{960}, which

Figure 4.1: State of Model Architecture

is a vector representation of each product within a 960-dimensional space - 960 in Top Products size;

Actions history containing the latest 3 taken actions - first 3 in Actions size. Each of them corresponds to the question type used and is represented as one-hot encoding, with 3 possible actions - second 3 in Actions size.

Regarding the model being expecting an input with history size of 3, since the authors of [4] have discussed and proven that having a history of the various state information is helpful for the model to know what happened in the past. This essentially prevents the model from getting stuck in repetitive cycles of decisions. Also, as these authors discussed, this historic information strategy also helps in saving space, since a possible alternative to accomplish this history stack would be to store the previous states themselves, occupying much more space.

Data Compression and Feature Extraction is responsible for processing the state of the conversation towards extracting useful features for the classification part. This part of the model is actually only applied to the Top Products of the State of the model and is represented in Figure 4.2;

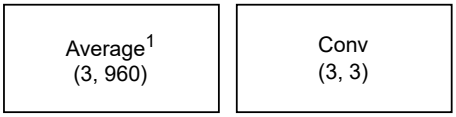


Figure 4.2: Data Compression and Feature Extraction of Model Architecture

Data Compression composed by the *average* layer, used for getting the average representation of each top products list. An example of this operation is depicted in Figure 4.3, where the gray dots are the top products - a list with 3 products - and the black dot represents the average embedding of those products;

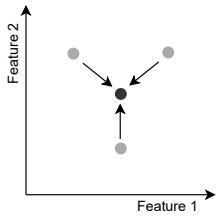


Figure 4.3: Top Products List Compression

Feature Extraction composed by the *conv* layer, used for extracting some features from the relationship of the evolution of the top products stack. The actions are not processed by any *conv* layer so the agent can clearly analyse the history of actions taken.

Classification is responsible for mapping the conversation history, which includes the latest 3 actions and respective states, into one of the 3 available actions - 3 in the right most Linear layer. In this problem context, this part is the one responsible for taking the decision of which of the 3 question types to follow in the next interaction of the conversation, based on the features extracted from the conversation so far. This part of the model is shown in Figure 4.4.

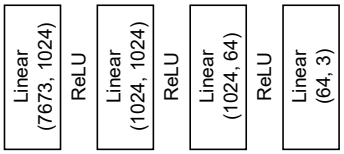


Figure 4.4: Classification of Model Architecture

These three parts of the model architecture, showed in Figures 4.1, 4.2 and 4.4, are combined to work together in Figure 4.5, where:

- **State**: is presented at the left, receiving the conversation history data;
- **Data Compression and Feature Extraction**: is presented in the middle, receiving the Top Products from the State of the model and transforming them in order to extract useful information for the classification part;
- **Classification**: is presented at the right, receiving the features extracted by the middle part (Data Compression and Feature Extraction), performing non-linear transformations in order to get the predicted rewards of taking each of the 3 available actions, given the conversation history.

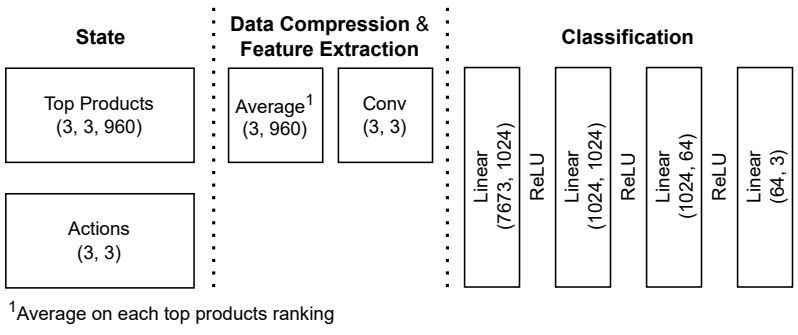


Figure 4.5: Model Architecture

4.1.2 Behaviours

Having discussed the model architecture, and since this problem requires continuous interaction with the environment in order to learn from its decisions, it is possible to help the assistant to learn specific behaviours by shaping the reward function in different ways, as explained and specified in Sections 4.1.2.1 to 4.1.2.4, and later confirmed with results in Chapter 5.

It is also important to note that each of the actions (the question types) might require different amounts of time to get feedback from the user. For the development of the following behaviours we estimated some durations for demonstration purposes, as defined in Table 4.1.

Estimated Durations	Binary	Relative	Comparative
Units of time	2	5	10

Table 4.1: Actions Estimated Durations

Additionally, by training the agent to follow some type of behaviour, the assistant might tend to choose the most appropriate follow up actions in order to achieve best effort results.

4.1.2.1 Sanity Check

To begin with, and in order to ensure the assistant is actually training and everything is correctly set up, the most simple reward schema that can be used with this goal is returning a single and fixed value. The reward function used for this is presented in Equation (4.1) and is further explained in Section 5.2.4.1.

$$reward = 0 \tag{4.1}$$

By having the same value for every decision taken, the assistant does not learn anything useful, but to mainly use the first available action since the various available actions will tend to be evaluated with the same value. As the first available action on the output of the Question Type Selector is the Binary Attribute Question, the assistant tends to use a lot of interactions per conversation and might not always find the product the user wishes to find when the catalogue size is big enough, tending to have a lot of products categorized into the same amount of categories to select only three of the selected category products.

4.1.2.2 Minimizing Interactions

With the statements of the previous reward schema, which simply associates each action with a single and only that value, leading to mostly choosing the first available action, we intend to make the assistant to learn to use as less interactions as possible. This way, the assistant can achieve the main goal of this thesis work and also the goal of the work we

based ours on [24, 25], but leads on the reward schema to have a greater attention on the number of interactions used.

As explored in Section 5.2.4.2 and the final result shown in Equation (4.2), by defining the reward function to be dependent on the number of interactions of each conversation, the assistant learns to reduce the number of interactions. However, this reward schema leads to the use of only one of the actions, since the Relative Attribute Feedback is capable of restricting the range of values of an attribute to a single value whenever the referenced product matches the user's target product value for that attribute.

$$reward = -\tanh\left(\frac{\#interactions - 6}{6}\right) \tag{4.2}$$

4.1.2.3 Minimizing Repetitiveness

Although the previous modification to the reward schema achieves the goal of our work, we went a step further and explored more ways of using the various actions during the conversation with the user, which might also help the assistant in avoiding local minimums. Also, with the previous modification to the reward schema we get a lot of repetitiveness regarding the action being chosen in each interaction of the conversation, since there may be one or more interactions with similar properties and capabilities of the Relative Attribute Question. This repetitiveness can become annoying and/or be unnecessary at all.

To overcome this challenge, as investigated in Section 5.2.4.3 and the final result defined in Equation (4.3), we want to instruct our assistant that varying the actions is a good thing, and for that we need to shape the reward function to be greatly dependent on the number of different actions used during the conversation.

$$reward = 3 * \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{4.3}$$

By defining a reward schema to be mostly dependent on the number of different actions taken, leads to an increase in the estimated time consumed during the whole conversation between the user and the assistant, since different actions (which correspond to different question types) have different feedback times from the user.

4.1.2.4 Minimizing Repetitiveness & Time

As discussed in the previous behaviour, avoiding repetitiveness is helpful but there might still be the case where some of the actions end up being repeated, specially when the variety of question types is not larger than the number of interactions used. In these cases, the reward schema also has to account for the estimated time each action takes to have feedback from the user instead of only taking care of the variety of actions taken, like the

ones shown in Equations (4.4a) to (4.4c) - each of them representing a different solution for this purpose - and explained in Section 5.2.4.4.

$$reward = 0.5 * \left(3 * \tanh\left(\frac{\#used - \#total - 3 - 0.5}{12}\right)\right) - \tanh\left(\frac{0.2 * (\#time - 60)}{12}\right) \quad (4.4a)$$

$$reward = -\tanh\left(\frac{0.2 * (\#time - 10)}{12}\right) if\ (\#actions) > \frac{3}{2} else - \tanh\left(\frac{0.2 * (\#time - 5)}{12}\right) \quad (4.4b)$$

$$reward = -\tanh\left(\frac{\frac{\#actions}{3} * (\#time - 10)}{12}\right) \quad (4.4c)$$

4.1.2.5 Summary

With these various solutions for the behaviours of the assistant, the one to choose depends on the expected behaviour of the assistant, keeping in mind that each of them has its own trade-offs:

Minimizing Interactions teaches the assistant to use as less interactions as possible during each conversation, but mostly uses only one of the actions;

Minimizing Repetitiveness teaches the assistant to repeat as less as possible each of the available actions during each conversation, but the number of interactions as well as the time consumed during the conversation increases;

Minimizing Repetitiveness & Time teaches the assistant to repeat as less as possible each of the available actions during each conversation, while keeping attention to the each of the used actions needs to get feedback from the user. This can be decomposed in three versions:

Version 1 tends to use most actions, but does not use them evenly and uses the most basic action (Binary Attribute Question) the same number of times independently of the size of the catalogue;

Version 2 tends to use most actions and uses them more evenly;

Version 3 tends to use every available action, but does not use them evenly, requires more interactions and consumes more time.

4.1.3 Hyperparameters

With the appropriate information being given to the model and having defined the various behaviours the assistant supports, there is still the need for it to learn how to properly use that information to make good decisions. For this purpose, the training process was implemented with the following hyperparameters:

Experiences Queue : since in reinforcement learning the agent learns by continuous interaction with the environment, it is important to store each of these interactions in a queue, so they can be used for training. Before starting training, 10000 experiences were generated and stored in a queue capable of storing 50000 total experiences. When this limit is reached, the ones added first, i.e. the older ones, are discarded;

Action Generation : for generating random actions, we used a probabilistic strategy which begun with 1.0 and gone progressively down to 0.1, as suggested by [24, 25]. With these random values, whenever these values pass a specific threshold value, then an interaction with a random action is used. This is useful whenever it is needed to generate new random interactions for the agent to have diversified experiences, i.e. when generating the initial interactions and when the agent is exploring the environment;

Loss Function : for evaluating how much the agent's predictions deviate from the actual results, there is a function that compares both values and returns their difference. For example, when used for classification this function checks if the action chosen was the right one, when used for labeling this function checks if the agent is mapping to the right labels.

In the case of reinforcement learning, this function compares the reward given by the agent with the expected rewards, where:

Chosen Action Reward : when the current state of the conversation is passed through the agent, it will give as output a list of values, each of them corresponding to each available action. Here, the action chosen for the next interaction of the conversation is the one with the highest predicted reward;

Expected Action Reward : this is computed based on the expected reward based on the current state of the conversation added by the discounted future expected rewards to the current question type reward;

Comparison : as discussed above, since the agent is used for choosing the action which presents the best predicted reward, there is the need for the agent to learn the actual reward values and not only which one is chosen. Therefore, it is needed to compare each of the output values with the expected ones, which is performed by using mean squared error, as shown in Equation (4.5).

$$loss = mse(chosen\_action\_reward, expected\_action\_reward) \tag{4.5}$$

Optimizer : although we already have the loss function defined for the problem, there is yet the need to update the various parameters of the agent, such as the weights and the learning rate, so the loss value can decrease.

For this purpose, the authors of [24, 25] used RMSProp, but we chose to use Adam because of its simplicity and practicality, as discussed in [36, 27], with a discount factor of 0.99;

Stopping Criteria : as discussed in [44], when working with deep reinforcement learning agents, the overfitting situation can actually be positive since it means that the agent has learned to effectively and efficiently perform tasks inside that environment. With this in mind, we let the agent to train until the measured metrics stabilise;

Validating and Testing Sets : for evaluating the performance of the agents trained, we then run them using always the same target products and also the same products in the catalogue in order to have a fair comparison in terms of contexts. This is further detailed in Section 5.1.3;

Frameworks and Libraries : for developing and researching variations of the solution developed in this thesis, there is demand for the use of a deep learning framework that treats of the engineering part for us, letting us focus on the development and research;

PyTorch Lightning gives a high-level application programming interface for PyTorch, which is a very popular deep learning framework and gives the ability to separate the research code from the engineering code.

