



DOCTORAL THESIS

An Explainable Semantic Parser for End-User Development

AUTHOR:
Juliano Efon Sales

SUPERVISOR:
Dr. Siegfried Handschuh

Department of Computer Science and Mathematics
University of Passau, Germany

An Explainable Semantic Parser for End-User Development

Juliano Efsen Sales

A dissertation submitted to the faculty of computer science and mathematics in partial fulfillment of the requirements for the degree of doctor of natural sciences.

Advisor: Prof. Dr. Siegfried Handschuh

Passau, February 2021

To Joaquim do Amor Divino Rabelo

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Dr. Siegfried Handschuh for the continuous support I have received since my acceptance in the research group.

I am also grateful to Dr. André Freitas and Dr. Adamantios Koumpis for the technical guidance and living advice during challenging moments of the development of this thesis.

Additionally, I would like to recognise the crucial support I received from Mrs Stephanie Pauli in all sort of bureaucratic and administrative tasks within the University of Passau.

This work could not be completed without the endless love and support of my wife Rachel Sales. Words cannot express the dedication and care you have been giving to our family.

The research reported by this thesis was supported by a research fund from the National Council for Scientific and Technological Development, Brazil (CNPq).

Abstract

Programming is a key skill in a world where businesses are driven by digital transformations. Although many of the programming demand can be addressed by a simple set of instructions composing libraries and services available in the web, non-technical professionals, such as domain experts and analysts, are still unable to construct their own programs due to the intrinsic complexity of coding. Among other types of end-user development, natural language programming has emerged to allow users to program without the formalism of traditional programming languages, where a tailored semantic parser can translate a natural language utterance to a formal command representation able to be processed by a computational machine. Currently, semantic parsers are typically built on the top of a learning method that defines its behaviours based on the patterns behind a large training data, whose production frequently are costly and time-consuming. Our research is devoted to study and propose a semantic parser for natural language commands targeting a scenario with low availability of training data. Our proposed semantic parser follows a multi-component architecture, composed of a specialised shallow parser that associates natural language commands to predicate-argument structures, integrated to a distributional ranking model that matches the command to a function signature available from an API knowledge base. Systems developed with statistical learning models and complex linguistics resources, as the proposed semantic parser, do not provide natively an easy way to associate a single feature from the input data to the impact in system behaviour. In this scenario, end-user explanations for intelligent systems has become a strong requirement to increase user confidence and system literacy. Thus, our research designed an explanation model for the proposed semantic parser that fits the heterogeneity of its multi-component architecture. The explanation model explores a hierarchical representation in an increasing degree of technical depth, providing higher-level explanations in the initial layers, going gradually to those that demand technical knowledge, applying different explanation strategies to better express the approach behind each component. With the support of a user-centred experiment, we compared the utility of different types of explanations and the impact of background knowledge in their preferences.

Contents

I	Preamble	1
1	Introduction	3
1.1	Programming to the Masses	3
1.2	End-User Programming	4
1.3	The Semantic Parsing Problem	4
1.4	The Proposed Semantic Parser	6
1.4.1	Shallow Parser	6
1.4.2	Ranking Model	7
1.5	Explainable Artificial Intelligence	9
1.6	The Hierarchical Explanation Model	10
1.7	Research Questions, Hypotheses & Goals	11
1.7.1	Semantic Parsing	12
1.7.2	Explainability	13
1.8	Contributions	14
1.9	Publications	15
1.10	Thesis Outline	16
II	Related Work	19
2	End-User Development & Explainability	21
2.1	Introduction	21
2.2	Types of End-User Development	23
2.2.1	Visual Programming	23
2.2.2	Programming by Example	24
2.2.3	Crowd-Supported Programming	26
2.2.4	Natural Language Programming	27
2.3	Other Aspects of End-User Development	27
2.3.1	Teaching	28
2.3.2	Machine Learning	29
2.3.3	Software Engineering	29
2.4	The User Perspective	30
2.5	Explainability	31
2.6	Summary	32
3	Semantic Parsing of Natural Language Commands	35
3.1	Introduction	35
3.2	The Typical Tasks of Parsing of Natural Language Commands	36
3.2.1	Commands for Database Queries	36

3.2.2	Navigational Instructions	38
3.2.3	Program Synthesis	39
3.2.4	Natural Language Interface for APIs	41
3.2.5	Other Interests in Semantic Parsing of Commands	42
3.3	Approaches and Methods	43
3.3.1	Methods Based on Rules	43
3.3.2	Learning-Based Methods	46
3.4	An Overview of the Methods for Parsing Natural Language Commands	52
3.4.1	Approach and Feature Set	52
3.4.2	Explainability	52
3.4.3	Evaluation	57
3.5	Exposing the Gap	57
3.6	Summary	58
III Designing an Explainable Semantic Parser		59
4	A Semantic Parser for End-User Development	61
4.1	Introduction	61
4.2	Re-engineering the Learning Task	63
4.3	Componentising the Semantic Parsing	64
4.4	The Proposed Architecture	66
4.4.1	Command Segmenter	69
4.4.2	Shallow Parser	70
4.4.3	Type Inferencer	71
4.4.4	Predicate-Argument Structure	73
4.4.5	API Filter	73
4.4.6	Function Call Generator	75
4.4.7	Intent Classifier	77
4.4.8	Ranker	77
4.5	Summary	78
5	Explanation Model of a Multi-Component Semantic Parser	79
5.1	Introduction	79
5.2	Levels of Interpretability	80
5.3	The Explanation Model	81
5.3.1	Shallow Parser Rules & Syntactic Tree Layers	81
5.3.2	Word Embedding Layer	83
5.3.3	The Ranking & Classification Layer	83
5.3.4	The Comparative Explanation	85
5.4	Summary	85

IV	Experiments and Validation	89
6	The Evaluation of the Semantic Parser	91
6.1	Introduction	91
6.1.1	The Data Set	91
6.2	Indra	92
6.2.1	Word Embedding Vector	93
6.2.2	Composition Vector	93
6.2.3	Semantic Relatedness	94
6.2.4	Nearest Neighbours	94
6.3	The Encoder-Decoder Baseline Models	94
6.3.1	The Sequence-to-Sequence Baseline	94
6.3.2	The Attention-Based Baseline	95
6.3.3	Baseline Analysis	97
6.4	Evaluating the Proposed Architecture	97
6.4.1	API Filter Settings	97
6.4.2	Intent Classifier Settings	97
6.5	Results & Discussion	98
6.5.1	API Filter	99
6.5.2	Classifiers	102
6.6	Summary	104
7	The Evaluation of the Explanation Model	105
7.1	Introduction	105
7.2	The Experimental Setting	106
7.2.1	Participants	106
7.2.2	General Instructions	107
7.3	Evaluation Criteria	107
7.3.1	Mental Models	108
7.3.2	Type of Explanations	109
7.3.3	Accuracy <i>vs.</i> Explainability	109
7.4	Results & Discussion	110
7.4.1	Mental Models	110
7.4.2	Relevance of the Explanations	111
7.4.3	The Accuracy <i>vs.</i> Explainability Dilemma	112
7.4.4	Concerns About the Design of the Experiment	113
7.4.5	Considerations about Completeness	114
7.4.6	Participants' Comments and Impressions	114
7.5	Summary	115

V	Conclusions and Perspectives	117
8	Conclusions and Future Research	119
8.1	Summary	119
8.2	Discussion and Conclusions	121
8.2.1	Semantic Parsing	121
8.2.2	Explanation Model	123
8.3	Revisiting the Principles and Requirements of End-User Programming	124
8.3.1	Motivational Task	124
8.3.2	Principles	125
8.3.3	Challenges	126
8.4	Future Research Directions	129
8.4.1	Enriching the Training Data	129
8.4.2	Further Research on Explainability	130
	Appendices	133
A	Questionnaire	135
A.1	Mental Models	135
A.2	Types of Explanation	136
A.3	Accuracy over Interpretability	137
	Bibliography	138

List of Figures

1.1	The grammar dependency tree for the natural language command “ <i>Exchange 1000 Chilean Pesos to Euros</i> ”, where the arrows represent the syntactic function between the constituents.	6
1.2	A natural language command, where “ <i>Exchange</i> ” represents the <i>function description</i> , and “ <i>1000</i> ”, “ <i>Chilean Pesos</i> ” and “ <i>Euro</i> ” are command objects, whose semantic types are $\langle number \rangle$, for the first, and $\langle currency \rangle$, for the last two.	7
1.3	Reading from left to right, first “ <i>Exchange</i> ”, which represents the <i>function descriptor</i> , is projected into the Predicate Hyperspace in which the full set of function signatures are already represented. For each function signature into the pivoting area, the model projects into the Argument Hyperspace (<i>from</i> , <i>from_amount</i> , <i>to</i> in the case of <i>Currency Converter</i>) and also projects the command objects (<i>1000</i> , <i>Chilean Pesos</i> , <i>Euro</i>).	8
1.4	A high-level representation of the proposed semantic parsing.	9
1.5	Explanations of the Shallow Parser and the Type Inferencer .	11
1.6	Plot of the elements from the command and function signature in which the cosine between the points represents the semantic relatedness.	11
2.1	Cost-scope trade-offs in EUD tools (adapted from Fischer et al. (2004)).	23
2.2	A screenshot of the KidSim platform, where a user describes “ <i>a successfully written rule for jumping over a rock, incorporating 4 actions</i> ” (Gilmore et al., 1995).	24
2.3	The pair of input-output in the string manipulation of phone format. It shows a “ <i>small sampling of different ways of generating parts of an output string from the input string</i> ” (Gulwani, 2011).	25
2.4	An overview of the taxonomy to classify explanation in the context of artificial intelligence proposed by Lipton (2016).	31
3.1	A card from Hearthstone , one of the games whose cards are modelled in the program synthesis task proposed by Ling et al. (2016). The task focuses on the card description, aiming at generating a code to execute its behaviour.	41
3.2	The data set generated from the IFTTT platform associates a natural language utterance to triggers and actions.	42

3.3	Parsing of a factual sentence, where S and NP stand respectively for sentence and noun phrase, whereas \rightarrow and \leftarrow represent respectively the application of the forward and backward functions.	48
3.4	Parsing of a question, where S , NP and N stand respectively for sentence, noun phrase and noun, whereas \rightarrow represents the application of the forward function.	49
4.1	A typical method to solve this task is the encoder-decoder machine learning model in the sequence-to-sequence fashion (Sutskever et al., 2014; Ling et al., 2016; Gulcehre et al., 2016). The grey and green boxes represent LSTM cells and the <i>parameters' assignments</i> are one-hot vectors associating chunks of the input text to the parameters.	62
4.2	The current learning task receives as input a set of features from the <i>natural language command</i> , whereas the output defines the intent <i>function signature</i> together with their <i>parameters' values</i> . On its turn, the new learning task represents a model that receives as input features a pair of a natural language command and a function call, outputting a score that defines the level of representativeness of the user intent ranging from 0 to 3 whose meaning is defined in Table 4.1. . .	63
4.3	The top level of the pyramid shows the high-level components, which represent the main elements from which our proposed architecture is built. Immediately below, there are the low-level components, representing some fundamental NLP tasks, such as named entity recognition and semantic relatedness. At the bottom, we have linguistic tools and thesauri of common-sense knowledge.	66
4.4	The architecture, from a software perspective, of the proposed semantic system to interpret natural language commands. . .	68
4.5	Adapted context-free grammar from Weigelt et al. (2018) describing the basic structures, where NP and VP stand respectively for noun phrase and verb phrase.	70
4.6	A natural language command, where “ <i>Send</i> ” represents the <i>function description</i> , and both “ <i>file.doc</i> ” and “ <i>sandra@email.com</i> ” are command objects, whose semantic types are respectively $\langle file \rangle$ and $\langle email \rangle$	74
4.7	The natural language command “ <i>Exchange 1000 Chilean Pesos to Euro</i> ” is projected on the Predicate Hyperspace in which the set of function signatures is also represented. The pivoting area, represented by a blue ellipsis, determines the set of relevant function signatures for the given command. . .	75

4.8	The natural language command “ <i>Exchange 1000 Chilean Pesos to Euro</i> ” is parsed to generate a predicate-argument structure, whose command objects are then projected on Predicate Hyperspaces . Each hyperspace represents a function signature selected by the API Filter , on which the function parameters are also projected.	76
5.1	The three levels of interpretability defined in a hierarchical model.	80
5.2	A command in natural language and a list of potential function calls representing the user intent.	81
5.3	Explanations of the Shallow Parser and the Type Inferencer	82
5.4	The second layer of the explanation model describing the linguistic features and highlighting the relationship between them and the command objects.	83
5.5	Plot of the elements from the command and function signature in which the cosine between the points represents the semantic relatedness.	84
5.6	The relevance of some features in the Intent Classifier using the <i>Forest Floor</i> visualisation method.	84
5.7	The comparative explanation shows the function calls <i>currency converter</i> and <i>make a payment</i> in the context of the natural language command <i>Exchange 1000 Chilean Pesos to Euro</i>	87
6.1	A typical method to solve this task is the encoder-decoder machine learning model in the sequence-to-sequence fashion (Ling et al., 2016; Gulcehre et al., 2016; Sutskever et al., 2014). The grey and green boxes represent LSTM cells and the <i>parameters’ assignments</i> are one-hot vectors associating chunks of the input text to the parameters.	95
6.2	The Transformer model architecture defined by Vaswani et al. (2017), where the left block represents the encoder and the right block the decoder.	96
6.3	The charts show the behaviour of the recall for the Random Forest(a), SVM(b) and MLP(c) considering the three filter functions. The fourth chart (d) describes the behaviour of the ranking system when the classification component of the ranking equation is ignored.	100