4.2 Attribute Selector

Having the action chosen for each interaction of the conversation, there is still the need to select the attribute to use in the clarifying question generation. This selection can be performed using binary search trees for each attribute as suggested in Section 2.3.3, which is a good idea to solve this challenge, but with the increase of the size of the catalogue it becomes inefficient in terms of used memory and specially the time required for their construction. In order to facilitate this selection implementation, we came up with two different solutions:

- **Round-Robin** makes the Attribute Selector to choose the attributes one by one from the first one to the last one available. With this strategy it is certain that the attributes picked will have a great variety, thus there is a good probability of getting different feedback in each interaction with the user; or
- **Random** makes the Attribute Selector to choose the attributes randomly. With this strategy there is the possibility of repeating the same attribute multiple times, which might not help in getting much useful information.

but since our focus is on how to choose which question type to use, we assumed their selection in a round-robin fashion, which is also how the authors of [24, 25] do.

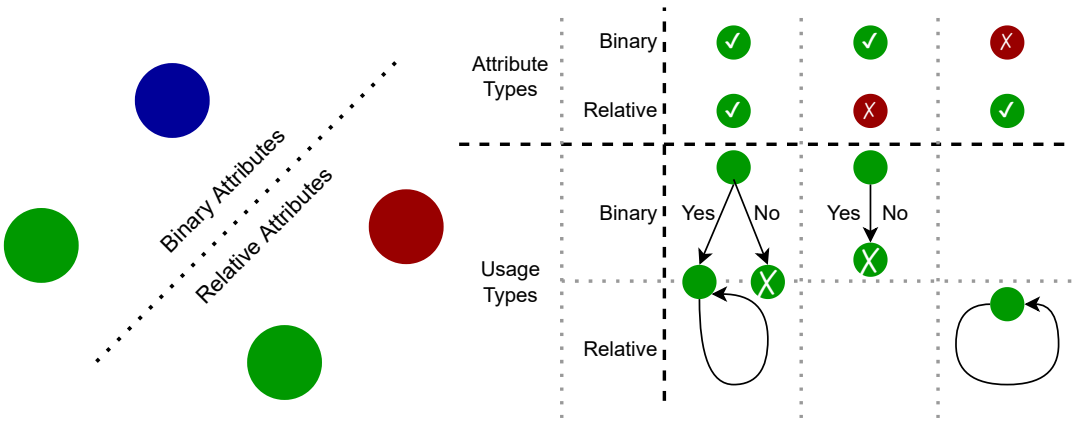
This strategy solves part of the attribute selection. Since the action chosen in each interaction is decided during the conversation, there is the possibility that in each of the interactions a different action selected, leading to the use of a mix of question types during the conversation.

Also, as different actions require different types of attributes - as seen on the definition of the Binary, Relative and Comparative Attribute Questions in Sections 3.3.1 to 3.3.3 - and there are two types of attributes - green is a binary and relative attribute, blue is a binary only attribute and red is a relative only attribute as shown in Figure 4.6a -, not being these two types of attributes mutually exclusive, i.e. one attribute can be of both types, there are some situations that need to be considered to properly enable the actions to be mixed within a conversation, as depicted in Figure 4.6b:

- Binary and Relative Attributes: the assistant needs to first know whether the user's target product has the specified attribute, by using the additional information given by the user on the Seed Query (Section 3.2.1) and/or by using a Binary Attribute Question (Section 3.3.1). If the user's target product has that attribute, then the assistant can endlessly use it to know how the target product relates to other available products by using the Relative and/or Comparative Attribute Questions (Sections 3.3.2 and 3.3.3), narrowing down the search space;

- Binary-only Attributes: used by the assistant only once to know whether the user's target product has the specified attribute, which can be gathered by using the additional information from the Seed Query (Section 3.2.1) or by using a Binary Attribute Question (Section 3.3.1);

- Relative-only Attributes: the assistant assumes the specified attribute is present in all available products and uses the attribute endlessly to know how the user's target product relates to other products by using Relative and/or Comparative Attribute Questions (Sections 3.3.2 and 3.3.3).



(a) Attributes Types

(b) Attributes Usage

Figure 4.6: Catalogue Attributes

5

Experiments and Results

In this chapter we present the work performed for the evaluation of the behaviour of the proposed framework. This includes the results obtained using different scenarios, a set of baselines and the different behaviours of the assistant (used for the mix of actions/question types).

5.1 Experimental Setup

To evaluate the performance of the assistant as well as the performance of each of the actions being used in each interaction of the conversation, it is useful to have a dataset which includes both the utterances between the user and the assistant and a catalogue with products, which must have some related information about each of the products.

5.1.1 Interaction Simulation

From our search for datasets used in related works, we did not find any that contained utterances of the interactions between the user and the assistant. To overcome this challenge, as mentioned before in Section 3.1, we developed a system that simulates these interactions, allowing us to use a dataset only containing products.

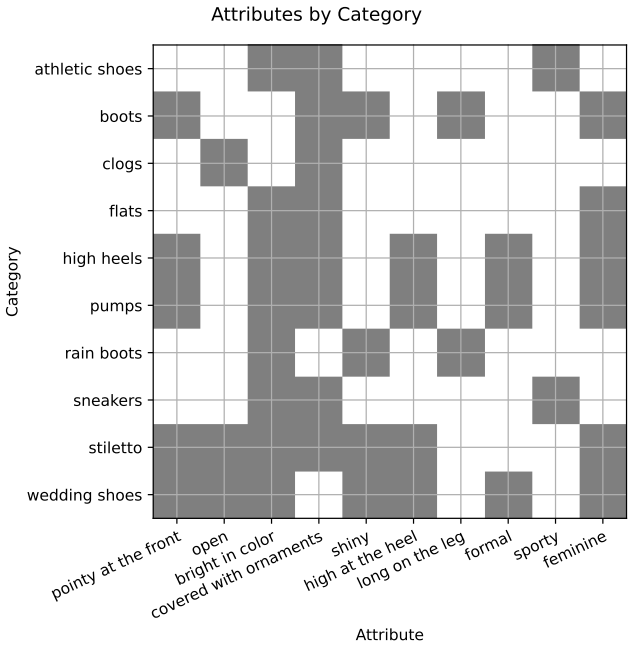
5.1.2 Catalogue

As discussed in Section 2.8.2, the catalogue used is a public catalogue available here, also used by [24, 25], and is composed of around 14k shoe products with their associated information. It is composed of 10 categories (Figure 5.1b) - *athletic shoes, boots, clogs, flats, high heels, pumps, rain boots, sneakers, stiletto and wedding shoes* - and 10 attributes - *pointy at the front, open, bright in color, covered with ornaments, shiny, high at the heel, long on the leg, formal, sporty and feminine*.

Each product has an associated category - binary attribute with 10 possible values - , which has some attributes associated with (Figure 5.1a) - binary attribute with 2 possible values. Also, for each attribute, there is a relative strength scale (Figure 5.1c, as defined

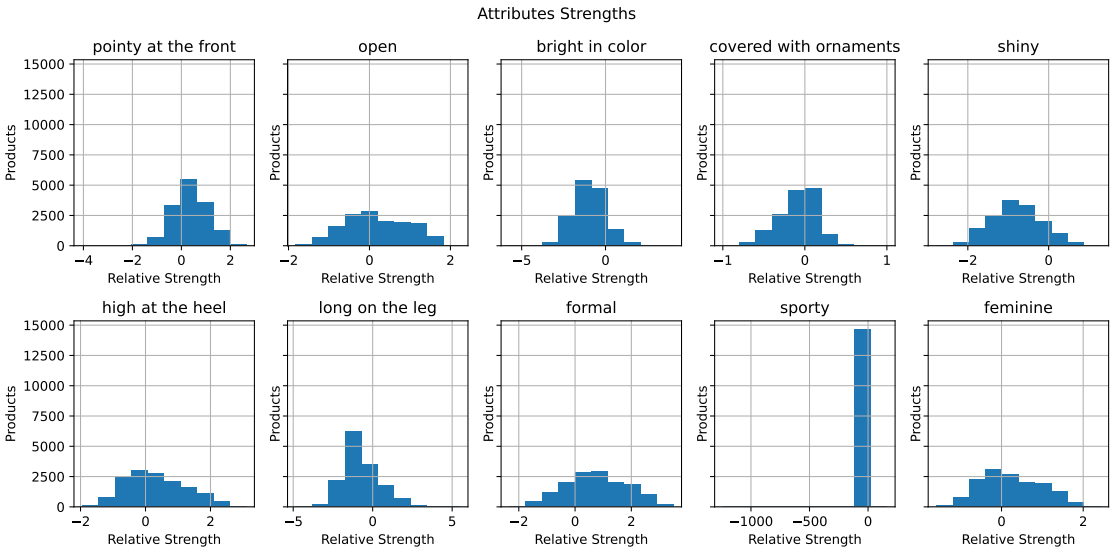
in Section 2.3) for its presence on each product, i.e. if it has more or less of the respective property.

Furthermore, each product has associated an embedding, which is a vector representation of each product within a n-dimensional space used for representing the products, independently of the available attributes in the catalogue.



(a) Attributes by Category

(b) Categories Examples



(c) Attributes Strengths

Figure 5.1: Catalogue Overview

5.1.3 Protocol and Metrics

For the evaluation itself of the assistant, where the protocol values are defined in Table 5.1, the conversation is tested with random products over a universe of several extracted products from the dataset, being 100 from each category. With each of the tested products, the conversation simulation is repeated some times, getting after each interaction between the user and the assistant a list of recommended products. The evaluation results are an average of the results over the various tested products, where for each product it is also the average over some repetitions.

Additionally, the conversations length varies since on the set of baselines there is no need for having longer conversations, while on the assistant behaviours, for the assistant to learn what is good and not good, we decided to let him run for some more interactions. Also, for having a closer to reality estimation of our measured metrics, we increased our number of products in the validation and testing set to 100, while we decreased the repetition times of each of them, not to impact the training time too much.

Protocol Values	Set of Baselines	Assistant Behaviours
Products Universe	1000	
Tested Products	20	100
Simulation Repetition	100	4
Recommended Products	3	
Number of Interactions	10	20

Table 5.1: Protocol Values

The results include the discussion of the metrics presented in Table 5.2:

Set of Baselines aka individual actions	Assistant Behaviours aka mix of actions
• Percentage of target products found after each interaction	• Number of mean, minimum, maximum, 3 minimum and 3 maximum interactions
• Number of products available after each interaction	• Question Types usage
• Number of interactions a conversation takes to find the target product	• Estimated time consumed

Table 5.2: Metrics

In order to evaluate each of the cases - set of baselines and assistant behaviours - we defined different metrics for each of them:

- **Set of Baselines**: this case uses only a single action per conversation, thus it does not require metrics for comparing the usage of different types of questions;
- **Assistant Behaviours**: this case is able to use any of the actions in each interaction of the conversation, thus can use a mix of actions, and therefore requires metrics for

comparing the usage of different types of questions. Also, the metrics related to the number of interactions are directly comparable to the various metrics logged for the set of baselines.

5.2 Results and Discussion

5.2.1 Overview

With the final goal of understanding the impact of the behaviours the assistant supports, it is important to start by evaluating some possible different **scenarios** he can be run on:

- **Seed Query**: for comparing the scenarios of using or not the seed query, which contains some initial hints of the user's target product;
- **Attributes Schedule**: for comparing the scenarios of using each of the Attribute Selector's strategies, round-robin and random, as specified in Section 4.2;

Then evaluating a **set of baselines** which evaluates the actions individually:

- **Binary Attribute Question**: for analysing the assistant action relative to the Binary Attribute Question with multiple catalogue sizes, as specified in Section 3.3.1;
- **Relative Attribute Question**: for analysing the assistant action relative to the Relative Attribute Question with multiple catalogue sizes, as specified in Section 3.3.2;
- **Comparative Attribute Question**: for analysing the assistant action relative to the Comparative Attribute Question with multiple catalogue sizes, as specified in Section 3.3.3;
- **Bigger Catalogue**: for analysing how the assistant adapts when presented with bigger catalogue sizes.