6.4	The charts show the behaviour of the mean reciprocal rank for the Random Forest(a), SVM(b) and MLP(c) considering the three filter functions. The fourth chart (d) describes the behaviour of the ranking system when the classification component of the ranking equation is ignored.	101
6.5	The confusion matrix in percent of the Random Forest classifiers considering the evaluation scenario with the nearest neighbours filter function.	103
6.6	The confusion matrix in percent of the SVM classifiers considering the evaluation scenario with the nearest neighbours filter function.	103
6.7	The confusion matrix in percent of the MLP classifiers considering the evaluation scenario with the nearest neighbours filter function.	104
7.1	A command in natural language and a list of potential function calls representing the user intent.	106
8.1	An example of a multiple-command scenario composed of three commands. The chart induces the user to write a command to search for “ <i>iPhone</i> ” in Amazon followed by exchanging its price from Euro to US Dollar , and finally sending the calculated price to the user @MyMother in Skype	131
8.2	A chart describing a chain of a command to search a video on YouTube , followed by a pair of commands to share it on social media platforms.	131

List of Tables

3.1	Examples of sentences from the HuRIC data set.	38
3.2	Examples of pairs of natural language command and Bash command from the NL2Bash data set (Lin et al., 2018).	40
3.3	Extra characteristics of the encoder-decoder-based machine learning models.	53
3.4	Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 1).	54
3.5	Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 2).	55
3.6	Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 3).	56
4.1	Range of <i>intent scores</i> and their respective meaning.	64
4.2	List of possible function calls for the command “ <i>Exchange 1000 Chilean Pesos to Euro</i> ”, where the last row represents the intent score from the user perspective. The calls are generated considering a selected set of function signatures, which can be classified according to its intent score as <i>wrong function signature</i> (0), <i>right function signature with wrong parameters</i> (1), <i>right function signature with partially right parameters</i> (2) or <i>right function signature with right parameters</i> (3).	65
4.3	Tree-based rules to identify command objects (CO) and their respective qualifiers, where τ represents the POS-tag of the element $e \in V$	71
4.4	Examples of natural language commands with their respective function descriptors (d) and set of command objects (o_1, o_2, \dots, o_k) expected to be produced by σ	72
4.5	The list of reported verbs supported by the approach designed by Krestel et al. (2008).	73
4.6	Example of semantic types identified by the Named Entity Recogniser . Types marked with an asterisk are identified purely by regular expression, whereas those with double asterisks are identified by the Reported Speech Detector	74
5.1	Components present in the semantic parser along with their input features and mechanism of work. Each component uses also the features described in the predecessor components.	82

6.1	Examples of function signatures present in the KB.	92
6.2	Number of commands and function signatures present in the data set, according to their respective types.	92
6.3	Recall for Random Forest (RF), Support Vector Machine (SVM) and Multilayer Perceptron Neural Network (MLP) for different filter functions evaluated in the TOP-10 and TOP-50 scenarios. The last line (None) describes the results when no classifier is used.	98
6.4	Mean Reciprocal Rank for Random Forest (RF), Support Vector Machine (SVM) and Multilayer Perceptron Neural Network (MLP) for different filter functions evaluated in the TOP-10 and TOP-50 scenarios. The last line (None) describes the results when no classifier is used.	99
6.5	The average number (#) of <i>function signatures</i> into the pivoting area and how many functions are missed in percentage.	99
7.1	Distribution of the participants according to their knowledge in machine learning and English Grammar.	107
7.2	The results regarding the mental model assessment, presenting the average scores and Pearson correlation coefficient r in relation to machine learning knowledge (r ML) and English grammar knowledge (r EG) for both treatment and control groups. The last two rows represent the average scores grouped by the level of specialisations.	110
7.3	The average scores of the relevance given to each type of explanation on a Likert 7-point scale (-3 to 3). The second and third columns show respectively the averaged scores for the group of participants acquainted and non-acquainted in machine learning, whereas the last column represents the p-value of the statistical significance (t-test).	111
7.4	The results represent the degree to which participants favour accuracy over explainability. The average scores are represented on a 7-point scale, where 1 denotes high explainability and low accuracy, and 7 the opposite.	113
8.1	Tabular content within an email message.	125
A.1	137
A.2	137

Part I
Preamble

Introduction

Contents

1.1	Programming to the Masses	3
1.2	End-User Programming	4
1.3	The Semantic Parsing Problem	4
1.4	The Proposed Semantic Parser	6
	1.4.1 Shallow Parser	6
	1.4.2 Ranking Model	7
1.5	Explainable Artificial Intelligence	9
1.6	The Hierarchical Explanation Model	10
1.7	Research Questions, Hypotheses & Goals	11
	1.7.1 Semantic Parsing	12
	1.7.2 Explainability	13
1.8	Contributions	14
1.9	Publications	15
1.10	Thesis Outline	16

1.1 Programming to the Masses

The market demand for automation increases yearly pushed by the advances in information technologies (IT), which as a side effect boosts the demand for IT professionals. In the United States in 2018, there were more than 220,000 open positions for software developers without qualified people to fill them (The App Association, 2018), whereas Europe suffers now from a shortage of up to 500,000 IT professionals (Empirica, 2017). As traditional business faces new waves of digital transformation, there is no sign of reduction in this demand, especially considering that programming has become a key skill in almost every industry, including the automotive, healthcare, hospitality, transportation and even food (Westerman and Bonnet, 2015).

Many programming tasks, however, require neither the exercise of deeper computer science skills nor sophisticated software engineering, consisting of the composition of functions and the semantic harmonisation of data flows across different components (Paternò and Wulf, 2017). This is typically the case for analytic tasks or small-scale business processes, which do not

pose high requirements for robustness or performance under critical conditions. However many non-technical professionals such as domain experts and business analysts are still unable to build their own software due to the intrinsic complexity of coding (Sales et al., 2017). Such a complexity comes from the formalism of traditional programming languages and the technical knowledge to understand and integrate multiple software artefacts.

1.2 End-User Programming

The field of *end-user programming* has responded to this issue by proposing methods and tools to free up non-technical users to program without the syntactic, vocabulary and formal constraints of traditional programming languages (Paternò and Wulf, 2017). We define *end-user programming* as “*programming to achieve the result of a program primarily for personal, rather public use*” (Ko et al., 2011).

While this definition distinguishes the nature of the programme, we still need to characterise the end users. Blackwell (2017) classifies end users into three groups: those who *like programming*, those who *find programming useful* and those who *believe they will be good at programming*. Among other differences, the groups can be understood from a motivational point of view. While the first and third groups are motivated from an intrinsic desire, moved from an aesthetic vision of programming, the second has a more pragmatic perception, moved by the benefits the technology can offer and evaluating the trade-off of dedicating time and attention to both learning how to program and automating a task. This second group represents the target user profile of natural language programming for our research.

For instance, some repetitive manual work executed in a business context is the typical end-user task we are interested in. Despite being feasible to be automated, those tasks usually are not big enough to call the attention of the IT department, or the company has not any IT department at all, such as small ventures and liberal-professional offices.

Among other techniques for end-user development, our thesis is focused on a natural language programming approach, whose core tasks concern the ability to parse a natural language utterance into a data representation able to be executed by a computational system.

1.3 The Semantic Parsing Problem

Our task of *semantic parsing of natural language commands* consists of mapping a natural language command to a formal representation, called *function signature*, from an **API Knowledge Base**. The parser is responsible for (i) identifying the suitable function signature, and (ii) matching correctly spans

of the commands to the respective parameter values. Assuming the following natural language command:

Write to newton@example.com asking him to take a look at the NYT today.

The natural language command targets a specific function signature named “*send an email*” present in the **API Knowledge Base**. Besides identifying the intended function signature, the semantic parser also needs to both (i) identify “*newton@example.com*” and “*take a look at the NYT today*” as argument values and (ii) figure out to which parameters they should be assigned, which in this case correspond respectively to “*to address*” and “*message*”.

In the context of our research, we name *function call* the instantiation of a function signature describing the function name along with values for its parameters, mapped partially or totally, as shown below:

function name: *send an email*
params: *message*=“*take a look at the NYT today*”
to address=“*newton@example.com*”

We formalise the target problem as follows. Let F be an **API Knowledge Base** composed of a set of k function signatures (f_1, f_2, \dots, f_k) . Let $f_i = (n_i, l_i, P_i)$ be an element of F , where n_i is the *function’s name*, l_i is the *function’s provider*, and P_i is the set of *function’s parameters*. Let f'_i be a call of f_i , which also holds values for their parameters, totally or partially. Let c_i be a natural language command which semantically represents a target function call f'_i . The parser aims at building a ranking model which, given a set of function signatures F and a natural language command c , returns a list B of ordered function calls, satisfying the command intent.

This task is materialised by the data set obtained from the Task 11 of the SemEval 2017, named *End-User Development using Natural Language* (Sales et al., 2017, 2018a). The main challenge of this task relies on the size of the data set, which is composed of 185 annotated natural language commands and a set of 2005 function signatures in the **API Knowledge Base**. There are two types of commands. The first type is *if-then* recipe, which is composed of a *protasis*, i.e. the conditional clause, and the *apodosis*, i.e. the main clause. In this type of command each clause is associated to a function signature. The second type is the solo direct command, which is associated to a unique function signature. Function signatures are also of two types. **Triggers**

comprehend function signatures meant to map to the conditional part of the if-then command. **Actions** represent the set of function signatures meant to map to the main clause of the if-then commands and to direct commands.

1.4 The Proposed Semantic Parser

Following a multi-component architecture, we propose a semantic parser composed of a specialised shallow parser that associates natural language commands to predicate-argument structures, integrated to a ranking model supported by semantic relatedness metrics over a word embedding model.

1.4.1 Shallow Parser

Supported by a dependency grammar, the **Shallow Parser** relies on the analysis of the syntactic functions of the constituents in the natural language command to produce a lightweight representation defined as a predicate-argument structure. In this structure, the predicate assumes the role of the *function descriptor*, whereas the arguments assume the role of *command objects*. We define *function descriptor* as the minimal subset of tokens present in the command that allows one to identify the target function signature in the **API Knowledge Base**, and *command objects* is understood as potential descriptors or parameters or their values. Figure 1.1 shows the grammar dependency tree of the natural language command “*Exchange 1000 Chilean Pesos to Euro*”, showing its constituents and the syntactic functions between them.

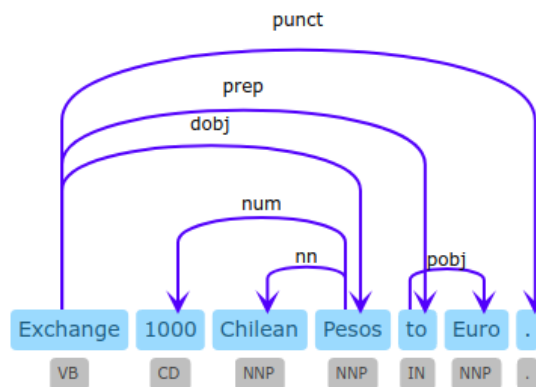


Figure 1.1: The grammar dependency tree for the natural language command “*Exchange 1000 Chilean Pesos to Euros*”, where the arrows represent the syntactic function between the constituents.

Following a rule-based approach, the **Shallow Parser** identifies “*Exchange*” as the *function descriptor* and the trio “*1000*”, “*Chilean Pesos*” and

“Euro” as command objects, which are later analysed by a **Named Entity Recogniser** and a **Type Inferencer**, assigning them the tags of “*number*” for the first and “*currency*” for the others. Figure 1.2 presents the predicate-argument structure of the natural language command obtained from this process.

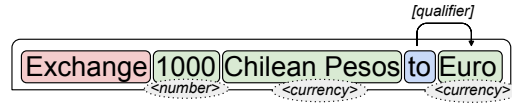


Figure 1.2: A natural language command, where “*Exchange*” represents the *function description*, and “1000”, “*Chilean Pesos*” and “*Euro*” are command objects, whose semantic types are $\langle \text{number} \rangle$, for the first, and $\langle \text{currency} \rangle$, for the last two.

1.4.2 Ranking Model

In an open semantic parsing setting, both the natural language command and the function signatures are not constrained to a restricted vocabulary. The role of the ranking model is to bridge the predicate-argument structure to a set of relevant functions according to the user intent.

The ranking model operates by projecting word embedding vectors of the text elements of both the predicate-argument structure and the **API Knowledge Base** in a set of distributional hyperspaces as depicted in Figure 1.3.

The ranking model has two types of distributional hyperspaces. The **Predicate Hyperspace** projects the vectorial representation of the function descriptor from a natural language command, together with the vectorial representation of the function names¹ of the function signatures present in the **API Knowledge Base**. The vectorial representation is obtained from a word embedding model, which enables the ranking model to measure the semantic relatedness between them by geometry. Figure 1.3 depicts **Predicate Hyperspace** in the left, where the red dot represents the function descriptor and the other dots the function names.

Considering a nearest neighbours measure, the **API Filter** defines a *pivoting area*, selecting the set of function signatures with higher semantic relatedness. For each function signature present in the pivoting area, the ranking model instantiates an **Argument Hyperspace**, projecting the set of parameters from the function signature and the set of command objects from the natural language command as represented in the right part of Figure 1.3. As the **Predicate Hyperspace**, the **Argument Hyperspace** also serves to measure the semantic relatedness between the projected elements.

¹The function name is represented by n in the formal definition of the task, which is described in Section 1.3

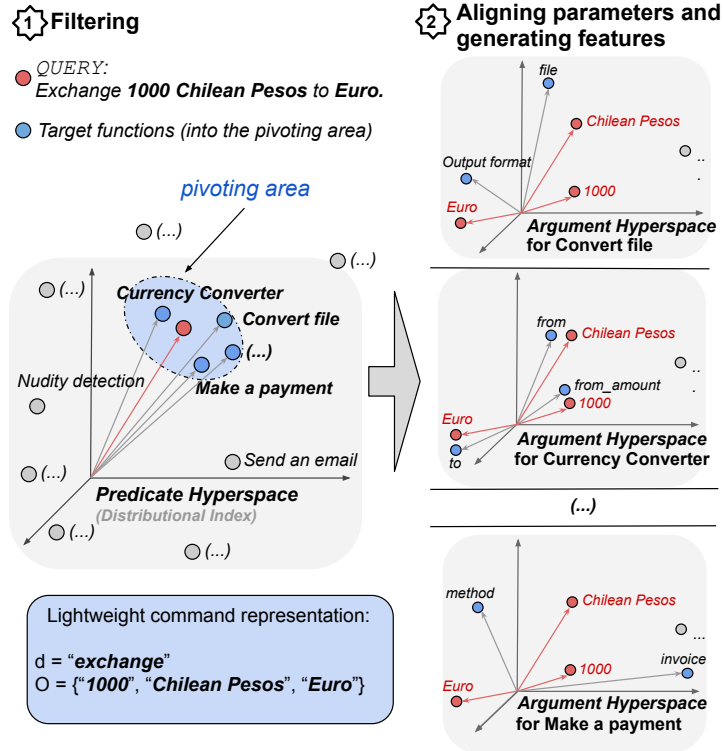


Figure 1.3: Reading from left to right, first “Exchange”, which represents the *function descriptor*, is projected into the **Predicate Hyperspace** in which the full set of function signatures are already represented. For each function signature into the pivoting area, the model projects into the **Argument Hyperspace** (*from*, *from_amount*, *to* in the case of *Currency Converter*) and also projects the command objects (*1000*, *Chilean Pesos*, *Euro*).