And finally the evaluation of the different **behaviours of the assistant** which is capable of using every available action:

- **Sanity Check**: for analysing the assistant behaviour when only forcing him to learn to associate a single value to the various actions, as specified in Section 4.1.2.1;
- **Minimizing Interactions**: for analysing the assistant behaviour when forcing him to learn to use as less interactions as possible, as specified in Section 4.1.2.2;
- **Minimizing Repetitiveness**: for analysing the assistant behaviour when forcing him to learn to vary in the use of actions as much as possible, as specified in Section 4.1.2.3;
- **Minimizing Repetitiveness & Time**: for analysing the assistant behaviour when forcing him to learn to vary in the use of actions as much as possible but also taking attention to the time each action takes to get feedback from the user, as specified in Section 4.1.2.4.

To finish, we also analyse how slight **modifications to the architecture** of the model impact its performance, varying both:

- **Depth**: by varying the number of hidden layers;
- **Width**: by varying the number of input units of the hidden layers.

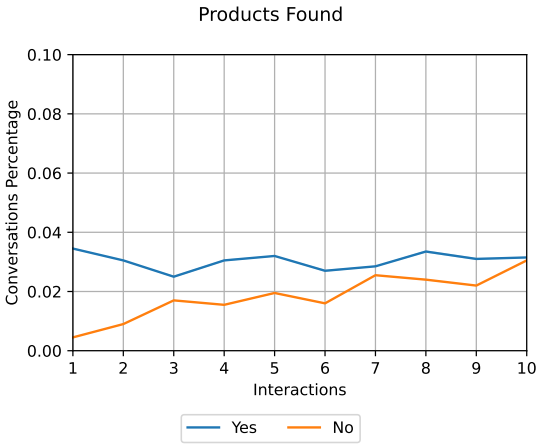
5.2.2 Scenarios

In order to get the best out of the assistant behaviours presented above, it is essential to first know which scenario works best in the context of the problem from a set of scenarios.

5.2.2.1 Seed Query

The first scenario used is testing the assistant with and without the seed query. For this, only the Binary Attribute Question can be used with a seed query containing only the category of the product, because there are only 10 attributes in the catalogue and the conversation performs 10 interactions with the user, not repeating any binary attribute (as it does not help in getting more information about the user's target product).

Analysing Figure 5.2, it is possible to verify that, although the percentage of products found is low with or without the use of the seed query, when it is used the percentage starts higher from the beginning (first interaction), thus improving the performance of the assistant from the beginning of the conversation.



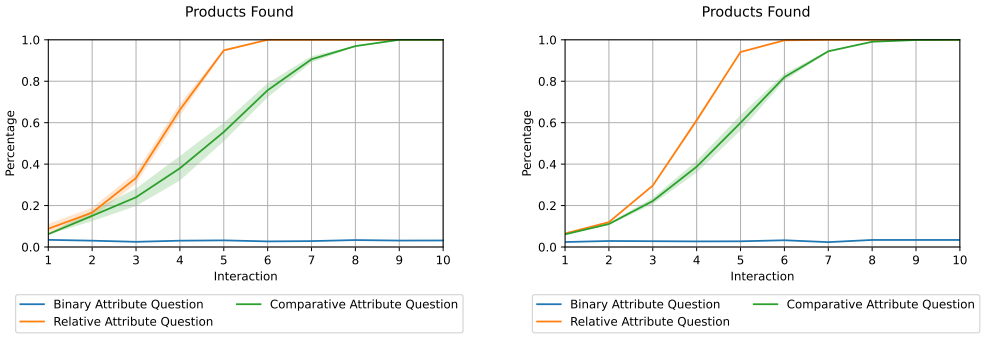
Comparison of percentage of conversations where the target product was found using and not using the Seed Query

Figure 5.2: Seed Query Scenario

5.2.2.2 Attributes Schedule

Another interesting scenario to check is how the assistant is affected by the variation on how the attributes are selected for the clarifying questions.

As shown by comparing each of the question types in Figures 5.3a and 5.3b, the conversation was not much affected since there are just a few binary attributes known, gathered by the Seed Query, in the Relative and Comparative Attribute Questions.



(a) Percentage of conversations where the target product was found after each interaction with round robin schedule

(b) Percentage of conversations where the target product was found after each interaction with random schedule

Figure 5.3: Attributes Schedule Scenario

Although in this context there is not much difference in the performance, we think it is preferable to use a round-robin schedule since it ensures that the attributes are not always repeated, thus helping in getting varied information from the user.

Overall, the scenario that best fits and helps the development of the assistant is using the Seed Query - as already used when comparing the Attributes Schedule - and using a round-robin schedule for the attribute selection.

5.2.3 Set of Baselines

With the scenario chosen and set up, let's start by analysing each of the actions individually. Since for this analysis there is no need to use a variety of actions during a conversation, it won't be needed any additional decision component to decide which question type to use in the next interaction.

For an easier initial evaluation, the assistant is run in a smaller catalogue containing 1000 products, as indicated in the protocol values of Section 5.1.3, and later compared to when there are 5000 and 10000 total products in the catalogue.

5.2.3.1 Binary Attribute Question

The first action tested is the usage of the Binary Attribute Question with a Seed Query like the one used in Section 5.2.2.1, which only included the category of the user's target product.

As mentioned in Section 5.2.2.1 and as we can observe from the left plot of Figure 5.4, it is possible to confirm that the percentage of products found is very low, no matter how many interactions are performed between the user and the assistant. This is due to the fact

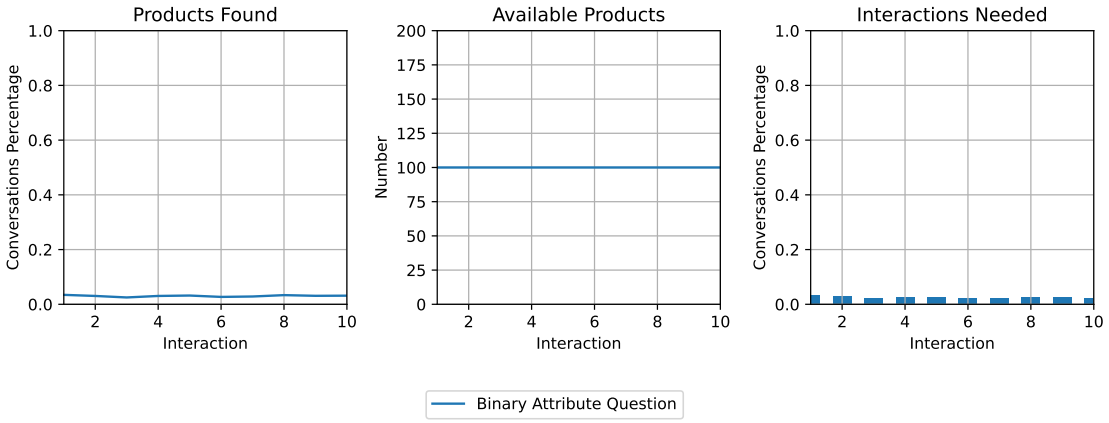


Figure 5.4: Binary Action Baseline

that every attribute is associated with at least one category and, as such, keeping asking the user if his target product has an attribute is not helpful.

Also, as can be seen in the middle plot of Figure 5.4, the restrictions gathered by this question type are only able to extract the products of the mentioned category, each of which includes 100 products. This then leads the extraction of the 3 recommended products out of the 100 products to not be efficient, getting low scores.

To further confirm this assessment, the right plot of Figure 5.4 shows that there is not a good indication for the number of interactions needed to find the user's target product, not being very reliable to be used standalone.

5.2.3.2 Relative Attribute Question

Another action tested is the usage of the Relative Attribute Question, which uses a Seed Query with a category and two attributes. The category helps in discarding a lot of products from the beginning, as seen in Section 5.2.2.1, and the attributes are needed for the assistant to know at least one attribute. Revising Figure 4.6b, at least one attribute is needed because the assistant needs to use a relative attribute and in the catalogue all the attributes are both binary and relative at the same time, so, the assistant needs to know that the attribute exists in the user's target product to be able to use it.

As shown in the left plot of Figure 5.5, the percentage of products found grows rapidly with a few number of interactions. This is due to the fact that this action allows the user to compare his target product with a referenced product according to the indicated attribute by telling if his product has More, Less or Equally of that attribute's value. This comparison allows the assistant to restrict the range of available products after each interaction, as can be seen in the middle plot of Figure 5.5. This restricted range goes narrower even faster if the user's target product has the same strength as the referenced product.

Contrary to the Binary Attribute Question, the right plot of Figure 5.5 shows that for this question type it is enough to perform 4 interactions in order to find the user's target

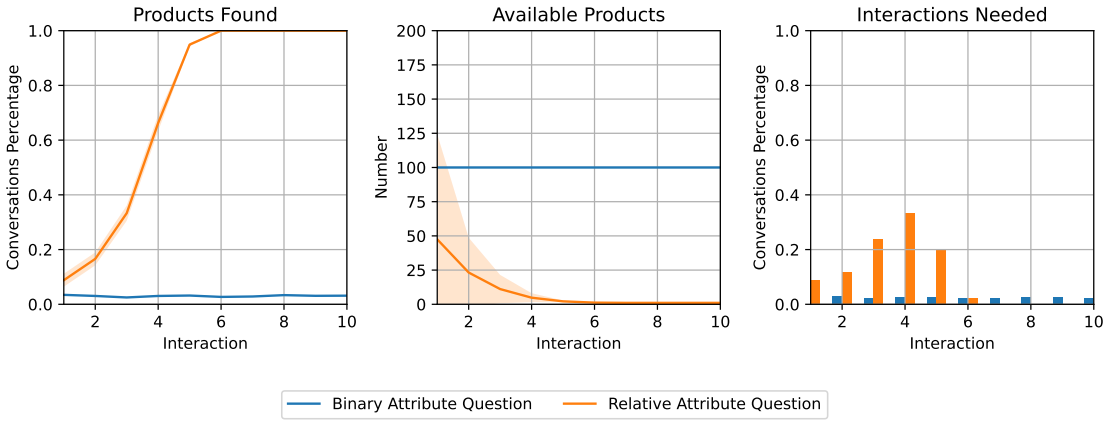


Figure 5.5: Relative Action Baseline

product in around 40% of the times.

5.2.3.3 Comparative Attribute Question

The last action tested is using the Comparative Attribute Question, which also uses a Seed Query with a category and two attributes, for the same reasons as the Relative Attribute Question presented above.

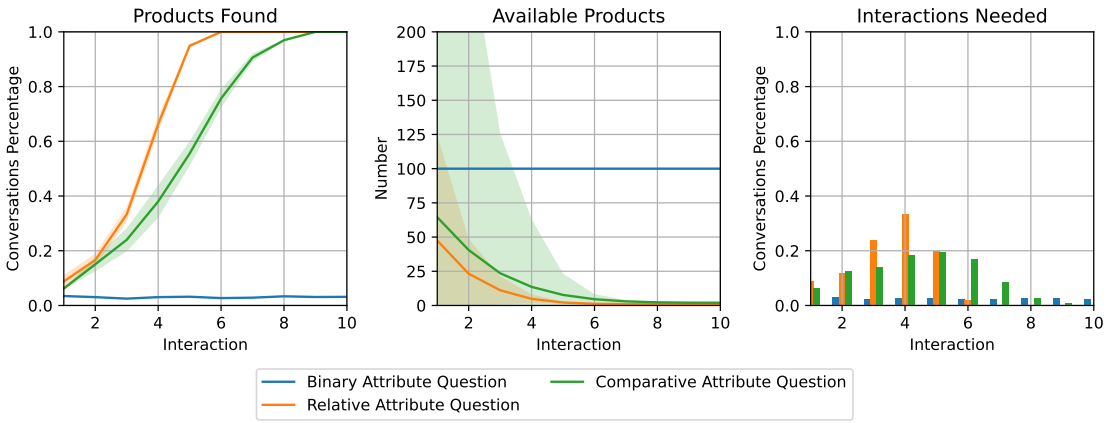


Figure 5.6: Comparative Action Baseline

As shown in the left plot of Figure 5.6, similarly to the Relative, this action starts with a similar score but after a few interactions it scores lower than the Relative. This is due to the fact that, since it allows the user to choose the product (from the list of referenced products) which has the attribute strength closest to his target product according to the mentioned attribute, the assistant is able to restrict the range of available products (while the Relative is able to restrict the range to a single value when the referenced product has the same value as the user's target product). Although not as much as the Relative enables to, but allows the user to give a greater feedback about where in the relative strength scale his product is situated.