The semantic relatedness scores obtained from the hyperspaces, together with other features later defined in this thesis, support the construction of an **Intent Classifier**.

The **Intent Classifier** is a Random-Forest model trained with a task-specific data set to discriminate relevant function calls, i.e. those that represent the user intent, from noisy function calls, i.e. whose function signatures do not represent the user intent, or that associate object commands incorrectly to the set of parameters.

At the end, the ranking model produces a list of relevant function calls to the final user combining the set of semantic relatedness scores from the **Predicate** and **Argument Hyperspaces** and the score from the **Intent Classifier**.

Figure 1.4 shows a high-level representation of our proposed semantic

parser, highlighting its multi-component nature.

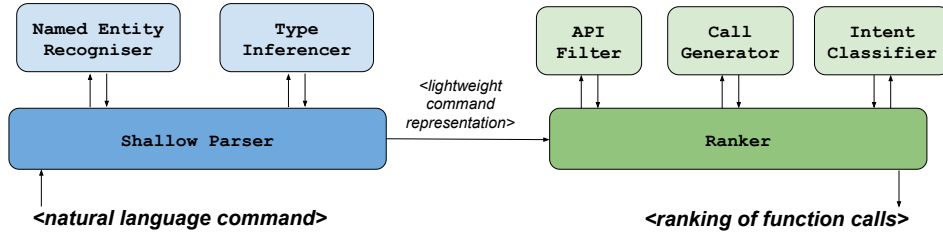


Figure 1.4: A high-level representation of the proposed semantic parsing.

In Chapter 4, we present the proposed semantic parser, describing in detail every of its components.

1.5 Explainable Artificial Intelligence

The improvement of learning models (Goodfellow et al., 2016) and the large availability of annotated data (Halevy et al., 2009) and linguistic resources (Miller, 1995) supported a strong evolution of natural language processing (NLP) methods. Such evolution provides a paradigmatic shift in the ability to build systems that analyse information at scale, it also carries risks associated with its prevalent models, such as the lack of transparency and understanding (Biran and Cotton, 2017).

To a large extent, the high performance of intelligent systems comes with the cost of transparency (Lipton, 2016), as linguistics resources and statistical learning models are generated from an intricate set of computational operations from which there is no easy way to associate a given input data to its impact in the trained model.

Given these challenges, end-user explanations for intelligent systems has become a strong requirement either to comply with legal requirements (Goodman and Flaxman, 2017) or to increase the user confidence (Zhou et al., 2016).

Natural language understanding systems that require the complex coordination of multiple NLP components, *e.g.* POS-tagger, syntactic parsing, named entity recogniser and task-oriented machine learning models, increase the complexity, since each component can explore a large spectrum of resources and learning methods (Burgess, 2018).

Although explanation models have been focusing on decision-making tasks (Stumpf et al., 2007; Zhou et al., 2018; Vorm, 2018), natural language understanding systems can also benefit from them. In addition to increasing the users' confidence on the system, explanation models can help the users to adapt their writing styles to improve the system comprehension according to the underline model.

However, while delivering a human-interpretable explanation for a single component is challenging, the problem is aggravated in the context of multi-component systems (Lipton, 2016; Burgess, 2018).

From this perspective, our research proposes a hierarchical explanation model for our semantic parser, which also supports an analysis of the human aspects of explainability in the context of our task.

1.6 The Hierarchical Explanation Model

To deliver explanation for the proposed semantic parser, we designed a model that explores a hierarchical representation in an increasing degree of technical depth. The proposed explanation model presents transparency-oriented explanations in the higher levels, going gradually to explanations that demand technical knowledge, and to the *post-hoc* ones.

Transparency-oriented explanations allow the user to understand the algorithm’s mechanism of decision, by contemplating “*the entire model at once*” and understanding each of its parts and its learning mechanism. Diversely, *post-hoc* explanations make use of interpretations to deliver meaningful information about the intelligent model. Instead of showing how the model works, it presents evidences of its rationale by making use of approximate textual descriptions and visual models (Lipton, 2016). Those concepts will be detailed in Section 2.5.

We propose an explanation model composed of seven explanations grouped into three layers.

The first layer describes the **Shallow Parser**, showing the rules activated to identify the command objects and to identify the semantic types, highlighting the words and features involved in the process as depicted in Figure 1.5. The second layer is associated with the dependency grammar, depicting the syntactic functions of the constituents in the natural language command.

The explanation model also provides a cluster-based visualisation for the word embedding representation using the **t-SNE** approach (van der Maaten and Hinton, 2008), where it plots the semantic elements that contribute in the matching process from both the predicate-argument structure and the function signatures, as shown in Figure 1.6.

In its lower layer, the explanation model is devoted to the most technical explanations, which present the mathematical expression that defines the final ranking function, along with the features used in both the expression itself and in the **Intent Classifier**.

In Chapter 5, we describe the proposed hierarchical explanation model in detail, depicting visually every of its layers.

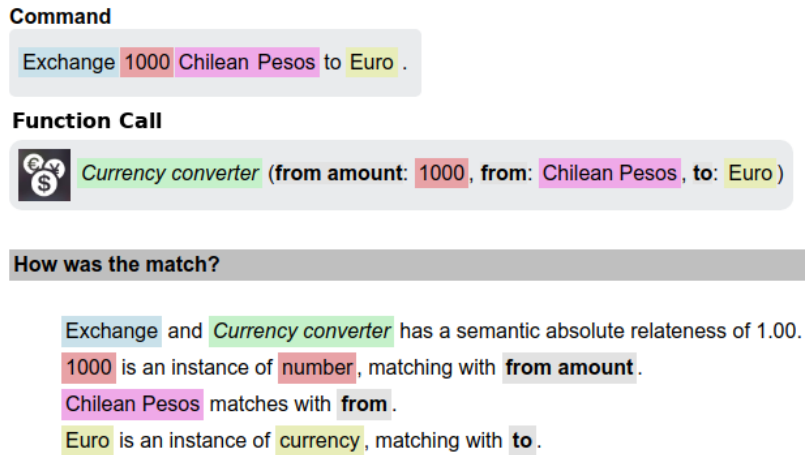


Figure 1.5: Explanations of the Shallow Parser and the Type Inferencer.

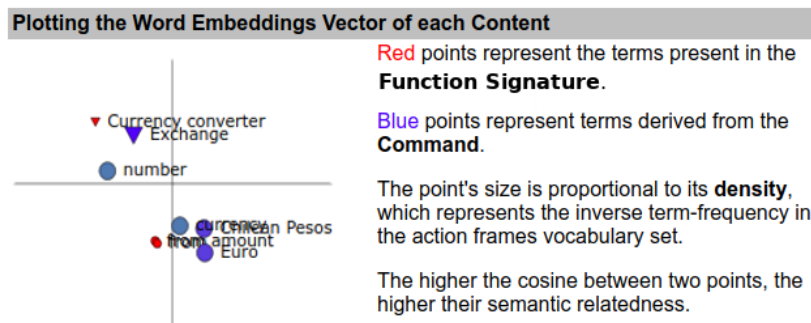


Figure 1.6: Plot of the elements from the command and function signature in which the cosine between the points represents the semantic relatedness.

1.7 Research Questions, Hypotheses & Goals

We define the goal of our research by five research questions grouped as *semantic parsing research questions* and *explainability research questions*.

The semantic parsing research questions aim, first, at determining whether end-to-end encoder-decoder machine learning algorithms are appropriate to construct a semantic parsing with a restriction in training data. Secondly, assuming the hypothesis of failure, we also aim at assessing an alternative multi-component approach that combines a specialised shallow parser that associates natural language commands to predicate-argument structures, integrated to a ranking model supported by semantic relatedness metrics over a word embedding model.

The explainability research questions aim at a user-centred analysis to

evaluate both the helpfulness of different types of explanations, and the impact of background knowledge in the preference for the different explanations using the alternative multi-component parser as the object of study. Furthermore, considering different intelligent applications, we also investigate the user perception on the dilemma of accuracy versus explainability.

1.7.1 Semantic Parsing

We analyse the semantic parsing focusing on answering the following research questions.

Research Question 1.1

How suitable are encoder-decoder machine learning models to build a semantic parser for our target task, considering the low availability of training data?

Typically, end-to-end encoder-decoder machine learning models are composed of many layers of neurons, whose large set of parameters depend on large data sets to learn a task (Du et al., 2018). Despite the existence of theoretical models trying to estimate formally the minimum recommended size of a data set for a given task, providing such a number is still impractical (Du et al., 2018). To assess the performance of those models in an empirical approach, we evaluate several neural network architectures based on both sequence-to-sequence models and attention mechanisms varying their hyper-parameters.

Research Hypothesis 1.2

The combination of a specialised shallow parser with type-inference capabilities and a ranking model supported by relatedness measures on a word embedding model can be used to define a semantic parser for natural language commands under the restriction of low availability of training data.

We evaluate the proposed semantic parser in different settings. During the experiments, we analyse the performance of the architecture considering distinct implementations of `API Filter` and the `Intent Classifier`.

We evaluate three implementations of the `API Filter`. The first implements the identity function, i.e. no filter is applied, aiming at measuring the relevance of the filtering step. The other implementations are a TF/IDF weighting scheme and the nearest neighbours method. For the `Intent Classifier`, we also evaluated three implementations: Random Forest, Support Vector Machine and a simple *Multilayer Perceptron Neural Network* (MLP).

The evaluation was carried out in two scenarios: the first considers the function calls ranked up to the 10th position (TOP-10), whereas the second, up to the 50th (TOP-50).

We present and discuss the results regarding the semantic parsing research questions in Chapter 6.

1.7.2 Explainability

We analyse the explanation model focusing on answering the following three research questions.

Research Question 2.1

To what extent are users able, irrespective of their technical background, to improve their mental models by associating the linguistic features from the explanations to the system's behaviour?

A mental model is a cognitive representation of the external world to support the human reasoning process (Jones et al., 2011). The closer a person's cognitive representation to the external world, the higher the understanding and the ability to take decisions about it (Johnson-Laird, 1983).

We designed an experiment to understand the effectiveness of the explanations to help the end users understand the underlying model of the semantic parsing. We recruited a group of people to participate in our study, dividing them into the experiment group and control group. We asked both groups to use the semantic parser respectively with and without access to the explanations. At the end, they answered a set of questions designed to evaluate the user's mental model by associating linguistic features to the system's behaviour.

Research Question 2.2

How does the user knowledge in machine learning affect the preference for technical explanations?

This research question supports the analysis of the utility of different types of explanations considering the impact of background knowledge in the preferences of the users. We asked the participants of the treatment group to assess the helpfulness of each of the explanations presented in the model. Considering the background knowledge of each participant in machine learning, we analyse their preferences for different types of explanations.

Research Question 2.3

To what degree are users willing to favour explainability over accuracy given a certain contextual task?

For a large number of tasks, the techniques that deliver the higher performance are also those that provide less transparency (Lipton, 2016). Despite the evolution of the explanation methods in providing interpretability to a raising range of learning models, the dilemma is still relevant (Ribeiro et al., 2016). We asked the participants about their preferences for explainability over accuracy in three hypothetical tasks:

- *in the case of a command selection tool;*
- *in the case of a cancer exam of a relative;*
- *in the case of a bank loan decision.*

The results are analysed to assess whether the simple fact of having been exposed to explanations and their concepts impact the preference of the participants in the dilemma.

We present and discuss the results regarding the explainability research questions in Chapter 7.

1.8 Contributions

This work provides the following contributions:

- The definition of a multi-component semantic parser for natural language commands under the restriction of a small training set.
 - The proposed distributional semantic parsing method operating with a Nearest Neighbours `API Filter` and a Random Forest `Intent Classifier` was able to solve up to 68% of the commands considering the TOP-10 evaluation scenario, and up to 85% when considering the TOP-50, with the mean reciprocal rank (MRR) scoring around 0.3, which means that the target function calls are placed on average at the 3rd or 4th positions.
- The definition of a hierarchical explanation model for the proposed semantic parser.
 - On average, participants in the treatment group gave scores 55% higher than those in the control group (1.13 *vs.* 0.73, $p < 0.05$) in mental model assessment.

- A user-centred analysis of the preference for explanations, and the user’s willingness to favour explainability over accuracy.
 - Simple explanations associating the input provided by the user to the features of the system are the more useful type of information for a general audience.
 - Users tend to favour accuracy over explainability. Among the three hypothetical task, the *bank loan decision* diverges from the others. We conjecture the possible social unfairness of a bank loan decision incentivises the participants to increase their interest for an explanation, whereas, for the other two hypothetical contexts there are, in theory, no incentives for unfairness.
- An analysis of the existing literature with regard to semantic parsing of natural language commands;
- Definition of a long-term vision for a new end-user programming model centred on searching.

At its centre, this thesis concentrates on the proposal and evaluation of a semantic parsing approach for natural language commands with low availability of training data. Additionally, our research proposes and evaluates an explanation model analysing human factors related to the preferences on the type of explanations and the perception of their relevance in contrast to the system performance. This central part is complemented by broader discussions concerning the challenges and principles of end-user programming.

1.9 Publications

Different aspects of this work were disseminated on the following publications:

- Juliano Efon Sales, André Freitas, Siegfried Handschuh, *A User-centred Analysis of Explanations for a Multi-component Semantic Parser*, 25th International Conference on Natural Language & Information Systems - NLDB, Germany, 2020.
- Juliano Efon Sales, André Freitas, Douglas Oliveira, Adamantios Koumpis, Siegfried Handschuh, *Revisiting Principles and Challenges in Natural Language Programming*, 13th International Joint Conference on Knowledge-Based Software Engineering, Cyprus, 2020.
- Juliano Efon Sales, André Freitas, Siegfried Handschuh, *An Open Vocabulary Semantic Parser for End-User Programming using Natural Language*, 12th IEEE International Conference on Semantic Computing (ICSC), USA, 2018 (**Best Student Paper Award**).

- Juliano Efsen Sales, Leonardo Souza, Siamak Barzegar, Brian Davis, André Freitas and Siegfried Handschuh, *Indra: A Word Embedding and Semantic Relatedness Server*, 11th Language Resources and Evaluation Conference (LREC), Japan, 2018.
- Juliano Efsen Sales, André Freitas, Siegfried Handschuh, *SemEval-2017 Task 11: End-User Development Using Natural Language*, Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval), Canada, 2017.

1.10 Thesis Outline

This thesis can be outlined as follows:

- After introducing the target problem and formalising the task, as well as presenting the research questions, hypotheses and goals, this introductory chapter describes an overview of the structure of the thesis.
- In Chapter 2, we provide an overview of the area of end-user development, showcasing the benefits and limitations of the four main schools of thought related to it, namely, *natural language programming*, *programming synthesis*, *visual programming* and *crowd-supported programming* (Section 2.2). Our analysis suggested that a comprehensive end-user development system can take advantage of hybrid approaches. In Section 2.5, we also shed light on the topic of explainability of intelligent systems. Despite the high attention received in recent years, we show that the literature has not yet presented studies concerning explainable models applied to natural language understanding tasks. As the understanding of the system can both increase the users' confidence and allow them to take more advantage from the systems, explainability is considered as a core aspect of our research.
- Chapter 3 addresses a comprehensive survey on semantic parsing for natural language commands. We start with an analysis describing the main tasks associated to the four branches of research in the area, namely, *commands for database queries*, *navigational instructions*, *program synthesis* and *natural language interface for APIs*. Next, we survey established strategies and methods in the field, suggesting a classification that is orthogonal to the task domain. This classification divides the methods in two main groups, *rule-based approaches* and *learning-based approaches*, presenting their evolutions chronologically: Section 3.3.1 demonstrates the evolution of rule-based approaches from simple slot-filling-templates and string matching, to the massive use of external resources, mainly in the form of thesauri, while, Section 3.3.2 demonstrates how learning-based systems evolved from methods based

on grammar formalisms, as *combinatory categorial grammar*, to deep neural network models based on encoder-decoder architectures. The survey also highlights the lack of attention dedicated to explainability in the field of semantic parsing (Section 3.4).

- Chapter 4 presents our semantic parsing method for natural language commands focused on a task for which there is not a large set of training data available. The chapter starts presenting evidence, which is later confirmed by the experiments described in Chapter 6, that the encoder-decoder methods for this type of task are highly dependant on large data sets. Because a reasonable large data set is not available, the goal of our research is to provide an alternative parsing method, which is accomplished by a hybrid seven-component architecture.
- Chapter 5 describes an explainable model for the proposed semantic parsing. In contrast to other approaches, our explainable model targeted a multi-component setting, which demands the construction of different explanation mechanisms for the different types of strategies used in each component.
- Chapter 6, which is the first of two chapters dedicated to the analysis of the results and discussion, is devoted to the evaluation of the semantic parser presented in Chapter 4. Firstly presenting the shortcomings of the encoder-decoder deep neural networks to learn from small data sets in the context of our target task. Subsequently, we evaluate our proposed semantic parsing using the data set from the Task 11 of the SemEval 2017.
- Chapter 7 represents the second part of the evaluation, which focuses on a user-centred analysis of the effectiveness of the different types of explanation methods.
- Chapter 8, in addition to a summary, provides an assessment and discussion of the conducted research from the perspective of the initial research questions. We also revisit the principles and challenges for the development of an effective end-user programming environment. Lastly, it debates the applicability and the limitations of the thesis, concluding with what we consider as the future research directions.

Part II
Related Work

End-User Development & Explainability

Contents

2.1	Introduction	21
2.2	Types of End-User Development	23
2.2.1	Visual Programming	23
2.2.2	Programming by Example	24
2.2.3	Crowd-Supported Programming	26
2.2.4	Natural Language Programming	27
2.3	Other Aspects of End-User Development	27
2.3.1	Teaching	28
2.3.2	Machine Learning	29
2.3.3	Software Engineering	29
2.4	The User Perspective	30
2.5	Explainability	31
2.6	Summary	32

2.1 Introduction

The field of *end-user development* is focused on shifting the main discussion in software engineering and human-computer interaction from “*making systems easy to use*” to “*making systems that are easy to develop*” (Lieberman et al., 2006), under a very pragmatic motivation. As detailed in the introductory chapter, the current scenario of the shortage of information technology professionals points to a real risk our society confronts of having the pace of automation reduced due to the lack of code writers.

Domain experts processing data, analysts automating recurrent tasks, or a businessman testing an idea on the web are potential end users that, given access to the correct set of techniques and tools, can materialise their programming demands without the mediation of professional developers. The notion of *correct set of techniques and tools*, although hard to precisely define, is associated with the cost of learning the development environment and the programming skill itself. As presented by Ko et al. (2004), among

the six learning barriers in end-user programming systems (*design, selection, coordination, use, understanding, and information*), four (*design, selection, use, and information*) can have a large impact when running away from the specific syntax of traditional programming languages.

Simplifying the programming language, however, has a side effect in expressivity (Fischer et al., 2004). Figure 2.1 depicts the dilemma between the coverage scope of a given programming language and its cost of learning. Traditional general-purpose programming languages such as Java and C++ have high expressivity, allowing programmers to build a wide gamut of different types of programs. This expressivity, however, comes with a high cost to learn the languages' peculiarities that go beyond their already complex syntax and lexicon, such as module and library integration methods and execution requirements to only mention a few. On the opposite corner, domain-specific languages and tools for simple adaptation and customisation give a low entry barrier for end users but offer a limited capacity of computation description. For instance, spreadsheets and the IFTTT platform¹ are examples of successful methods able to deliver clear value for end users. Their coverages, however, are respectively restricted (*i*) to a small set of applications that are numerically focused, without a straightforward mechanism to integrate with external libraries and services, and (*ii*) to programs in the form of *if-then* recipes. As summarised by Fischer et al. (2004), "*the goal [and challenge] for EUD tools is to reduce the learning burden while providing powerful facilities to address a wide range of problems*".

The literature in EUD is broader, presenting different flavours of techniques. In Section 2.2, we introduce the main types of strategies that have been applied in the EUD field. We extend the discussion presenting other important aspects of EUD in the context of disciplines such as teaching, software engineering and crowdsourcing development. This preliminary explanation aims at setting the basis for a short discussion on EUD from the user perspective (Section 2.4).

Most of the end-user development techniques are somehow supported by machine learning methods. While the recent evolution of those methods provides a paradigmatic shift in the ability to build systems that analyse information at scale, it also carries the risks associated with its prevalent models, such as the lack of transparency and understanding, posing a challenge to the adoption of intelligent systems in environments with high social and economic impact (Biran and Cotton, 2017). Either to comply with legal requirements (Goodman and Flaxman, 2017) or to increase the user confidence (Zhou et al., 2016), offering end-user explanations for intelligent systems has become a strong requirement. Section 2.5 provides a more thorough insight to this issue. This chapter concludes with a summary in Section 2.6.

¹<http://ifttt.com>

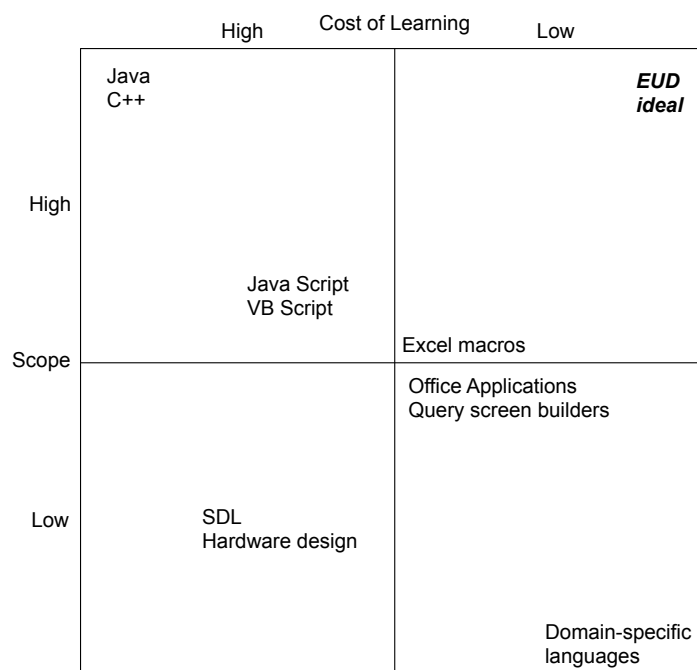


Figure 2.1: Cost-scope trade-offs in EUD tools (adapted from Fischer et al. (2004)).

2.2 Types of End-User Development

Since its debut, end-user development (EUD) has matured four main branches of research: *visual programming*, *programming by example*, *natural language programming* and *crowd-supported programming*. This section depicts a general view of each branch, fostering a debate over their capabilities and limitations.

2.2.1 Visual Programming

Myers (1990) defines visual programming as coding in at least two dimensions, in contrast to text-based, arguably a uni-dimensional mean of expression. The author classifies visual-programming techniques according to their *form of specification*, which includes *flowcharts* whose primary focus is the algorithm, and *spreadsheets* whose focus is the data, to mention a few.

Regardless of the focus, the literature can also be understood from the perspective of the field by classifying domain-specific visual programming platforms. For instance, different initiatives focused on sub-fields of computer science, where Smutný (2011) and Francese et al. (2017) targeted mobile applications, Ray (2017) paid attention to Internet of Thing platforms, Booth and Stumpf (2013) focused on the open-source hardware platform

Arduino, and more recently Xie et al. (2019) built tools for the development of deep machine learning models. Visual programming also got popular in other domain-specific fields, traditionally dominated by professionals without computer science background such as Biology (Milicchio et al., 2016), Finance and Astrophysics (Segal, 2007).

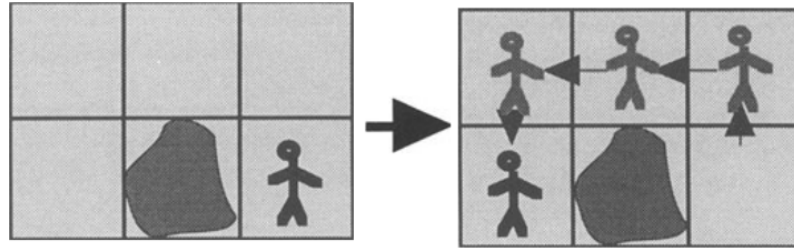


Figure 2.2: A screenshot of the KidSim platform, where a user describes “a successfully written rule for jumping over a rock, incorporating 4 actions” (Gilmore et al., 1995).

Figure 2.2 shows how students can write visually a rule to move an agent in the KidSim platform (Gilmore et al., 1995).

Supporters of visual programming argue that “the human visual system and human visual information processing are optimised for multi-dimensional data”, whereas traditional programming languages are “conventionally presented in a one-dimensional textual form, not utilising the full power of the brain” (Myers, 1990). Additionally, they also believe graphic representations favour a higher level description of the code, which puts less emphasis on the language syntax, a perception also shared by Fischer et al. (2004), according to whom the syntax and lexicon issues make learning a traditional programming language comparable to learning a new human language. The limitation in expressivity, however, counts against visual programming, as better exposed in Section 2.4.

2.2.2 Programming by Example

Program synthesizers can be designed to either act as the desired program, mimicking its behaviour, or generate the desired program’s formal representation, usually a source code (Kant, 2018). In both cases, the most important distinction is related to the method of construction. *Programming by example*, also called *programming by demonstration*, is a type of program synthesis that emerged after the attempt of producing programs from formal specifications (Lieberman, 2001). As the effort to generate programs from such specification was found equivalent to produce the code itself, new approaches started stating the problem in such a way that a given model could induce the program from a set of input-output pairs, which is now

called *programming by example* (Gulwani, 2016).

The work of Gulwani et al. (2012) is a well-known example of the success of rule-based systems for programming by example. Part of its success is due to the focus on string manipulation tasks and table transformation, in which a component is responsible for generating rule-based programs per sample, while a second component generalises those programs based on a domain-specific language.

Figure 2.3 shows an instance of a pair of input-output of the problem of string manipulation in the context of phone number edition. In addition to Gulwani et al. (2012), other researchers investigated the problem, as the recent work of Shu and Zhang (2017) which proposed a deep learning model based on the typical encoder-decoder architecture (discussed in detail in Section 3.3.2). Also in the domain of text editing, Liang et al. (2010) promote an approach for *learning by examples* based on Bayesian probability. In particular, the authors define a combinatorial logic representation based on a tree model, which simplifies the use of a probabilistic context-free grammar to infer the correct tree representation for a given program amongst different tasks.