Looking deeper to the ranges of available products (middle plot of Figure 5.6), just like the Relative, the number of products available goes down quickly, although not as fast as the Relative. Analysing the right plot of Figure 5.6, this action needs just a few more interactions than the Relative to be able to find the user's target product. These two conclusions are due to the difference in the processing of the ranges between the Relative and the Comparative Attribute Questions as described above.

5.2.3.4 Bigger Catalogue

With the set of baselines of the assistant evaluated compared with each other, let's see how they behave when the size of the catalogue is increased from 1000 total products to 5000 and 10000 total products, with 500 and 1000 from each of the categories, respectively. Since when the catalogue size is increased there are more products to search through, this means the assistant has a greater search space to deal with.

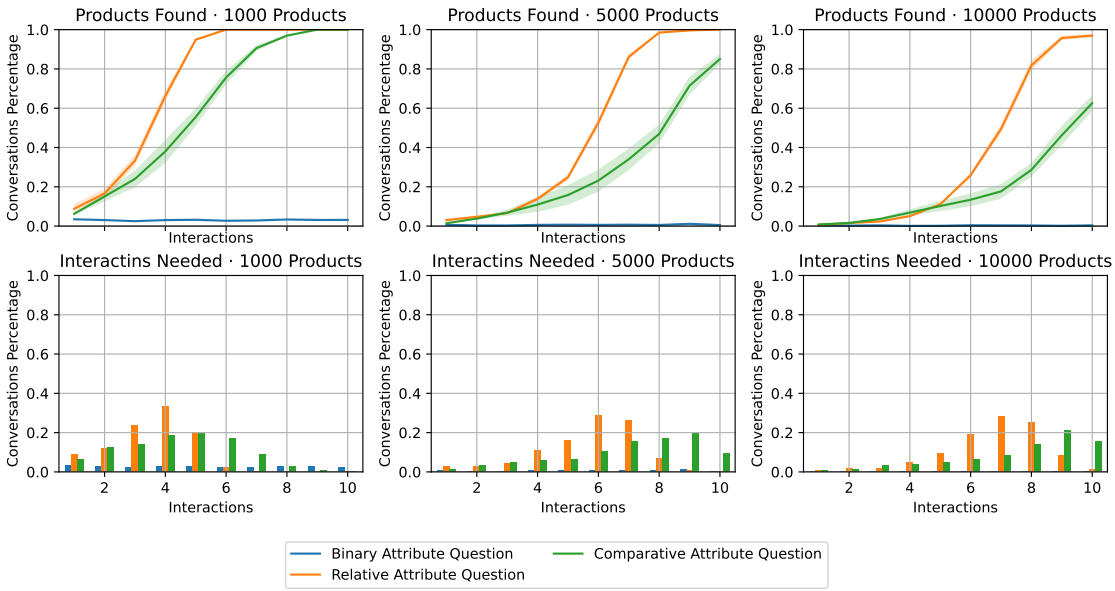


Figure 5.7: Bigger Catalogue Baseline

As can be confirmed in Figure 5.7, having a bigger search space turns it more difficult to extract the correct product using a list of only 3 products out of all of them (with the highest scores), leading the actions to take some more interactions to get the same scores. Also, the Relative and Comparative Attribute Questions are very useful to fine tune the user's search, since they are able to find the user target product fast in the various conversations.

5.2.4 Assistant Behaviours

Having the right scenario and each of the actions analysed, we can now look to the development of each of the behaviours supported by the assistant, which is able to use all of the available actions, being able to use any of them in each interaction of the conversation.

Testing these mix possibilities by hand is time consuming, inefficient and not scalable, being this thesis focused on getting the Question Type Selector automated and optimized according to the previous information gathered from the user, and with the goal of maximizing the percentage of products found while minimizing the number of interactions needed to find. This is achieved by using deep reinforcement learning, as proposed by [24, 25] in a similar context, but we further optimize the assistant to get different behaviors according to different requirements as shown in Sections 5.2.4.2 to 5.2.4.4 and later summarized in Figure 5.15.

For the development of the behaviours of the assistant, a sample architecture was used as shown in Figure 5.8, which is an adaptation from the architecture used by [24, 25], as explained in Section 4.1.1.

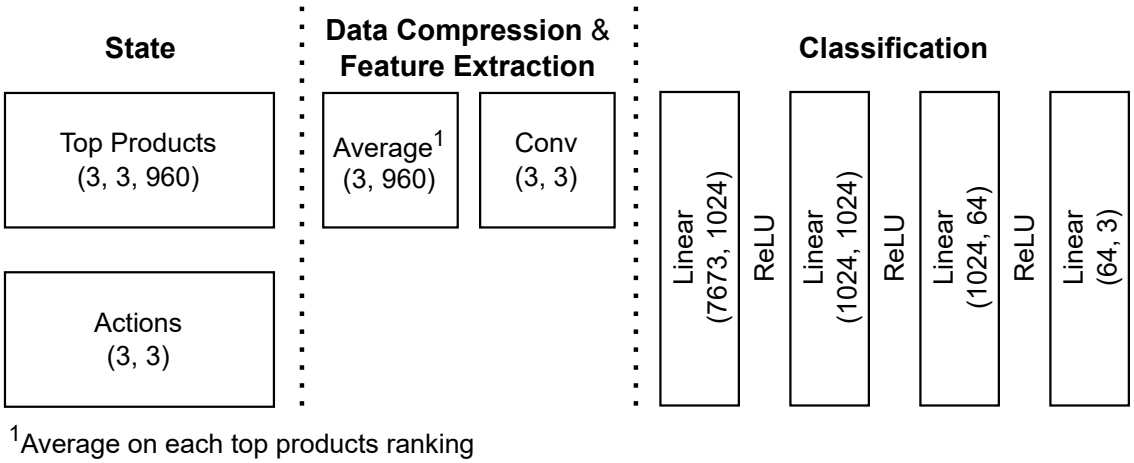


Figure 5.8: Initial Model Architecture

Additionally, it is important to revise , in Table 5.3, the different feedback times each of the actions (the question types) require to get information from the user, as they will be needed for the analysis of the various behaviours.

Estimated Durations	Binary	Relative	Comparative
Units of time	2	5	10

Table 5.3: Actions Estimated Durations Revisited

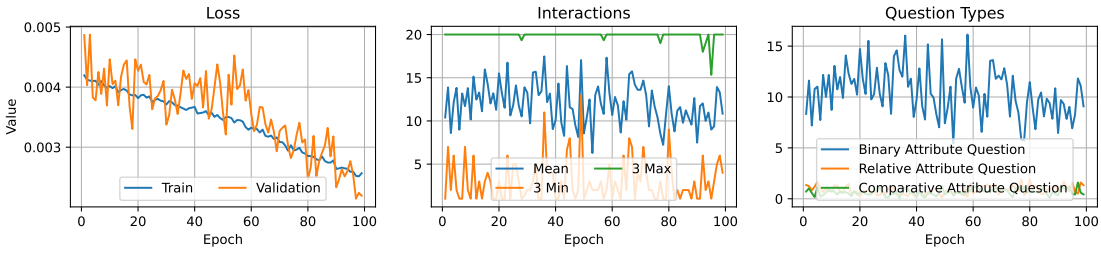
5.2.4.1 Sanity Check

As specified in Section 4.1.2.1, this simple behaviour serves as a check up for the training and environment set up. Therefore, no special and well thought reward was needed, so we associated every available action with a 0 value, as suggested in Equation (5.1).

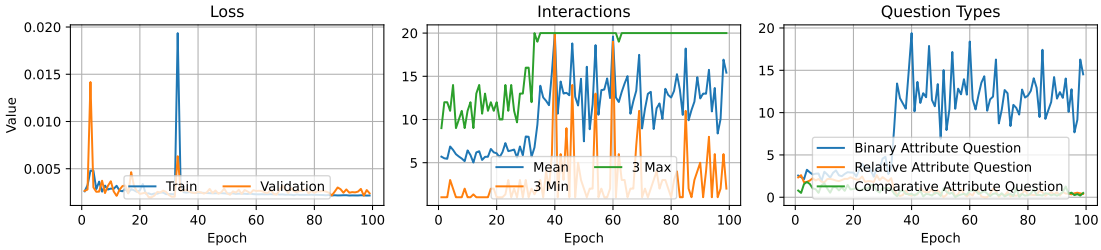
$$reward = 0 \tag{5.1}$$

With this simple reward function, we spotted some problems and difficulties in the implementation of the deep reinforcement learning environment and architecture, defined in Section 3.1.3, which we enumerate:

Initial Architecture although we mentioned above that we adapted the architecture of [24, 25] to our problem context, the agent was not learning anything useful as can be seen in Figure 5.9a, where the actions chosen started almost identical to where they should stabilize the learning process, meaning it is not learning anything. After confirming the metrics were correctly set up, we noticed that in the classification part of the model there were too few non-linear transformations to the data - only 1 hidden layer - , leading to the agent not being able to learn anything from it. Therefore, to solve this issue, we added an additional hidden layer to the architecture, leading to the agent to be able to start learning as expected, as can be verified in Figure 5.9b, which lead the assistant to have a different behaviour as the actions start by being chosen more evenly and after a few epochs they start to get closer to the value where they stabilize.



(a) Not Learning



(b) Learning

Figure 5.9: Initial Architecture

Environments Isolation in order to simulate the conversation between the user and the assistant inside the learning architecture presented in Section 3.1.3, we had to further isolate the System component by instantiating on System component per data set. This was a huge problem since by using the same instance for every data set, they would be acting together and not separately, leading to the metrics measured by the System not accurate towards their real behaviour. Additionally, since the reward schemas will require some metrics measured from the conversation history to get the expected assistant behaviours, it would also influence those results;

Training Time another problem we faced was the long time needed to train, but specially the time needed for each validation set to run after each training epoch. Initially we ran it as we did for the set of baselines, using 100 repetitions for each of the 20 products, but it was taking too long to get results, so we decided to invert the amounts to get a more realistic and even comparison, by adding more products to this set while decreasing the repetitions of each of them.

With these initial challenges addressed, we can now analyse this behaviour in detail. As can be visualized in Figure 5.10, with no restrictions the assistant learns to have conversations with a lot of interactions on average, being this value distant from the minimum and maximum, specially from the minimum values. With the increase of the catalogue size, the number of mean interactions increases, almost hitting the maximum limit (20). This has a worse performance than using each of the actions individually, as can be verified in Figure 5.7, where at around 6 to 8 interactions they were able to find the user's desired product, when dealing with a 1000 products catalogue size.

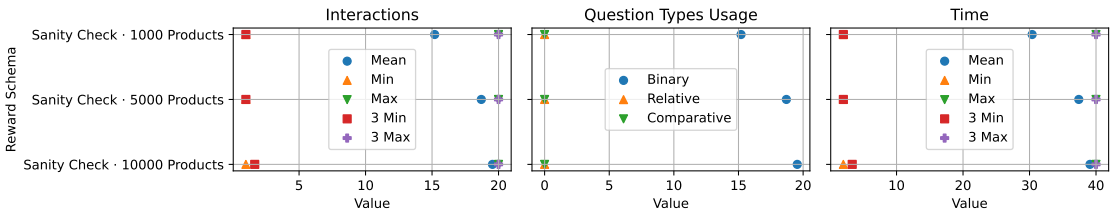


Figure 5.10: Sanity Check Behaviour

Although the mean number of interactions is high, the assistant does not consume much time and tends to only use the Binary Attribute Question, which is the first action available on the output of the model, proving our expected performance in Section 4.1.2.1.

5.2.4.2 Minimizing Interactions

From these initial results, as discussed in Section 4.1.2.2, we intend to make the agent to learn to use as less interactions as possible, so the assistant can achieve the main goal of this thesis work. For this, the schema should have a great attention on the number of interactions, as happens on the Minimizing Interactions rewards presented in Figure 5.12.

Although it seems pretty forward to develop, we noticed another learning aspect could be improved:

Convergence as discussed in Section 4.1.3, specifically in the Stopping Criteria specification, in this types of problems, which make use of deep reinforcement learning, it is a positive aspect to overfit, leading the agent to learn very well how to perform tasks in this specific environment. To check that the agent has learned for a sufficient amount of epochs, we first ran this reward schema using a large amount of epochs, like 100 of them, and then analysed where our measured metrics started stabilizing their values.

Analysing Figure 5.11 we noticed that the model did not need more than 20 epochs to learn and stabilize successfully - specially visible in the evolution of the Question Types - , although the authors of [24, 25] use 30 epochs and the authors of [4] use 15 epochs;

Note: the loss value is the difference between the predicted reward by the model for each action and its expected value, and not if the model chose the correct action directly.

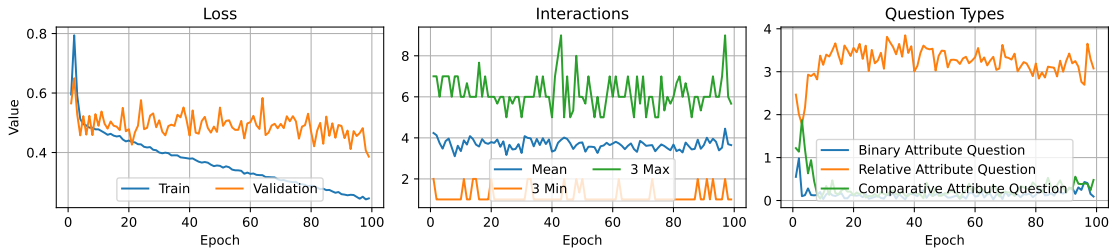


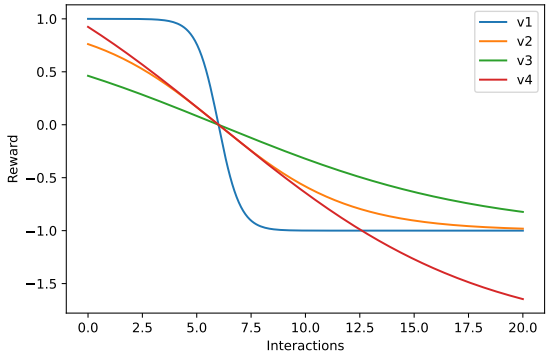
Figure 5.11: Training Convergence

To accomplish this goal, we start by using a function which starts with a high value and decreases over the number of interactions used. To accomplish this reward function behaviour, we make use of a $tanh$ function as specified in Equation (5.2) and plotted in Figure 5.12a.

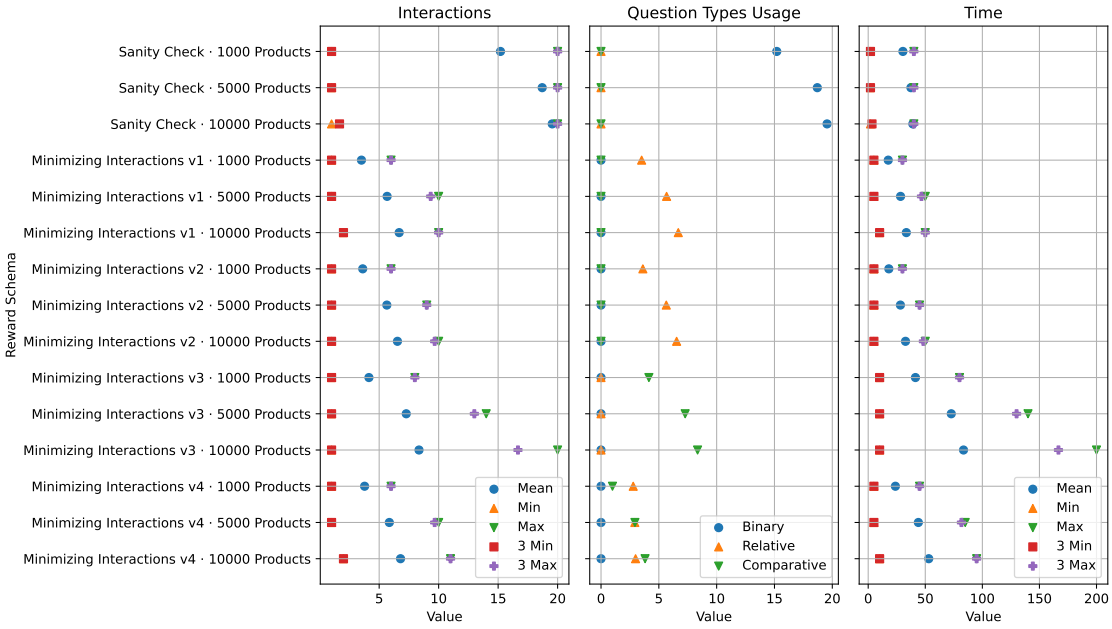
$$v1 = -\tanh(\#interactions - 6) \tag{5.2}$$

With this version, we noticed a huge reduction in both the mean number and the maximum number of interactions. With the use of a bigger catalogue, these values grow a bit, but never going beyond 10 interactions. Comparing with the actions individually (as depicted in Figure 5.7), with a catalogue size of 1000 products, the assistant is able to get the user's product within less interactions.

Although this version presents good results, we want to make the shorter than 4 and longer than 8 conversations to have a bigger difference in terms of reward for every amount of interactions used, so the assistant actually learns to minimize the number of



(a) Graph



(b) Metrics

Figure 5.12: Minimizing Interactions Rewards

interactions. With this in mind, the schema version presented in Equation (5.3) and also plotted in Figure 5.12a is used.

$$v2 = -\tanh\left(\frac{\#interactions - 6}{6}\right) \tag{5.3}$$

With this modification to the reward schema, analysing Figure 5.12b we noticed a slight improvement in the extremes of the interactions metrics (mean, minimums and maximums), making them have slightly lower values. Having this improvement, we want to check how going further in the steepness of the reward function would affect the results by trying two more versions presented in Equations (5.4) and (5.5).

$$v3 = -\tanh\left(\frac{\#interactions - 6}{12}\right) \tag{5.4}$$

$$v4 = -2 \tanh\left(\frac{\#interactions - 6}{12}\right) \tag{5.5}$$

Both of these versions ($v3$ and $v4$) have the worst results, although $v4$ one is close to the second version (Equation (5.3)), but being worse when the catalogue size is increased.

From these comparisons, we think the second version ($v2$) is the best one - specified again in Equation (5.6) - , since it decays the reward value gradually but not too steep, helping the assistant to learn to use as less interactions as possible.

$$reward = -\tanh\left(\frac{\#interactions - 6}{6}\right) \tag{5.6}$$

This schema further confirms our expected performance from Section 5.2.4.2, with which the assistant tends, on average, to consume even less time than with no restrictions on every catalogue size. Also, it learns to only use the Relative Attribute Question. A possible reason for this choice, as discussed previously, is because this action is capable of restricting the search space to a single value in the specified attribute axis when the referenced product matches the target product value for that attribute.

Although with the reward schema the main goal of this thesis is achieved as well as the behaviour explored in the work of the authors of [24, 25], we further investigate and explore additional behaviours in the following Sections 5.2.4.3 and 5.2.4.4. These additional behaviours enable the assistant to be used in several contexts and situations which might have different requirements, and might also help the assistant avoiding local minimums in the search for the user's target product.

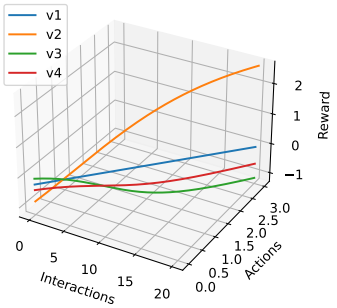
5.2.4.3 Minimizing Repetitiveness

With the previous modification to the reward schema, as mentioned in Section 5.2.4.3, by reducing the number of interactions, the assistant learned to use less time in average but tends to mostly use only one of the actions - in our case the Relative Attribute Question is always repeated essentially because of its capabilities to narrow down the search space ranges quickly.

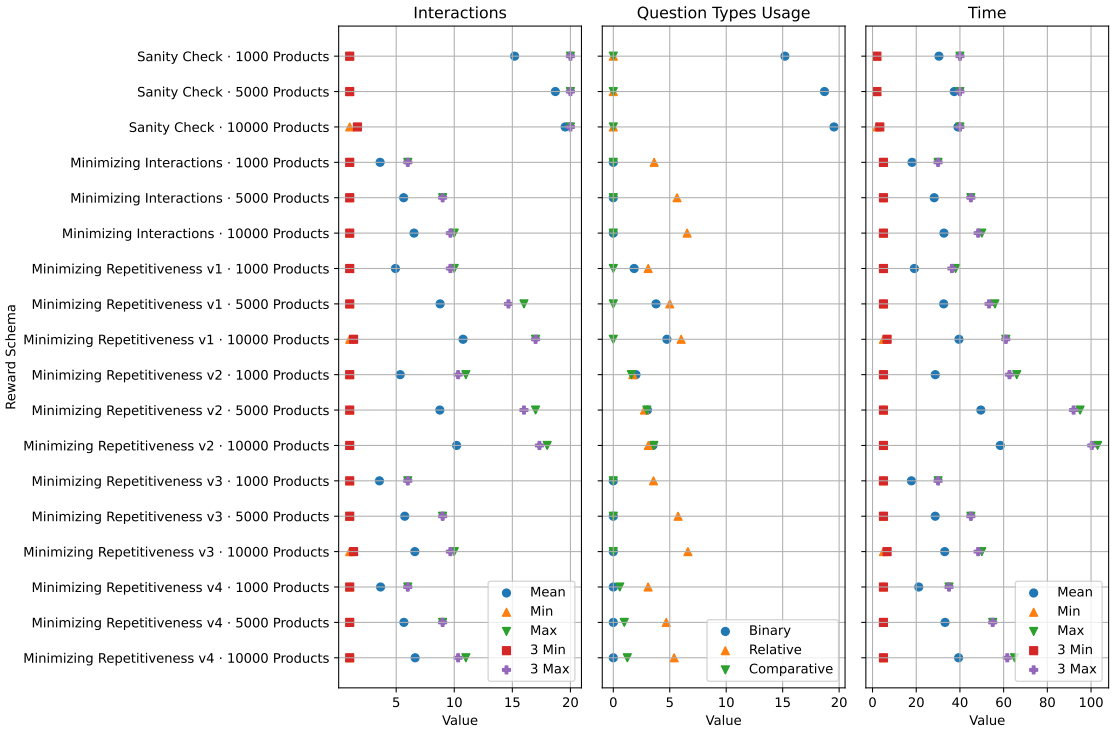
Being asking a type of question repeatedly can become annoying and/or be unnecessary at all. To overcome it, we want to instruct the assistant that varying the actions is a good thing, by making the reward function to have a great focus on the variety of the use of actions, as shown on the Minimizing Repetitiveness rewards in Figure 5.13.