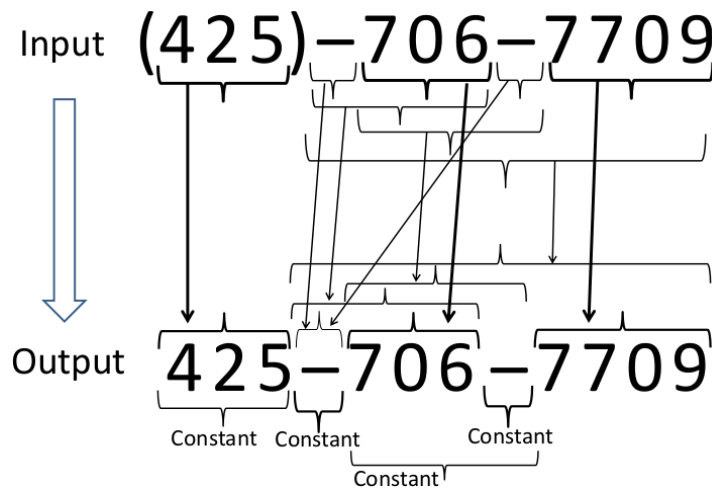


Figure 2.3: The pair of input-output in the string manipulation of phone format. It shows a “*small sampling of different ways of generating parts of an output string from the input string*” (Gulwani, 2011).

Programming by example, however, is not restricted to the string manipulation problem. For instance, whereas `GoScripter` targets a tool to automate repetitive web tasks, learning how to execute clicks and filling fields in a browser (Leshed et al., 2008), in a more theoretical fashion, Reed and de Freitas (2016) described a neural-based approach to learn math operations.

From a user-centred perspective, Gulwani et al. (2015) conducted a study on **StriSynth**, a programming by example tool to generate operating system scripts developed on the top of **Flash Fill** (Gulwani, 2011), showing that users with scripting experience found scripting more “*helpful*” than the programming by example tool, although taking more time to code (Santolucito et al., 2018). The study, however, doesn’t evaluate end users without a technical background.

Manshadi et al. (2013) presents an example of a hybrid approach aiming at improving the performance of a programming-by-example system by integrating natural language programming, where a statistical method, in addition to the pair of input-output, also incorporates textual description in the form of the grammar dependency tree.

The simplicity for generating training data, which in some cases can be done in an automated fashion, is the appealing point of programming by example techniques. This type of method, however, as most of the program synthesizers, suffers from the lack of graceful repair, which prevents end user to correct or take advantage from partially incorrect outputs, as we describe later in detail in Sections 2.4 and 8.3.3.

2.2.3 Crowd-Supported Programming

Crowd-supported programming is the newer strategy among the four branches of research in end-user development. Its main idea is to build a platform from which end users can have access to programs developed by other developers after providing either a high-level specification in the form of a natural language description or input-output pairs. The platform puts in the loop a set of qualified supporters (generally professional developers) able to provide code or give online feedback for the user’s programming demands. An example of a crowd-supported programming platform is presented by Huang et al. (2016), whose framework helps end users not only in constructing *if-then* rules but also in identifying a proper solution to the target problem.

From another perspective, the work of Manshadi et al. (2012) combines crowd-supported programming and programming by example, where, after receiving the program description in natural language, the supporters organised under the crowdsourcing mechanism produce a set of high-quality input-output pairs that comply with the original specification. Once a certain number of pairs are collected, the platform applies a programming-by-example model to automatically generate the desired program.

LaToza et al. (2014) and Cochran et al. (2015) apply similar methods to break down large programs into micro-tasks that can be distributed among many developers. Extra requirements to select and merge the resulted programs, however, make those approaches less suited for end users.

In the main, this method can be understood as a simplified method for “hiring” qualified developers, instead of effectively allowing end users

to program. While the method can be effective with a low learning curve involved, besides not teaching the end user how to program in the long run, there will always be a cost associated with the service.

2.2.4 Natural Language Programming

Sammet (1966) was one of the first to advocate that technical language is an overhead, representing a constraint to concentrate on the problem and to design a proper solution. However, the idea that human-computer languages should be designed with principles of natural language was not a consensual one.

A more structured criticism was provided by Dijkstra (1979) who calls natural language programming a “*foolish idea*”. His main plea is supported by the fact that a machine does exactly what it is instructed to do and natural language is essentially vague, imprecise and ambiguous (Dijkstra, 1964). From his point of view, in addition to demanding the construction of a much more complex “compiler”, the main point relies on the comparison to mathematics, where the lack of proper formalism would stop the development of the natural science, as during the time of rhetoric texts that prevented ancient mathematicians to go far.

The prototypes developed by Ballard and Biermann (1979) and Biermann et al. (1983) represent the typical initial effort in constructing a natural language programming environment. As the main use of the computer was focused on mathematical methods, these first attempts targeted a general-purpose programming language able to execute low-level operations such as matrix manipulation.

Commands such as “*Double x and store in y*”, which was the focus of the first initiatives, show the reason why part of the scientific community was opposed to such technology. However, with the advent of the web and the proliferation of web services, software artefacts with a higher level of granularity could better represent the typical level of abstraction present in the end-user mind, and so, making a natural language interface closer to her/his intents.

Although textual language increases the expressivity, natural language also adds a significant overhead to interpret it. Despite its initial criticism, the idea of using natural languages to program was not completely rejected. The literature points out some initiatives throughout the decades, whose initiatives are surveyed in Chapter 3.

2.3 Other Aspects of End-User Development

Lieberman et al. (2006) presented end-user development as a confluence of other fields, such as software engineering and artificial intelligence, to name a few. As most of the areas are in fact sub-areas of computer science, we

tend to focus on technical aspects, ignoring the way humans learn and how research findings about the human brain are associated with coding (Krishnamurthi and Fisler, 2019). To properly understand the realm of EUD, we also need to look, not only from a technical perspective but also from other disciplines, such as teaching and software engineering. Additionally, we highlight in this section the recent impact of new machine learning techniques in the field, especially from the use of statistical methods in natural language processing.

2.3.1 Teaching

Since the early times of microcomputers, educators saw programming as a powerful competence to help students develop basic thinking and problem-solving skills (Pea, 1987). Given the intrinsic barriers of traditional programming language, since the beginning simplified languages were applied, such as LOGO, one of the first initiative to teach programming skills using an end-user educational environment where students could program with visual aids (Muller, 1985).

Indeed, visual programming is very present in teaching programming for children. In the study of Gilmore et al. (1995), children wrote rules using visual elements and the mouse to move an agent in a spatial scenario using the KidSim end-user platform. In another platform called RAPTOR, students made use of visual flowcharts not only to develop, but also debug their initial programs (Carlisle et al., 2005).

Principles to improve the teaching process of programming to novices can also be used to improve the usability of EUD systems. Krishnamurthi and Fisler (2019) suggest that, instead of focusing on programming paradigms, we need to emphasise the behavioural properties and features of a language itself. Looking from this new perspective affects directly the teaching process since we stop taking for granted a series of interpretations about a given syntax and their semantics. The main goal is to avoid the classical notion of paradigms that suggests a disjoint classification, considering that several modern languages offer constructions or sub-language features that permeate functionalities originally associated with different paradigms (Krishnamurthi and Fisler, 2019). For instance, although Python is generally classified as an imperative language, it also allows native constructions inspired in functional languages.

Pane and Myers (2006) conducted a study to identify how students would program game scenarios without any guidance or restriction. From the answers they received it was evident that the participants used graphical depictions and texts to “*program*”. This allowed the researchers to conclude that event- and rule-based structures were the dominant strategy used by non-programmers, while users privileged aggregation functions applied to sets instead of iterations.

Despite the known simplicity of visual programming to serve effectively as an introductory stage to programming, the pedagogical community, however, has recognised the necessity for the transition to text-based programming, given its higher expressivity (Kölling et al., 2015).

2.3.2 Machine Learning

With the maturity of machine-learning-based techniques, new methods started moving from applications of rules and templates to neural-based approaches which can deal with latent information and deliver approximative solutions that go beyond the examples expressed in the training data set (Goodfellow et al., 2016).

These new classes of machine learning models, namely, recurrent neural networks such as *Long Short-Term Memory* (LSTM) and *Gated Recurrent Unit* (GRU), and attention-based models like Transformers (Vaswani et al., 2017), have been attracting attention over the last years due to their performance on natural language understanding tasks, producing robust representations of language models, especially when aligned with word embedding (Camacho-Collados and Pilehvar, 2018).

In addition to the impact seen in the academic literature (presented in detail in Chapter 3), machine-learning-based products and services already available in the market have gained popularity. Supported by these technologies, companies such as *Google*, *Amazon* and *Apple* have built assistants that trigger atomic requests and answer simple questions by voice (López et al., 2017). For instance, the Amazon team defined a language for the virtual assistant **Alexa** to represent natural language commands based on an ontology of *actions*, *types*, *properties* and *roles* (Kollar et al., 2018). Despite the robustness of the model in delivering results under real-world conditions, there is a lack of comparability and transparency (Perera et al., 2018) as the data set, for instance, are not publicly accessible by the community.

Although such an approach has been proved to be helpful for daily-life situations, it does not attend the fundamental concept of programming, *e.g.* to group sets of instructions to which we can assign a meaning. So, voice assistants are ineffective for business environments, where the demand for automation requires the composition of multiple services along with the integration with a large volume of structured data.

2.3.3 Software Engineering

Software engineering has also extended their attention to the code of end users. Ko et al. (2011) surveyed how the literature covers the practices of engineering during the end user development of software, detailing how end users gather and identify their requirements, their main development strategies, their ability to reuse code from other developers (end users or

not) and their debugging practices and tools.

In a recent paper, Barricelli et al. (2019) study the field by a mapping study to analyse software engineering practices based on a quantitative approach and by means of classifying the literature according to seven dimensions (*type of approach, interaction technique, phase in which the approach is adopted, application domain, target use, class of users, and type of evaluation*). Different from previous works, this mapping study adopts a quantitative approach trying to identify the representativeness of methods and techniques in the literature.

2.4 The User Perspective

Considering the different characteristics of the four main branches of research in end-user development, along with the other aspects of end-user development so far presented, the question still pertains regarding how can a programming platform better serve the (needs of the) end users?

Visual programming where blocks and graphics provide an easier syntax to end users allows the developers to start fast, but its effectiveness, in the long run, is limited due to (i) restrictions in its expressivity and (ii) the change of the *notional machine* (Guzdial, 2015). Notional machine is the abstraction of an idealised computer, from the runtime point of view, aiming at describing the semantics of the programs and defining the typical behaviour of the code instructions (Sorva, 2013). Furthermore, visual programming is frequently constrained to a sub-set of programming constructors, which limits the type of computation that can be described. To gain expressivity, this scenario eventually forces to change to a textual language (Kölling et al., 2015). As there is a significant difference between the notional machines of visual programming environments and textual environments, the previous programming experience does not help to construct a gentle learning curve (Krishnamurthi and Fisler, 2019).

Programming by example (Gulwani, 2016) and *code synthesis* (Rodrigues Filho et al., 2019) have evolved significantly in recent years allowing respectively the inference of programs from some instances of input-output examples and the “*translation*” of natural language into code. These approaches, however, suffer from the lack of *graceful repairing*, i.e. in the case of wrong output, they don’t provide a mechanism to correct or improve the results, except by redoing the task from scratch, without taking advantage of the previous results.

Peleg et al. (2018) expressed this limitation as the dilemma of accepting or rejecting fully a synthesised program. As an alternative, the authors suggest that users could give “*granular feedbacks*” in order to accept or reject parts of the programs. This approach, however, as evidenced in the evaluation population which has a technical background, does not fit for end

users, given the judgement about whether a given part is relevant or not depends on understanding the target programming language. In this sense, to the best of our knowledge, there is no approach for end users to deal with the issue of lack of graceful repairing, in terms of providing a mechanism to correct or improve the results, except by redoing the particular task from scratch. This limitation prevents the systems from taking advantage of the previous results.

Having a textual language at the centre of the end-user development environment since the beginning, and maintaining it as the main programming interaction method, allows the user to keep the same notional machine during their programming experience, without preventing the use of visual aids and functions inspired by programming-by-example as auxiliary tools.

This context suggests the construction of a hierarchical model in which the user deals with simple elements at first, easy to manipulate, and which can be later improved by a different mechanism.

2.5 Explainability

Providing end-user explanations for intelligent systems has become a strong requirement either to comply with legal requirements (Goodman and Flaxman, 2017) or to increase the user confidence (Zhou et al., 2016). However, what does *explanation* really mean in the context of machine learning?

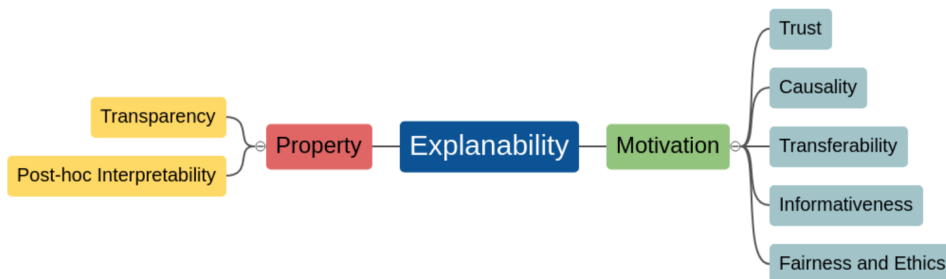


Figure 2.4: An overview of the taxonomy to classify explanation in the context of artificial intelligence proposed by Lipton (2016).

Lipton (2016) defined a comprehensive taxonomy, summarised in Figure 2.4, highlighting various criteria of classification such as motivation (*trust, causality, transferability, informativeness* and *fairness & ethics*) and property (*transparency* and *post-hoc interpretability*).

Trust is by far the most common motivation presented in the literature, as Pazzani (2000), Zhou et al. (2016) and Biran and Cotton (2017) suggest, whose results showed users demonstrate higher confidence when using a system whose workings they understand. *Fairness & ethics* is also a strong driver as the well-known European General Data Protection Regula-

tion (Goodman and Flaxman, 2017) guarantees both rights “*for meaningful information about the logic involved*” and “*to non-discrimination*” to prevent bias and unfair behaviour (Selbst and Powles, 2017). The regulation targets decision-making algorithms as in the works of Mooney (1996) and Bolukbasi et al. (2016). Although less representative, explanation is also used to support user’s feedback as Stumpf et al. (2007) and Kulesza et al. (2015). From the *property* criterion, *transparency* allows the user to understand the algorithm’s mechanism of decision, by contemplating “*the entire model at once*” and understanding each of its parts and its learning mechanism. Typical methods complying with these requirements are the so-called “*explainable by design*” such as *linear regression*, *decision trees* and *rule-based approaches* when dealing with small models (Lipton, 2016).

Post-hoc explanations, however, make use of interpretations to deliver meaningful information about the AI model. Instead of showing how the model works, it presents evidence of its rationale by making use of (i) textual descriptions (Silva et al., 2018), (ii) visualisations able to highlight image parts from which the decision was made (Selvaraju et al., 2016), (iii) 2D-representation of high-dimensional spaces (van der Maaten and Hinton, 2008), or (iv) explanation by similarity (Caruana et al., 1999). While this type of explanation does not tell precisely how the output was generated, it still presents useful information about its internal mechanism. For instance, Ribeiro et al. (2016) show a typical *post-hoc* method, which associates meaning to elements of the neural network by perturbing the model to approximately identify partial predictions.

Other works are also interested in explanation models whose evaluation is centred on the users. As one of the first initiatives, Pazzani (2000) evaluated user preferences about the explanation of mail filtering profiles. Applied to different tasks, Stumpf et al. (2007) and Kulesza et al. (2015) analysed the feedback of end users to improve the system’s training process. More recently, Zhou et al. (2018) studied how information regarding uncertainty and correlation in AI systems impacts the confidence of specialist and non-specialist users taking decisions. Applying a distinct methodology, Vorm (2018) studied user’s explanation demands by systematic interviews and group dynamics.

2.6 Summary

Over the decades, end-user development has explored four main branches of research: *visual programming*, *programming by example*, *natural language programming* and *crowd-supported programming*.

Visual programming methods have their focus on bringing the simplicity of dealing with visual elements. As a side effect, the techniques usually fails to allow a higher level of expressivity.

Programming by example defines a very convenient method to construct a program based on instances of input-output pairs, lowering the demand for technical knowledge. However, the level of generalisation can be limited in complex tasks. Additionally, it suffers from the lack of graceful repair, which prevents trained methods to take advantage of partially correct models.

The more recent crowd-supported programming, a strategy to integrate workers with programs, gives the ability to avoid relying on imprecise methods and models. However, the dependency on humans makes the systems, expensive, complex and less scalable.

Natural language methods allow high expressivity and scalability, although, their methods require the development of more sophisticated natural language understanding methods, as discussed in the next chapter.

The new generation of programming platforms for end-user developers has the potential to change the ways we commercialise, reuse and publish software, as well as creating new business opportunities. Jointly with functionalities such as information extraction and text generation, it can enlarge significantly the range of tasks that can be programmed by end-user developers. The applications emerging from this new context will profoundly affect the level of automation in the society and how humans deal with repetitive tasks.

Semantic Parsing of Natural Language Commands

Contents

3.1	Introduction	35
3.2	The Typical Tasks of Parsing of Natural Language Commands	36
3.2.1	Commands for Database Queries	36
3.2.2	Navigational Instructions	38
3.2.3	Program Synthesis	39
3.2.4	Natural Language Interface for APIs	41
3.2.5	Other Interests in Semantic Parsing of Commands	42
3.3	Approaches and Methods	43
3.3.1	Methods Based on Rules	43
3.3.2	Learning-Based Methods	46
3.4	An Overview of the Methods for Parsing Natural Language Commands	52
3.4.1	Approach and Feature Set	52
3.4.2	Explainability	52
3.4.3	Evaluation	57
3.5	Exposing the Gap	57
3.6	Summary	58

3.1 Introduction

In this chapter, we are interested in surveying semantic parsers applied to *natural language commands*, i.e., the problem of translating a natural language utterance to either formal code or machine-readable actions to control a robot or program a software application. For us, we make no distinction between a solution that controls a given application by itself, accessing directly the means of control, or those that translate the user intent to an intermediate, formal or technical language, such as lambda calculus, code in a programming language like Python or Java, or even variations of the Structured Query Language (SQL).

This chapter starts presenting typical tasks of natural language commands, grouping them in four categories: *commands for database queries*, *navigational instructions*, *program synthesis* and *natural language interface for APIs*. We also present some co-related research initiatives in the field, such as methods to self learn action and vocabularies.

In Section 3.3, we review prominent approaches and methods, debriefing them into two groups. The first group is composed of the *methods based on rules*, where we also put in evidence the *problem of vocabulary mismatch*. We also discuss methods supported by ontologies and standards defined by the *Semantic Web* community, which had an important impact on allowing easier access to external resources, together with derived metrics such as distances and similarities. The second group comprehends *learning-based methods*, where we first depict *hybrid learning methods*, which combine rule-based generated features and grammar formalisms. Then, we describe the methods based on *end-to-end machine learning models*, whose main idea is to feed a sequence-to-sequence deep neural network with a large set of samples from which the model can learn the latent pattern without intervention.

In addition to classifying the approaches and methods by those groups, Section 3.4 proposes a broader analysis that considers other important aspects, such as the relevant feature set, the type of evaluation and its support to explanation.

The chapter concludes exposing the gap of the literature this work aims at covering, by analysing the specificities of the methods proposed so far, putting in perspective the particularities of the data sets.

3.2 The Typical Tasks of Parsing of Natural Language Commands

In the context of natural language commands, we classify semantic parsing in four main tasks: *commands for database queries*, *navigational instructions*, *program synthesis*, and *natural language interface for APIs*.

3.2.1 Commands for Database Queries

Frequently, commands in natural language aim at retrieving information, as shown in the *Air Travel Information System (ATIS)* data set, which is composed of queries to search for flights according to cities and dates (Price, 1990; Dahl et al., 1994). Following a similarly focused scope, the *GeoQuery* data set comprehends searching for geographic information of US states (Zettlemoyer and Collins, 2005), the *JOBS* data set comprises queries for work positions according to skills, salary and location (Tang and Mooney, 2001), and the series of *Dialog System Technology Challenges* deals with bus timetable and restaurant data (Williams et al., 2013; Hen-

derson et al., 2014a,b). For instance, the quote below shows a typical query from the JOBS data set.

what job uses languageid0 and languageid1 are available in locid0 and pay num_salary a year.

Those data sets, each of them focused on a unique narrow domain, represent a simplified scenario for semantic parsing of commands, because they deal with a constrained set of entity types, a restricted vocabulary, and one (or a few) types of command (*e.g.* getting travel information), varying only the parameters. These features significantly benefit the training process, making the data set denser, where the syntactic and semantic regularity facilitates the generalisation.

Since the 70s (Hendrix et al., 1978), another popular task is regarding *natural language interface for databases*, whose main shape is in the form of generation of SQL queries from natural language, now called *text-to-SQL*¹.

Data sets for *text-to-SQL* are either adapted from other tasks or made specifically for it. For instance, although initially planned to deal with ambiguity in the context of academic information, such as authors, papers and affiliations, Li and Jagadish (2014) used the **Microsoft Academic Search Dataset** (Roy et al., 2013) as a base for a text-to-SQL task, producing a new data set composed of 196 query sets. On the other hand, the community has also built larger data sets to expand from single- to multi-domain tasks. The large size of the data sets also aims to allow the use of deep learning methods. For instance, Zhong et al. (2017) proposed the **WikiSQL**, which potentially covers all domains of knowledge present in the **Wikipedia**. From a list of more than 24k tables found on the pages of the **Wikipedia**, users were recruited to manually paraphrase questions generated by a template previously mapped to valid SQL queries. In total, the corpus comprehends more than 80k queries (Zhong et al., 2017). Although expanding the domain, **WikiSQL** is only composed of simple queries over a single table. The **Spider** data set is an answer to this simplicity by adding queries over multiple tables and requiring the use of nested queries and more complex constructors such as **GROUP BY** and **JOIN** (Yu et al., 2018).

Another limitation is the *ad-hoc* nature of the queries. As the name suggests, the *Semantic Parsing in Context* corpus (**SParC**) mainly differentiates from the previous by adding context to the queries in the form of a dialogue. Generated from more than 200 different databases, while also targeting different domains of knowledge, the corpus is structured as a conversation between a user and an intelligent system, represented by the target

¹As the task of text-to-SQL aims at translating a natural language utterance to a technical language, it can be also classified as code synthesis. Our decision to describe it under the *database queries* context relies on historical reasons, since the topic of natural language interface for databases has been more associated to the information retrieval community than to the program synthesis community.

parser (Yu et al., 2019b). A subsequent initiative called CoSQL for *Conversational text-to-SQL* goes in the same direction, but including utterances for which there is no valid SQL or valid data to answer it. This increases the complexity by reproducing a typical real scenario, where users might want to query information that is not available (Yu et al., 2019a).

3.2.2 Navigational Instructions

Many applications related to the parsing of natural language commands are within the context of human-robot interaction. *Navigational instructions* aim at controlling the movements of a physical device, usually a robot, by natural language. Although not meant for intelligent applications, the **HCRC Map Task Corpus** is one of the first linguistic resources targeting spatial instructions, where one instructor guides a follower in a predefined route on a partially shared map (Thompson et al., 1993). Vogel and Jurafsky (2010) used this resource to construct one of the first data sets for navigational instruction tasks.

The HuRIC (*Human Robot Interaction Corpus*) described a list of spoken commands between humans and robots. It is composed of three data sets which were developed in the context of three different events in a crowd-sourcing fashion. Examples of instructions are shown in Table 3.1. The corpus differentiates by having their data annotated using *Frame Semantics* together with *Holistic Spatial Semantics* (Bastianelli et al., 2014).

<i>Queries</i>
go to the kitchen
activate the television
go to the bathroom take the rag go to the hall and clean the mirror

Table 3.1: Examples of sentences from the HuRIC data set.

The most popular data set, however, is the **SAIL** proposed by MacMahon et al. (2006), which became the *de-facto* evaluation method for navigational instruction parsers. It has the goal of guiding an agent in a virtual environment to a specific location on a map. The instructions are based on visual elements present in the paths such as hanging pictures and floor patterns. The corpus is composed of 786 route instructions on three virtual indoor environments. The **SAIL** data set shows the essence of the navigation-instruction tasks, mixing text and visual elements in a context of directional instructions and spatial movements.

From a detailed analysis, Can and Yuret (2018) demonstrated the **SAIL** data set lacks diversity. For instance, more than half of the instructions follow the same linguistic pattern, which makes the learning models biased and not capable to generalise to all possible instructions. To overcome this limitation, the authors suggested the **SAILx** data set, which aims at balancing

the type of instructions by enriching the data set with synthetic instructions generated from templates, which, on their turn, were constructed following the instructions present in the original SAIL data set.

In 2014, the SemEval workshop hosted a task related to the parsing of natural language spatial commands, also targeting a robotics scenario (Dukes, 2014). In this case, however, the task proposed the parsing of commands to a *robot control language* to control a robot arm that moved objects of different types and colors on a squared board representing the spatial region (Dukes, 2013). The 3.4k+ sentences were generated from an online game that encouraged users to describe a formal instruction to a robot to transform an initial state into a second one. Although claiming to allow “*rich linguistic structure*”, such as ellipsis, anaphoric references, multi-word spatial expression and lexical disambiguation, the vocabulary is rather restricted when describing the objects and the actions.

The works of Tellex et al. (2011) and Hemachandra et al. (2015) represent a set of initiatives in navigational instructions tasks that operates on physical prototypes (or their simulators) in real scenarios. Those tasks consist respectively of manipulating large objects with a robot forklift in logistic operations and moving a generic robot around a university campus. Real-world scenarios allow exploring models that combine language and data captured from physical sensors and cameras.

More recently, new data sets have opened the space for research in other contexts, such as controlling a robot arm in a kitchen environment to “*make coffee*” and “*boil water*” (Misra et al., 2016), or moving in a house with commands such as “*go up the stairs and turn right*” (Anderson et al., 2018). Recently, Chen et al. (2019) formulated the **Touchdown** data set as a treasure-hunt task aiming at finding hidden objects in a map made of real-life visual urban environments from *Google Street View*.

Predominantly, as navigational instructions are centred on spatial aspects, the linguistic variability, i.e. vocabulary and syntactic variations, tends towards simpler linguistic patterns. In contrast to addressing a vocabulary gap, the main challenge relies on interpreting language elements in the context of spatiality and movements. For instance, it is common that the semantic parsing phase is executed by a method defined by a third party, while the contribution relies on grounding language to sensor signals, discovering the map or planning the execution of the movements (Matuszek et al., 2010; Brooks et al., 2012; Matuszek et al., 2013; Walter et al., 2014).

3.2.3 Program Synthesis

As mentioned in Section 2.2.2, we can divide program synthesis into two main groups, depending on the nature of the training data. We consider problems formulated from the program’s input-output perspective as an instance of the *programming by example* task. Instead, this section is inter-

ested in tasks aiming at constructing programs (or their source code) from natural language utterances.

In this context, the canonical task of program synthesis is the generation of general-purpose programming code such as `Java` or `Python`, from natural language descriptions. Gvero and Kuncak (2015) defined a small data set composed of 90 pairs of text-code, where the text is represented by a set of keywords and the code by a single-line expression in the `Java` programming language. In the case of `Python`, Oda et al. (2015) presented a task to generate the code of functions from English pseudo-code, whose data set reaches around 18.8k data pairs. More recently, Barone and Sennrich (2017) collected a larger data set from `GitHub` to allow the training and testing of the generation of functions from documentation. As the code is generated from repositories of active projects, the authors argue the new data set better represents the real complexity of the task.

Some researchers opted for using a *domain-specific language* (DSL) to intermediate the natural language and the system control. For instance, Desai et al. (2016) extended the `ATIS` (Price, 1990) data set to include query representation in the form of a DSL using this data set in the problem of synthesizing programs, also releasing data sets for the problem of mapping utterances to text-editing scripts. In a similar task, Pasupat et al. (2018) evaluates a set of off-the-shelf models to analyse the problem of mapping commands to web elements in the context of `HTML/CSS`.