To accomplish this, we start by defining the reward function by also using a $tanh$ function, since it naturally decays over the input value, but also dependent on the number of different actions used during the conversation. Since in this problem we have 3 available actions, we defined the reward presented in Equation (5.7) and plotted in Figure 5.12a.

$$v1 = \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{5.7}$$



(a) Graph



(b) Metrics

Figure 5.13: Minimizing Repetitiveness Rewards

With this first version of the reward, we noticed an improvement over the variety of the actions chosen and consequently an increase on the number of interactions, since different actions have different restriction effects. The time consumed by the conversations also increased, but just a bit. When dealing with a bigger catalogue, the actions usage tends to maintain the proportion, while the number of interactions increased when compared to the previous behaviour (Minimizing Interactions), although the number of interactions is still smaller then with the Sanity Check behaviour. Although the number of interactions increased when the catalogue size is increased, from 5000 products to 10000 products, the increase is smaller then when it is increased from 1000 products to 5000 products, which might mean that this schema scales well - not linearly - with the size of the catalogue.

As we did for the previous behaviour, we want to try to make the reward function to decay steeper. For this, we used the reward presented in Equation (5.8).

$$v2 = 3 * \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{5.8}$$

With this update to the reward, the number of interactions did not suffer much differences comparing to $v1$, the time consumed increased a lot, but all the actions are now being used, which is our goal from this behaviour.

Even though this second version has great results regarding our ambition, we want to try some more alternatives, considering also the number of interactions taken to check if it could help in improving further the results: Equations (5.9) and (5.10).

$$v3 = -\tanh\left(\frac{\#interactions - 6}{6}\right) + \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{5.9}$$

$$v4 = -0.5 * \tanh\left(\frac{\#interactions - 6}{6}\right) + \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{5.10}$$

In both of these versions ($v3$ and $v4$) the number of interactions reduced comparing to the previous versions, the time also decreased, but the usage of actions is worse than any of the other versions ($v1$ and $v2$), which is a result of making the reward function dependent also on the interactions used in each conversation.

From these comparisons, we think the second one ($v2$) - specified again in Equation (5.11) - is the best one to achieve less repetitiveness of the different actions, although the time consumed during the conversations increased a lot.

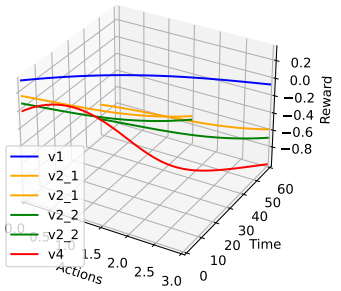
$$reward = 3 * \tanh\left(\frac{\#used - \#total - 0.5}{12}\right) \tag{5.11}$$

This schema confirms our expectations in performance from Section 4.1.2.3, with which the question types variety is great, with the drawback of taking more interactions and longer to find the user's target product. Also, comparing this behaviour with the use of each action separately, the assistant is capable of finding the user's target product within about the same interactions as using only the Relative Attribute Question in all of the catalogue sizes (1000, 5000 and 10000), but now not repeating that specific question type over and over.

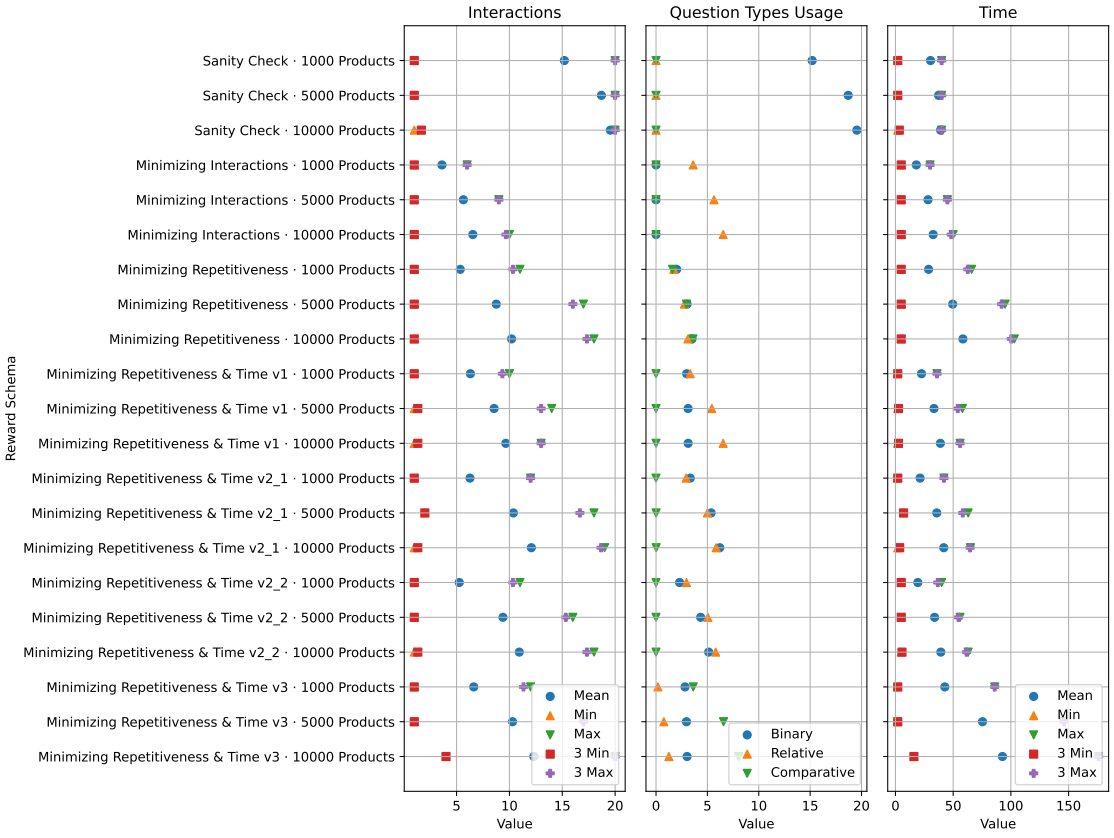
5.2.4.4 Minimizing Repetitiveness & Time

As stated in Section 4.1.2.4 and discussed in the above reward schema, avoiding repetitiveness is a great step towards to a more natural and varied conversation, but there may still happen the repetition of some of the actions, specially when there are not enough different question types comparing to the number of interactions needed for each conversation. To solve this additional challenge, we try three different types of solutions, all dependent on

the number of different actions chosen and on the time of the conversations, as shown in Figure 5.14.



(a) Graph



(b) Metrics

Figure 5.14: Minimizing Repetitiveness & Time Rewards

From the results gotten in versions $v3$ and $v4$ of the previous behaviour, the first type of solution $v1$ for this behaviour, defined in Equation (5.12), uses the same idea and makes a sum of both the number of different actions chosen and the time spent during the whole conversation. With this solution we can check that the time used during the conversation is short and uses two of the available actions. Interestingly, the Binary Attribute Question

is used always the same amount of times with every catalogue size, which means that after receiving this binary information from the user, the other type of question is repeated over and over.

$$reward = 0.5 * \left(3 * \tanh\left(\frac{\#used - \#total - 3 - 0.5}{12}\right)\right) - \tanh\left(\frac{0.2 * (\#time - 60)}{12}\right) \quad (5.12)$$

The second type of solution $v2\_1$, specified in Equation (5.13), makes the reward penalize longer conversations according to the number of different actions used, i.e. penalizes less if it is a short conversation and penalizes more otherwise. This improved the usage of question types in the sense that the ones that are used, are used evenly in number of times. This leads on longer conversations, but the time spent during the conversation is not affected much comparing to the previous solution type.

$$reward = -\tanh\left(\frac{0.2 * (\#time - 20)}{12}\right) if\ (\#actions) > \frac{3}{2} else - \tanh\left(\frac{0.2 * (\#time - 10)}{12}\right) \quad (5.13)$$

Even though this solution has quite good results, we want to try and minimize even more the number of interactions used with the reward $v2\_2$ presented in Equation (5.14), which only reduces the time constraint.

$$reward = -\tanh\left(\frac{0.2 * (\#time - 10)}{12}\right) if\ (\#actions) > \frac{3}{2} else - \tanh\left(\frac{0.2 * (\#time - 5)}{12}\right) \quad (5.14)$$

This improvement actually improved the number of interactions needed per conversation, while not affecting much the variety of actions used and the time consumed from the previous attempt (Equation (5.13)).

Lastly, we try another solution $v3$ which makes the penalization dependent on the trade-off of both the variety of actions taken and the time consumed, as presented in Equation (5.15).

$$reward = -\tanh\left(\frac{\frac{\#actions}{3} * (\#time - 10)}{12}\right) \quad (5.15)$$

This type of solution tends to use more interactions during the conversations and consume more time than any of the above two versions, but uses at least one time every action available.

With these comparisons, the best version to use depends on the specific requirements of each environment, since one tends to use all the question types but takes longer, the others two use much less time but do not tend to use every question type available and uses them in different manners. In terms of scalability with the size of the catalogue, the third version tends to present the worst performance, while the others two scale well -

from 5000 to 10000 products the difference is smaller than from 1000 to 5000 products in the catalogue.

5.2.4.5 Summary

Having explored all the reward function for each of the behaviours, we now summarize the best ones of each of them in Figure 5.15, excluding Sanity Check, which was used only to perform a check up on the implementation.

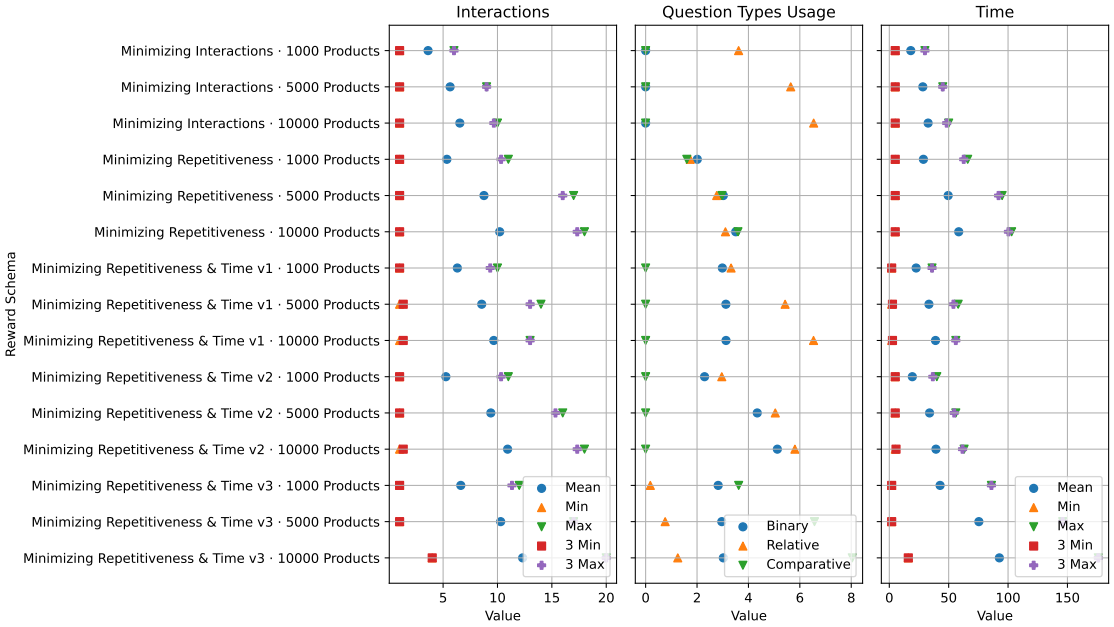
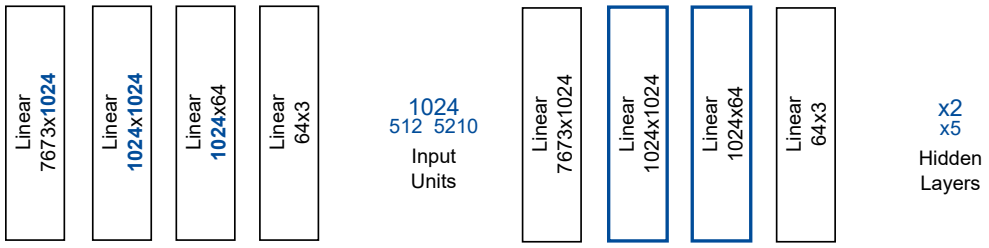


Figure 5.15: Behaviours Summary

5.2.5 Model Architectures

Having the versions of the various behaviours developed within the scenario specified and using a sample version of the model architecture, based on the one used by [24, 25] but adapted to the problem of this thesis (because of the context of the problem as explored in Section 4.1.1), it is interesting to assess the impact of some modifications to the assistant's neural network architecture in terms of the training success and regarding the final results. For this purpose, the classification part of the architecture - the *linear* layers - was modified in two different ways:

- **Depth Variation**: by varying the number of hidden layers within 2 and 5 hidden layers, as suggested by Figure 5.16b;
- **Width Variation**: by varying the width of the hidden layers within 512, 1024 and 5210 input units, as suggested by Figure 5.16a.