Other types of formal languages, such as spreadsheets and operating system scripts are also of relevance in this area. Gulwani and Marron (2014) built a domain-specific language to bridge natural language and spreadsheets to execute operations like `map`, `filter`, `reduce`, `join` and `formatting`. The data set is composed of around 3.5k natural language commands in the context of 40 different tasks over a spreadsheet. From another perspective, Lin et al. (2018) explored the synthesising of `Bash` script from natural language, by defining the `NL2Bash` data set composed of more than 9k pairs of natural language/`bash` commands, covering more than 135 programs. Table 3.2 presents examples of pairs from the `NL2Bash` data set.

Natural Language	Bash Command
Monitor all processes whose command includes ‘java’.	<code>top -p “\$(pgrep -d ‘,’ java)”</code>
Add executable permission to “rr.sh”	<code>chmod +x rr.sh</code>
Make a copy of file file1 named file2	<code>su - jetty cp file1 file2</code>

Table 3.2: Examples of pairs of natural language command and `Bash` command from the `NL2Bash` data set (Lin et al., 2018).

Newer data sets have opened a space to interpret coding in a broader way. Ling et al. (2016) introduced two tasks based on card games, from

which a text description serves as a code specification to define the cards' behaviour in the game, such as attacking the opponent given a condition as depicted in Figure 3.1.



Figure 3.1: A card from *Hearthstone*, one of the games whose cards are modelled in the program synthesis task proposed by Ling et al. (2016). The task focuses on the card description, aiming at generating a code to execute its behaviour.

Although representing a challenging task from an academic point of view, the main drawback of program synthesis is its lack of *graceful repairing*, as debated in Section 2.2.2. In general terms, in the case of producing a wrong output, program synthesis doesn't provide a mechanism to correct or improve the results.

3.2.4 Natural Language Interface for APIs

The problem of semantic parsing in the context of natural language interface for APIs has changed significantly along the time. Its initial shape was in the form of web-service orchestration, which gained large attention during the consolidation of the Semantic Web standards (Berners-Lee et al., 2001).

More recently, using the data of the *ifttt.com* platform, Ur et al. (2016) defined a task to create *if-then* "recipes" from the natural language descriptions provided by the users. Figure 3.2 shows a typical example of a "recipe" and its description. This data set, however, has one important drawback: the values assigned to parameters are missing. The task is limited to the mapping of the actions that comprise the recipe, keeping aside the instantiation of the parameter values. This limitation arises from the fact that in most of the cases the description given by the users does not include such information. For instance, Figure 3.2 shows the natural language utterance "If *BitCoin* rises above \$3000, Send Notification", which is associated to the trigger `Price raises above` and the action `Send a notification`

from the IFTTT app, however the values that need to be assigned to the parameters `ticker symbol` (BTC) and `price` (3000) aren't present in the data set. This lack of information makes the data set very limited.



Figure 3.2: The data set generated from the IFTTT platform associates a natural language utterance to triggers and actions.

In the context of an email API, Su et al. (2017) proposed the NL2API, from which the user can retrieve messages and events combining query options. Their contribution has a special focus on a method to create the data set, employing combinatorial heuristics and a new crowd-sourcing methodology, which generates an initial natural language description for each API call using a simple grammar, and asking the crowd workers to paraphrase it.

The Task 11 of the SemEval 2017 presented a data set to support natural language programming which differs from other works in at least two aspects (Sales et al., 2017). First, the test collection presents a high variability in both vocabulary and grammar structure, when compared to the other test collections available so far. Secondly, the data set is composed of a small training set, requiring the application of semantic parsing methods able to generalize from few examples (Kubis et al., 2017).

3.2.5 Other Interests in Semantic Parsing of Commands

The domain literature also comprises initiatives focused on auxiliary aspects of semantic parsing. Many research groups explored self-learning methods as Thomason et al. (2015) covering the acquisition of new vocabulary, and Amos Azaria (2016) building an instructable mobile application focused on learning the composition of commands.

In navigational-instruction tasks, a typical co-challenge is the *simultaneous localisation and mapping problem* (SLAM), where the robots need to discover the map of unknown environments while keeping track of its relative position, as shown by Duvallet et al. (2016). Similarly, Walter et al. (2013, 2014) described the process of teaching a robot the environment by describing the premises using natural language while walking throughout a building, in which the human user acts as a map discovering assistant.

Other works have been more concerned about reliability, such as Popescu et al. (2003), which introduced a theory describing the notion of “*semantically tractable*” to define the level of confidence of the mapping between utterances and SQL queries.

3.3 Approaches and Methods

Now, under the context of the aforementioned typical tasks, we present a set of notable methods grouped into two categories: *rule-based methods* and *learning-based methods*.

3.3.1 Methods Based on Rules

At the centre of a *rule-based method*, we find a hand-crafted set of instructions to transform an input sentence to the expected output. In the initial systems, the rules take into consideration only the word, following basically a string matching approach. For instance, the natural language interface for a data base proposed by Guida and Tasso (1982) defines an algorithm that applies rules at the level of vocabulary.

What call our attention is the supposedly good performance that many of those primitive methods present, which is attributed to two reasons. First, in most of them, the authors used a data set described by a restricted vocabulary and a constrained variation of syntactical constructions, which nowadays we name as *controlled natural language* (Wyner et al., 2010).

The use of slot-filling-like rules reveals those expectations, as in the case below, from the work of Ballard and Biermann (1979), showing an example of a template in natural language to add a value to a given variable represented respectively by the two slots.

Add __ to __.

NaturalJava, a natural language interface for programming in Java, is another case of the use of slot-filling. It is composed of 23 frames to which a set of verb phrases do the matching later, with a series of assumptions that transfer their approach in a non-explicitly controlled natural language (Price et al., 2000).

Evaluations conducted with end users would in theory allow the assessment of those approaches in a real-world scenario, which would indicate more robustness of a system. However, the way the evaluation is designed plays a significant role. For instance, the parser described by Ballard and Biermann (1979), was, some years later, evaluated in a user-setting (Biermann et al., 1983). Composed of 23 students from a university, the users were instructed to write code to solve linear equation problems, where the method was evaluated qualitatively according to the users’ opinions. Yet,

the users were instructed to input text complying with the expected controlled natural language (Biermann et al., 1983).

The second reason why primitive methods showed impressive performance is because they were also engineered to cover the previously known set of utterances, in such a way that the evaluation does not include unknown samples as in the work of Maas and Suppes (1985). This latter, though making use of a rudimentary context-free grammar, comprised a table responsible to convert tokens in natural language to executable instructions which are manually crafted to cover all data set.

While still based on a rule-based method, Liu and Lieberman (2005a) expands the feature set, initially restricted to tokens, with POS-tags and dependency trees in the natural language interface **Programmatic Semantics**, which understands nouns as data structures, verbs as functions and adjectives as properties. Given the set of restrictions imposed to the input command, it can be understood, once again, as a parser for a controlled natural language. Their **Metafor** model (Liu and Lieberman, 2005b) forms a later application of the **Programmatic Semantics** in a context of describing code as a history.

This approach had wider applications, as in the work of Gulwani and Marron (2014) that transformed natural language utterances to an internal domain-specific language following two steps. In the first, a component assigned a type to each token using a simple string matching approach. Next, another component used a set of hand-made rules to identify the corresponding DSL-expression based on lexicon and their types.

The **NALIGE** prototype shows one of the first use of the grammar formalism to represent the natural language utterance (Manaris and Dominick, 1993). The prototype allowed software developers to construct natural language interface for a terminal of an operating system (OS) by defining manually a grammar using an extended version of the *augmented context-free grammar* (Hendrix et al., 1978). The framework was evaluated by 85 users accessing 27 commands of a UNIX-based OS (Manaris et al., 1994).

The Problem of Vocabulary Mismatch

An experiment from the 1980s that attracted high visibility and interest showed 80% of people familiar with a domain would use different words to name the same concepts, which is known as *the problem of vocabulary mismatch*, which quickly became the main concern of semantic parsing (Furnas et al., 1987).

Word normalisation is the strategy of Little and Miller (2006), which applied stemming to reduce the vocabulary variability for word edition of documents and manipulation of web pages. In both evaluation scenarios, the system used a simple heuristic of tokenize and stem that combines the largest set of character matching from the natural language command to the

function, aligned with the number and types of parameters that the function requires.

The main improvement, however, comes from the use of linguistic resources and structured ontologies as described in the next section.

Linguistic Resources, Semantic Web and Their Metrics

More sophisticated methods came with the advent of linguistic resources, such as *WordNet* (Miller, 1995), from which algorithms can, for example, easily expand vocabularies by playing with sets of synonyms and graph-based similarity metrics (Agirre et al., 2009).

For instance, the work of MacMahon et al. (2006) that released the *SAIL* data set, proposed an initial approach based on rules over syntactic tags, enriching the vocabulary with *WordNet* senses. A similar approach is applied in the context of the text-to-SQL task. Also supported by a method based on rules, Li and Jagadish (2014) mapped dependency trees to an intermediate query representation named *Query Tree* using the Jaccard Coefficient (Rogers and Tanimoto, 1960) together with *WordNet* to expand the vocabulary matching, from which a SQL query can be generated.

Supported by a statistical metric, Gvero and Kuncak (2015) proposed a *Java* synthesizer that generates code by either a predefined mapping of text and code declarations or comparing them by a score based on a bag-of-words metric.

Ontologies opened the space for taking advantage of third-party resources to expand linguistic knowledge. Englmeier et al. (2006) propose an approach to process storybooks written in natural language, restricted to a controlled vocabulary, aiming at interpreting the commands described in it, and thus translating them to a choreography script. Relying on the similarity between an ontology-based service description and a formal representation of the user request, Bosca et al. (2006) designed a method in which a restricted set of words and sentence templates compose services. Lim and Lee (2010) also present a similar work which makes use of keywords and main verbs to map a natural language command to an OWL-S ontology. Their data set, however, was artificially created, which favours the vocabulary matching. In addition to the ontology, the works of Pop et al. (2010) and Sangers et al. (2013) also make use of linguistic resources by respectively enriching the vocabulary with *WordNet* and applying *WordNet*-similarities metrics (Miller, 1995).

Wong (2007) creates a tool in which non-programmer users are able to use information collected from different websites using their web services APIs. The tool generates some sort of a mash-up, where complex tasks can be divided into simpler ones that are processed using data flow metaphors.

Aiming at better understanding the requirements of such users and attempting to simplify service composition, Namoun et al. (2010) propose

several guidelines for the development of service composition tools.

Unanimously, the main contribution of these works is the design of a software architecture able to compose distributed services using a natural language interface, even though the semantic methods present rudimentary strategies to comprehend the natural language utterances, grounding mostly in a controlled vocabulary, keywords and POS-tags rules.

On the other hand, these works highlight the concept of the choreography of high-level components and consolidate the notion of a distributed execution to create web-scale programs. Rules are still one of the main points of solution in those methods. However, the works present in this group differentiate by depending heavily on third-part resources defined under the semantic web patterns and principles. Another common characteristic is the use of graph-based measures to calculate the semantic relatedness of terms.

3.3.2 Learning-Based Methods

Generally, we can divide the learning-based methods into two groups. The first group, *hybrid learning methods*, combines rule-based components, generally to preprocess features, with learning methods that act in selected parts of the tasks. Given their architecture, we can also call them *multi-component learning methods*. On the other hand, the second group is composed of *end-to-end learning methods*, which are defined from full-fledged learning methods, generally based on deep neural network models, able to digest the input features to the expected answer. A set of initiatives from both types of methods are exemplified in the following sections.

Hybrid Learning Methods

Hybrid methods combine rule-based components with learning methods. In most of the literature, the rule-based step is responsible for pre-processing the data, mainly focused on the generation of features. For instance, we consider the work of Giordani and Moschitti (2012), which creates a text-to-SQL translator in an information-retrieval fashion. Although the authors crafted a set of manual rules to translate chunks of text to SQL snippets, the core of the contribution is a ranking function based on a *support vector machine* (SVM) (Cortes and Vapnik, 1995) that defines the relevance of the output considering scores calculated by a weighting schema of both the terms from the natural language text and the database schema.

Learning methods can also be supported by *inductive logic programming*, as in the *Chillin* model, which defines a logic checker that is trained from a set of logic statements written in the *Prolog* programming language (Zelle et al., 1994). The algorithm aims at compacting the set of statements to generalise the knowledge expressed in the clauses. A similar approach is later used in a series of works, for instance, as in Zelle and Mooney (1996)

to parse database queries, and Tang and Mooney (2001), which used first-order logic representation as an intermediate language, applying a primitive alignment algorithm that operates over string matching between the phrases and the logical representation.

A notable contribution in the field is the WASP semantic parsing algorithm, where Wong and Mooney (2006) interpreted semantic parsing as a language translation task. At its core, WASP used a *synchronous context-free grammar* to build a statistical machine translation parse from natural language to the expected output (varying according to the task, either **GeoQuery** or **RoboCup**). To learn the grammar, the method defines a lexical acquisition step, aligning words, which feed a simple *synchronous parser* (Aho and Ullman, 1972). Given its inability to handle logical variables, the authors later extend the WASP parser “*by adding a variable-binding mechanism based on lambda calculus, which allows for compositional semantics for logical forms*” (Wong and Mooney, 2007).

In the context of a navigational instruction task, where a robot acts in collaboration with humans to execute rescue and risk missions, Tellex et al. (2011) defined a model using a statistical model inspired by the *Broyden-Fletcher-Goldfarb-Shanno algorithm* (Andrew and Gao, 2007) to map the natural language utterance to a formal notation called **Spatial Description Clauses** (Kollar et al., 2010). As other methods targeting physical prototypes the challenge targets more the grounding phase, i.e. the matching between the instructions and the sensor signal, than the linguistic variability.

In the context of the **SAIL** data set, Chen and Mooney (2011) generated a formal navigation plan from the set of actions and the natural language instructions. Then, they constructed a data set to train the semantic parser by associating n-grams to actions according to their co-occurrence. The base of this work was later improved by three initiatives. Chen (2012) reduced the complexity of the algorithm responsible for the generation of the formal navigation plan, Kim and Mooney (2012) replaced the original learning method with a probabilistic context-free grammar, and later, added a re-ranking algorithm (Kim and Mooney, 2013).