(a) Width Variation (b) Depth Variation

Figure 5.16: Model Architectures Variations

With the results from Figure 5.20, which are applied to the Minimizing Interactions behaviour, we can see that when the hidden layers have a large amount of input units or has a large amount of input units and several hidden layers, it presents some difficulty in learning correctly. For a deeper understanding of what is happening, we now compare the three different training versions mentioned, where the first one serves as comparison:

Depth 2 · Width 512 Figure 5.17 represents the normal training of the assistant, where the Loss decays over time and the Question Types start evenly used and end up with the desired usage;

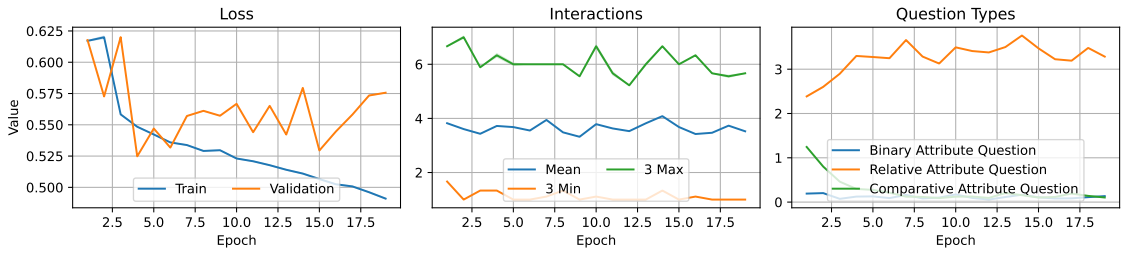


Figure 5.17: Model Architecture Results with Depth 2 · Width 512

Depth 2 · Width 5210 Figure 5.19 represents the version of the assistant that has some difficulty in learning to minimize the number of interactions, but is capable of learning something. The Loss also decays over time and the Question Types usage also starts from even usage to almost the desired usage, which is using only the Relative Attribute Question, but the learning is not very consisted as can be confirmed by the variance (colored areas around each metric curve) of each metric;

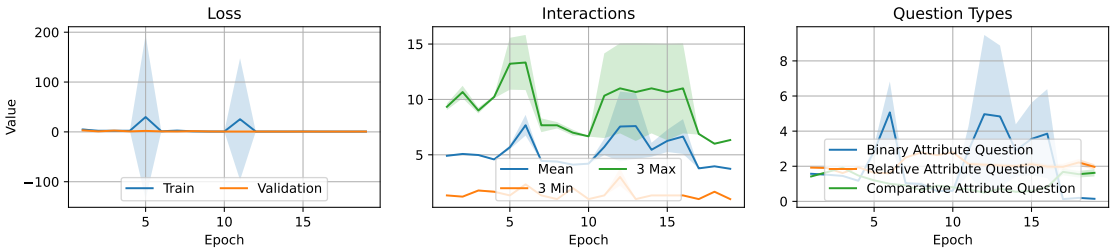


Figure 5.18: Model Architecture Results with Depth 2 · Width 5210

Depth 5 · Width 5210 Figure 5.19 represents the version of the assistant which is not able to learn anything useful because of the too large neural network architecture, leading in a lot of neurons to train and possibly not enough amount of data to train with. The Loss clearly shows this issue by being increasing over time and the Question Types usage has the highest variance and no difference in the usage of the actions over training time.

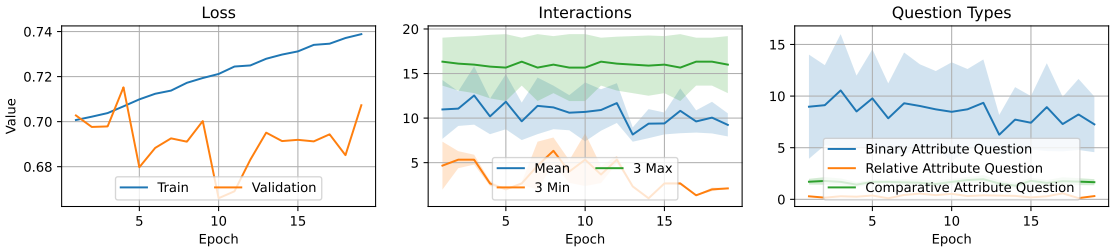


Figure 5.19: Model Architecture Results with Depth 5 · Width 5210

The remaining variations to the model architecture behave similar to each other. With this, we choose to use the model with 1 hidden layer (3 total *linear* layers in the classification part) and 1024 input units on the hidden layers since it presents a good trade-off: already trained with every assistant behaviour developed and the results are similar to other variants.

Having presented all the work performed during this thesis work, we present some discussions regarding the real usage and how this work can be used and deployed in projects.

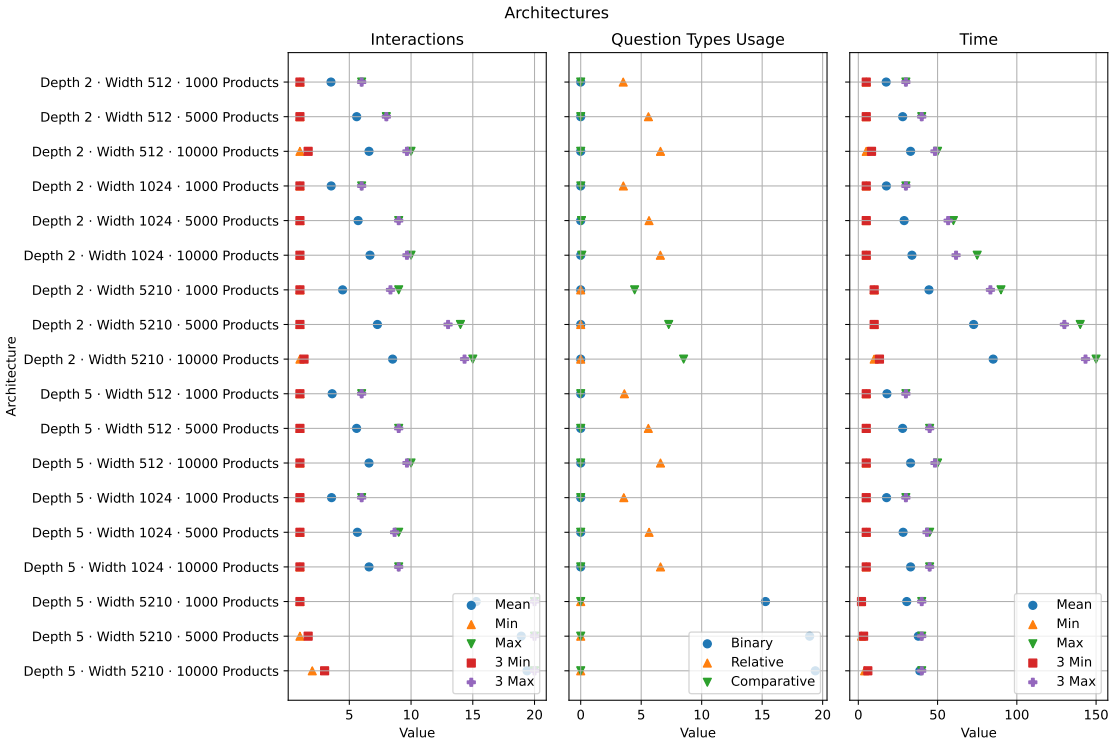


Figure 5.20: Model Architectures Results

6

Conclusions and Future Work

In this last chapter of this thesis, having presented all the work performed during the development of this thesis, we present some discussions regarding the real usage and how this work can be integrated in projects. Also, we discuss future work in order to further improve and extend this framework.

6.1 Conclusions

6.1.1 Overview

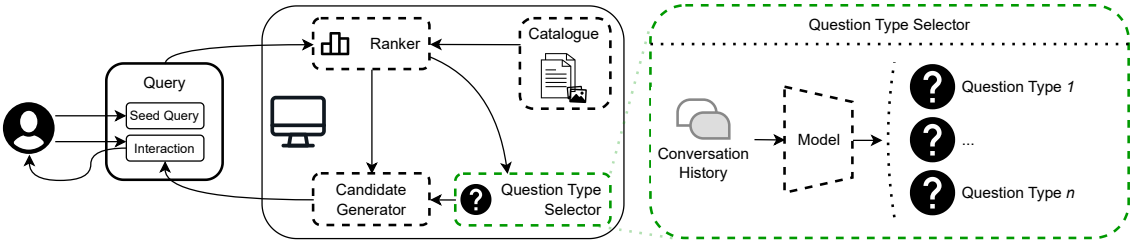


Figure 6.1: Framework Architecture Revisited

This thesis focuses on developing a framework, revisited in Figure 6.1, capable of improving conversational assistants towards getting the user's target product by asking a sequence of clarifying questions and giving recommendations. The clarifying questions are chosen in a real-time manner, which means that our model needs to receive the history of latest three top products lists and their corresponding question types.

The model of the assistant is used for knowing which of the question types to use and then the assistant generates the clarifying question with a pre-defined template as well as chooses the products to recommend, if needed. The clarifying question is presented to the user and then the assistant gathers his response and processes it in order to update the products ranking as well as the conversation history.

6.1.2 Real Experience

When using this assistant in a real world environment, we are expecting that this framework module is called when the assistant knows or has some confidence about the user knowledge of what he is looking for. Otherwise, accumulating the restrictions provided by the user can turn in not recommending what he looks after.

6.1.3 Integration

To integrate this framework module in a project is totally possible, as long as this module is able to access:

Relevant Conversation History : the conversation history related to the current recommendation only; and

Generate Questions : is able to generate questions for presenting to the user, i.e. is able to generate its own questions using a pre-defined template; and

Catalogue : is also able to access the catalogue information presented in Section 3.1, which must include: photo, category, binary and relative attributes and an embedding of each product.

6.1.4 Expected Improvements

With the integration of this framework module in a project which might have an existent agent, we expect this module to help the user in getting his desired product as soon as possible, depending on the requirements and expected behaviour of the project in which its being integrated, which might be:

Minimizing Interactions for teaching the assistant to use as less interactions as possible during each conversation;

Minimizing Repetitiveness for teaching the assistant to repeat as less as possible each of the available actions during each conversation;

Minimizing Repetitiveness & Time for teaching the assistant to repeat as less as possible each of the available actions during each conversation, while keeping attention to the each of the used actions needs to get feedback from the user. This can be decomposed in three versions:

Version 1 tends to use most actions, but does not use them evenly and uses the most basic action (Binary Attribute Question) the same number of times independently of the size of the catalogue;

Version 2 tends to use most actions and uses them more evenly;

Version 3 tends to use every available action, but does not use them evenly, requires more interactions and consumes more time.

Note that when comparing these behaviours with the work presented by [24, 25], we offer more ways to mix the various actions during the conversation with the user, enabling the assistant to behave in different behaviours: more or less interactions, more or less variety of asked questions, and more or less consumed time.

6.2 Future Work

This framework, which can be used as a module, still has room for improvements and extensions:

Question Types : behind the question types being used by the assistant, we use the Relative Feedback (Section 2.3) as their main logic except for the Binary Attribute Question.

As explored in Like/dislike Feedback (Section 2.2), there may be other types of feedback and interactions that may be useful to enrich the assistant's performance and the interactive conversational recommendation experience.

Question Type and Attribute Selection : the question type and attribute selection are separated from each other, being the selection of the question type made by the model inside the Question Type Selector and the attribute selection performed in a round-robin fashion by the assistant, after the question type selection.

In Section 2.3.3 we saw an approach for optimizing the attribute selection, and it would be worth to explore combining the question type selection together with the attribute selection, so the sequence of generated questions can be further optimized;

Estimated Durations of Question Types : the time it takes to get feedback from the user to each of the question types provided by the assistant are an estimation, but thought to be as close as possible to a real human-to-human interaction and also to verify the behaviour of having all of them with different values.

As also explored by [24, 25], these values need to be further studied for each question type for the assistant to have more accurate behaviours;

Controlling the Risk : the next question type selection is made in a non-stop fashion by the model, only stopping when the user explicitly tells that he got his desired product. This recommendation decision strategy can sometimes become annoying and/or be unnecessary.

As explored in Section 2.7, this model could be further optimized by integrating an additional action to end the conversation, not needing to wait until the user signals that he got his target product;

Model Architectures : the model is based on a deep q-network.

Although we experimented various variations of the deep q-network, as seen in Section 5.2.5, it would be worth to experiment with some more recent architectures such as Decision Transformers, which might help in getting better results, due to its ability in learning what is most important depending on the input.

Incorporating Natural Language : the question types being used by the assistant have pre-defined templates, which are being used to generate the clarifying questions.

To improve the performance of the assistant, these pre-defined templates can be replaced by natural language in order to have a more natural dialogue with the user. Having a more natural way of interacting with the user has been shown to have greater results than using pre-established and restricting formats [12].

Bibliography

[1] A. M. Andrew. "Reinforcement Learning: An Introduction". In: *Kybernetes* 27.9 (1998-12), pp. 1093–1096. ISSN: 0368-492X. DOI: 10.1108/k.1998.27.9.1093.3. URL: https://www.emerald.com/insight/content/doi/10.1108/k.1998.27.9.1093.3/full/html (cit. on p. 13).

[2] A. Biswas et al. "Seeker: Real-time interactive search". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2019-07, pp. 2867–2875. ISBN: 9781450362016. DOI: 10.1145/3292500.3330733 (cit. on p. 8).

[3] G. Cai et al. "Ask&Confirm: Active Detail Enriching for Cross-Modal Retrieval with Partial Query". In: (2021-03). URL: http://arxiv.org/abs/2103.01654 (cit. on p. 8).

[4] J. C. Caicedo and S. Lazebnik. "Active Object Localization with Deep Reinforcement Learning". In: *Proceedings of the IEEE International Conference on Computer Vision* 2015 Inter (2015-11), pp. 2488–2496. URL: http://arxiv.org/abs/1511.06015 (cit. on pp. 13, 16, 31, 52).

[5] J. Daniel and J. H. Martin. *Speech and Language Processing Chatbots & Dialogue Systems*. Tech. rep. 2020 (cit. on p. 6).

[6] A. Das et al. "Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob (2017-03), pp. 2970–2979. URL: http://arxiv.org/abs/1703.06585 (cit. on p. 16).

[7] *Deep Reinforcement Learning*. URL: http://ap.ssdi.di.fct.unl.pt/Lectures/lec/AP-18.html. (accessed: 10.02.2022) (cit. on p. 14).

[8] *Deep reinforcement learning - Wikipedia*. URL: https://en.wikipedia.org/wiki/Deep_reinforcement_learning. (accessed: 10.02.2022) (cit. on p. 14).

[9] *Deep Residual Networks (ResNet, ResNet50) – Guide in 2022*. URL: https://viso.ai/deep-learning/resnet-residual-neural-network/. (accessed: 16.02.2022) (cit. on p. 17).

[10] Y. Deldjoo, J. R. Trippas, and H. Zamani. "Towards Multi-Modal Conversational Information Seeking". In: *SIGIR 2021 - Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Inc, 2021-07, pp. 1577–1587. ISBN: 9781450380379. DOI: 10.1145/3404835.3462806 (cit. on pp. 6, 7).

[11] *For Complex Queries, Bots Will Replace Search Engines | by Max Child | Chatbots Magazine*. URL: https://chatbotsmagazine.com/for-complex-queries-bots-will-replace-search-engines-fc438bd06b16. (accessed: 09.09.2022) (cit. on p. 1).

[12] X. Guo et al. "Dialog-based Interactive Image Retrieval". In: *Advances in Neural Information Processing Systems* 2018-Decem (2018-04), pp. 678–688. URL: http://arxiv.org/abs/1805.00145 (cit. on p. 66).

[13] A. Gupta and L. S. Davis. *Beyond Nouns: Exploiting Prepositions and Comparative Adjectives for Learning Visual Classifiers*. Tech. rep. (cit. on pp. 8, 9).

[14] K. He et al. "Deep Residual Learning for Image Recognition". In: (2015-12). URL: http://arxiv.org/abs/1512.03385 (cit. on p. 17).

[15] *Introduction to Reinforcement learning*. URL: http://ap.ssdi.di.fct.unl.pt/Lectures/lec/AP-17.html. (accessed: 10.02.2022) (cit. on p. 13).

[16] P. Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: (2016-11). URL: http://arxiv.org/abs/1611.07004 (cit. on p. 12).

[17] B. R. Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* (2021), pp. 1–18. ISSN: 1524-9050. DOI: 10.1109/TITS.2021.3054625. URL: https://ieeexplore.ieee.org/document/9351818/ (cit. on p. 16).

[18] A. Kovashka and K. Grauman. "Attribute Pivots for Guiding Relevance Feedback in Image Search". In: *2013 IEEE International Conference on Computer Vision*. IEEE, 2013-12, pp. 297–304. ISBN: 978-1-4799-2840-8. DOI: 10.1109/ICCV.2013.44. URL: http://ieeexplore.ieee.org/document/6751146/ (cit. on p. 10).

[19] A. Kovashka, D. Parikh, and K. Grauman. "WhittleSearch: Interactive Image Search with Relative Attribute Feedback". In: *International Journal of Computer Vision* 115.2 (2015), pp. 185–210. ISSN: 15731405. DOI: 10.1007/s11263-015-0814-0 (cit. on pp. 7, 9, 10, 20).

[20] N. Kumar et al. *Attribute and Simile Classifiers for Face Verification*. Tech. rep. (cit. on pp. 8, 20).

[21] Z. Liu et al. *DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations*. Tech. rep. (cit. on p. 20).

[22] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[23] T. M. (M. Mitchell. *Machine Learning*, p. 414. ISBN: 0070428077 (cit. on p. 13).

[24] N. Murrugarra-Llerena and A. Kovashka. "Image Retrieval with Mixed Initiative and Multimodal Feedback". In: *Computer Vision and Image Understanding* 207 (2018-05). ISSN: 1090235X. URL: http://arxiv.org/abs/1805.03134 (cit. on pp. 1, 8, 11, 12, 16, 24, 30, 34, 36, 37, 40, 49, 50, 52, 54, 60, 65).

[25] N. Murrugarra-Llerena and A. Kovashka. "Image retrieval with mixed initiative and multimodal feedback". In: *Computer Vision and Image Understanding* 207 (2021-06), p. 103204. ISSN: 10773142. DOI: 10.1016/j.cviu.2021.103204. URL: https://linkinghub.elsevier.com/retrieve/pii/S1077314221000485 (cit. on pp. 1, 8, 11–13, 16, 24, 30, 34, 36, 37, 40, 49, 50, 52, 54, 60, 65).

[26] A. Oliva and A. Torralba. "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope". In: *International Journal of Computer Vision* 42.3 (2001), pp. 145–175. ISSN: 09205691. DOI: 10.1023/A:1011139631724 (cit. on p. 20).

[27] *Optimizers in Deep Learning. What is an optimizer? | by Musstafa | MLearning.ai | Medium*. URL: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0. (accessed: 16.02.2022) (cit. on p. 37).

[28] D. Parikh and K. Grauman. "Relative attributes". In: *2011 International Conference on Computer Vision*. IEEE, 2011-11, pp. 503–510. ISBN: 978-1-4577-1102-2. DOI: 10.1109/ICCV.2011.6126281. URL: http://ieeexplore.ieee.org/document/6126281/ (cit. on pp. 9, 20).

[29] D. Parikh et al. "Relative attributes for enhanced human-machine communication". In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 3. 2012, pp. 2153–2159. ISBN: 9781577355687 (cit. on pp. 9–11).

[30] F. Radlinski and N. Craswell. "A theoretical framework for conversational search". In: *CHIIR 2017 - Proceedings of the 2017 Conference Human Information Interaction and Retrieval*. Association for Computing Machinery, Inc, 2017-03, pp. 117–126. ISBN: 9781450346771. DOI: 10.1145/3020165.3020183 (cit. on p. 18).

[31] *Reinforcement learning - Wikipedia*. URL: https://en.wikipedia.org/wiki/Reinforcement_learning. (accessed: 10.02.2022) (cit. on p. 13).

[32] A. Saha, M. M. Khapra, and K. Sankaranarayanan. "Towards building large scale multimodal domain-aware conversation systems". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 2018-04, pp. 696–704. ISBN: 9781577358008. URL: http://arxiv.org/abs/1704.00200 (cit. on p. 7).

[33] B. Siddiquie and A. Gupta. *Beyond Active Noun Tagging: Modeling Contextual Interactions for Multi-Class Active Learning*. Tech. rep. (cit. on pp. 8, 9).

[34] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2014-09). URL: http://arxiv.org/abs/1409.1556 (cit. on p. 17).

[35] L. Su et al. "Controlling risk of web question answering". In: *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Inc, 2019-07, pp. 115–124. ISBN: 9781450361729. DOI: 10.1145/3331184.3331261 (cit. on p. 19).

[36] *Various Optimization Algorithms For Training Neural Network | by Sanket Doshi | Towards Data Science*. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. (accessed: 16.02.2022) (cit. on p. 37).

[37] *VGG Very Deep Convolutional Networks (VGGNet) – What you need to know*. URL: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/. (accessed: 16.02.2022) (cit. on p. 17).

[38] *Vision Transformers (ViT) in Image Recognition – 2022 Guide*. URL: https://viso.ai/deep-learning/vision-transformer-vit/. (accessed: 16.02.2022) (cit. on p. 18).

[39] G. Wang, D. Forsyth, and D. Hoiem. *Comparative object similarity for improved recognition with few or no examples*. Tech. rep. (cit. on p. 8).

[40] Z. Wang and Q. Ai. "Controlling the risk of conversational search via reinforcement learning". In: *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*. Association for Computing Machinery, Inc, 2021-04, pp. 1968–1977. ISBN: 9781450383127. DOI: 10.1145/3442381.3449893 (cit. on p. 19).

[41] *Will chatbots replace search engines?* URL: https://www.kmworld.com/Articles/Editorial/ViewPoints/Will-chatbots-replace-search-engines-153186.aspx. (accessed: 09.09.2022) (cit. on p. 1).

[42] B. Wu et al. "Visual Transformers: Token-based Image Representation and Processing for Computer Vision". In: (2020-06). URL: http://arxiv.org/abs/2006.03677 (cit. on p. 18).

[43] S. Xie and Z. Tu. "Holistically-Nested Edge Detection". In: (2015-04). URL: http://arxiv.org/abs/1504.06375 (cit. on p. 12).

[44] C. Zhang et al. "A Study on Overfitting in Deep Reinforcement Learning". In: (2018-04). URL: http://arxiv.org/abs/1804.06893 (cit. on p. 37).