Wang et al. (2015) defines a semantic parser composed of a *framework* “*which provides domain-general information*” and a *builder* who “*provides domain-specific information*”. Both components operate over a rule-based grammar to map utterances to logical forms considering the lexicon and its category. The parser is then trained using a set of paraphrased variations of the natural language phrases using a probabilistic model based on **AdaGrad** (Duchi et al., 2011).

Using the previously presented **HCRC Map Task Corpus**, Vogel and Jurafsky (2010) trained a semantic parser using reinforcement learning. Later, Andreas and Klein (2014) developed a model where the input data are represented as a vector condensing both the language and the spatial information.

More recently, the authors improved the model by refining a set of rules to align utterances to actions (Andreas and Klein, 2015).

The Case of Combinatory Categorical Grammars

Combinatory categorical grammars (CCG) is a grammar formalism defined under a set of lexicon, whose lexical items have a category and a semantic representation, and a set of combinatory rules (Steedman, 1996, 2000). A category can be a simple syntactic type such as NP (standing for noun phrase) or a more complex as $(S \setminus NP)/NP$, which acts as functions (where S standing for sentence). As an example, consider the lexicon defined below where the first component represents the word, which is followed by the category and the semantic type, whose representation is usually done in the form of lambda calculus (Zelle and Mooney, 1996; Artzi and Zettlemoyer, 2013; Artzi et al., 2014).

$$\begin{aligned} \text{California} &:= NP : \text{california} \\ \text{Nevada} &:= NP : \text{nevada} \\ \text{borders} &:= (S \setminus NP)/NP : \lambda x. \lambda y. \text{borders}(y, x) \end{aligned}$$

The combinatory rules aim to allow the combination of the categories. The following two functional applications represent the canonical operations for CCG:

1. Forward Application ($>$): $X/Y \quad Y \Rightarrow X$
2. Backward Application ($<$): $Y \quad X \setminus Y \Rightarrow X$

Those functions show that categories of the X/Y accepts a token of category Y to its right, while $X \setminus Y$ accepts a token of category Y to its left. Figure 3.3 and 3.4 depict examples of CCG parsers respectively of a factual sentence and a question both from the SAIL data set.

$$\begin{array}{c} \text{California} \qquad \text{borders} \qquad \text{Nevada} \\ \hline \text{NP} \qquad (S \setminus NP)/NP \qquad \text{NP} \\ : \text{california} : \lambda x. \lambda y. \text{borders}(y, x) : \text{nevada} \\ \hline \text{S} \setminus \text{NP} : \lambda y. \text{borders}(y, \text{nevada}) \\ \hline \text{S} : \text{borders}(\text{california}, \text{nevada}) \end{array}$$

Figure 3.3: Parsing of a factual sentence, where S and NP stand respectively for sentence and noun phrase, whereas $\text{—}>$ and $\text{—}<$ represent respectively the application of the forward and backward functions.

The methods based on grammar formalisms generally follow a two-component pattern, where the first component is responsible for inducing a

What	states	border	Hawaii
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	NP
$: \lambda f \lambda g. \lambda x. f(x) \wedge g(x)$	$: \lambda x. state(x)$	$: \lambda x. \lambda y. borders(y, x)$	$: hawaii$
$S/(S\backslash NP) : \lambda g. \lambda x. state(x) \wedge g(x)$		$S\backslash NP : \lambda y. borders(y, hawaii)$	
$S : \lambda x. state(x) \wedge borders(x, hawaii)$			

Figure 3.4: Parsing of a question, where S , NP and N stand respectively for sentence, noun phrase and noun, whereas \longrightarrow represents the application of the forward function.

lexicon, whereas the second component acts as a probabilistic model that learns an optimal derivation from the set of trained data. For instance, Zettlemoyer and Collins (2005) defines a **GENLEX** function, representing the first component, which operates on a set of rules to generate a large combination of the sentence words to the categories according to their POS-tag. The authors then apply a probabilistic model using gradient descent to identify the minimal set of lexicon items to maximise the derivation of the examples in the test set.

Artzi and Zettlemoyer (2013) extend this work to operate in a navigational instruction setting. The main changes rely on a joint parsing and execution model which is demanded to allow the model to update the internal state (location) while going through the map.

End-to-End Machine Learning Models

From a data set extracted from *ifttt.com*, similar to that described by Ur et al. (2016), a subsequent work also explored the combination of a feed-forward neural network with a context-free grammar to improve the previously presented results. Liu et al. (2016) also explored the data set, this time getting higher performance by using an attention-based neural network architecture. In the same task, Beltagy and Quirk (2016) developed a feed-forward neural network, which uses word and character n-grams and Brown clustering as features.

Deep learning models (Goodfellow et al., 2016), together with large sets of annotated data (Halevy et al., 2009) established new levels of maturity for natural language processing tasks in general, whose principles are largely applied in the context of natural language commands, as shown in the **De-canLP** model, which uses the same approach to deal with ten NLP tasks, including semantic parsing (McCann et al., 2018). The methods presented so far used what we now call *traditional learning models*, which comprehends linear regression, support vector machine or random forest, just to mention a few. The breakout moment of the era, however came with the popularity of the deep learning methods, in particular recurrent neural network models

and the attention mechanisms using the *encoder-decoder architecture*.

According to Kant (2018), given the *connectionist* nature of neural network approaches (Sun, 2000), neural-based approaches are not suitable to generate code properly when operating at character level. Using a variation of a recurrent neural network, Neelakantan et al. (2015) proposed a model to interpret questions over spreadsheets that require the application of built-in functions. Their contribution focuses on the induction of a compositional model for the operations. The architecture follows a combination of components able to deal with the specificities of each type of input data.

The Encoder-Decoder Architecture

The *encoder-decoder neural network architecture*, as its name suggests, is composed of two main components: an encoder responsible for learning an intermediate representation of the input data, and a decoder responsible for translating the internal representation to a meaningful output. The architecture has been applied to all sort of tasks in the parsing of natural language commands.

For instance, to address the **SAIL** navigational-instruction task, Mei et al. (2016) proposed an extra component named **aligner** to bridge the encoder and the decoder. Whereas both encoder and decoder components follow a typical implementation based on LSTM, the aligner acts as a enricher component that, in addition to the output of the encoder, also receives the input data, offering a short access to the raw features to the decoder.

A different type of improvement is regarding the feature representation. Also targeting the **SAIL** task, Can and Yuret (2018) innovates by representing the perceptual spatial state of the agent as grid, which fed the network together with the language features.

In the context of code synthesis, some tokens of the input descriptions need to be copied “*as is*”, such as when the words represent parameter values. Using a Bi-LSTM architecture with attention in the decoder, Ling et al. (2016) combine principles of pointer networks, which are capable of copying words from the input to the output together with a character-based softmax. Still in code synthesis tasks and text-to-SQL, frequently the output needs to comply with a specific syntax. As checking the correct syntax is straightforward, researchers included a checker mechanism in their decoder components. For instance, Yin and Neubig (2017) define a *syntactic neural model* to overcome the limitation of previous works in guaranteeing that the synthesised code respects the **Python** syntax. Dong and Lapata (2016), on the other hand, include a syntax control at the level of a tree, the structure used to represent the task in **GeoQuery**, **ATIS** and **IFTTT** data sets. The authors later propose the **Coarse-to-Fine** architecture where two pairs of encoder-decoder components are organised in sequence to produce both an intermediate representation, from which a second pair of encoder-decoder

components produce the expected output for the tasks, for example SQL (Dong and Lapata, 2018).

In a reinforcement learning fashion, Bunel et al. (2018) propose an intricate neural network architecture composed of LSTM and CNN layers. The main innovation of the model, however, relies on the construction of a syntax checker at the end of the model that learns valid program syntax jointly with the other components of the model. A similar strategy is used by Zhong et al. (2017), which associates a token-based network that generates SQL with a reinforcement learning component that ensures the correct syntax.

In the tasks of interpreting cards' behaviour, Ling et al. (2016) analyse machine learning baselines based on sequence-to-sequence, machine translation and the encoder-decoder architecture. Rabinovich et al. (2017) later re-engineered the data representation to make use of the **Abstract Syntax Description Language Framework** (Wang, 1997), from which an encoder-decoder architecture achieves better results.

Gardner et al. (2018) highlighted how the community has taken advantage of the modern *neural machine translation* method (Bahdanau et al., 2014), where the general idea is to follow the same principles of the previously presented WASP model, that suggests interpreting the parsing task as a translation task, but following a novel neural-network-based architecture.

Many papers analysed new data sets with popular end-to-end techniques that offer initial baselines to identify the complexity of the tasks when applying well-spread models. This is the case of Quirk et al. (2015), which applied machine translation and loosely synchronous generation in the IFTTT data set, Misra et al. (2016), playing with a model “*which is isomorphic to a conditional random field*”, Chen et al. (2019) which used the **LingUNet** architecture in the **Touchdown** data set, Misra et al. (2018) and Anderson et al. (2018) with sequence-to-sequence models to the vision-and-language navigation data set, and Barone and Sennrich (2017) using an open source tool for neural machine translation in the previously presented data set to build **Python** code from its documentation.

In most of the aforementioned works the results are empirical, without providing proper mathematical grounding.

To better distinguish the approaches based on the encoder-decoder architecture, we added additional information in Table 3.3. Mei et al. (2016), Can and Yuret (2018) and Bunel et al. (2018) differentiate their architectures by adding a new component between the encoder and the decoder. The main motivation behind this component is to better represent and “remember” parts of the information that was “forgotten” by the recurrent mechanism. For the same motivation, but using a different approach, Ling et al. (2016) and Dong and Lapata (2016) use attention mechanisms in the network. Another similarity is found in the improvements of the decoder component in the works of Dong and Lapata (2016), Zhong et al. (2017),

Yin and Neubig (2017) and Bunel et al. (2018), to guarantee the networks output the correct syntax format.

3.4 An Overview of the Methods for Parsing Natural Language Commands

To acquire a deeper understanding of the research field, apart from classifying the various approaches from the methodological perspective, we conducted a structured analysis according to their *approach, feature set, support to explainability* and *evaluation methods*.

Tables 3.4, 3.5 and 3.6 present an overview of 36 methods analysed according to the criteria aforementioned. Given their particularities, extra characteristics of the methods based on encoder-decoder machine learning models are described in Table 3.3.

3.4.1 Approach and Feature Set

Tables 3.4, 3.5 and 3.6 describe the main approach and feature set of each of the related works, disposed chronologically.

Independently on being based on rules or supported by a learning model, parsing methods need to rely on features. For a large set of methods, the feature set is a discriminator of the approach itself. Furthermore, simply changing the feature set can be the reason of a performance improvement.

The simplest semantic parsers operate merely based on string matching and slot filling. The complexity increased gradually when syntactic parsers and taggers became available, in a way that they added a new set of features based on which rules could be defined. Subsequently, the emergence of thesauri, mainly supported by semantic web standards, allowed the methods to use graph metrics in the field, when ontologies as **WordNet** (Miller, 1995), **FrameNet** (Baker et al., 1998) and **VerbNet** (Schuler, 2005) got prominent.

The popularity of statistical methods in association with some availability of training data and linguistic resources, made a significant shift in the field by enabling the generation of automated and semi-automated parsers. Grounded on the same basis, the more recent research agenda mainly dominated by neural-based approaches represents a new wave of the learning methods, now supported by larger models and much larger annotated training sets. Feature-wise, the main improvement is regarded to word embedding, which represents the meaning of tokens by a latent vector generated from large textual data sets in a non-supervised fashion.

3.4.2 Explainability

There is a rising demand in the artificial intelligence community to improve the levels of interpretability of the learning methods, not only from a tech-

Work	Encoder	Decoder	Highlights
Mei et al. (2016)	LSTM	LSTM	The architecture has an extra component between the encoder and the decoder, the Multi-level Aligner, to enrich the data input in the decoder.
Dong and Lapata (2016)	LSTM	LSTM with attention	The decoder uses attention to improve the generation of syntactically valid output.
Ling et al. (2016)	LSTM	LSTM with attention	Components are Bi-LSTMs combining pointer networks to copy words from the input to the output together with a character-based softmax.
Yin and Neubig (2017)	LSTM	LSTM adapted for AST	The decoder component is adapted to operate according to a programming language AST.
Zhong et al. (2017)	LSTM	LSTM	The decoder uses augmented pointer networks and reinforcement learning to ensure the correct output syntax.
Can and Yuret (2018)	LSTM	LSTM	An attention-based CNN component bridges the encoder and the decoder.
Bunel et al. (2018)	LSTM	LSTM and CNN	An extra LSTM network is attached at the end of the architecture to learn the program syntax and help pruning invalid outputs.
Dong and Lapata (2018)	LSTM	LSTM with attention	The architecture is composed of Bi-LSTMs with two pairs of encoder-decoder.

Table 3.3: Extra characteristics of the encoder-decoder-based machine learning models.

Work	Approach	Linguistic Features	Explanability	Gold standard evaluation	User centred evaluation
Guida and Tasso (1982)	rule-based	lexicon	potentially	no	no
Biermann et al. (1983)	rule-based	lexicon	potentially	no	yes
Maas and Suppes (1985)	rule-based	lexicon	potentially	no	no
Manaris et al. (1994)	rule-based and grammar formalism	lexicon	potentially	no	no
Price et al. (2000)	rule-based	lexicon	potentially	no	no
Tang and Mooney (2001)	rule-based and logical forms	lexicon	potentially	yes	no
Zettlemoyer and Collins (2005)	rule-based with probabilistic model	lexicon and categories	potentially*	yes	no
Wong and Mooney (2006)	statistical machine translation	lexicon	no	yes	no
MacMahon et al. (2006)	rule-based	lexicon, pos-tag and dependency tree	potentially	yes	no
Englmeier et al. (2006)	rule-based with semantic web patterns	lexicon and ontology	potentially	no	no
Bosca et al. (2006)	rule-based with semantic web patterns and similarity measures	lexicon and ontology	potentially	no	no
Little and Miller (2006)	rule-based	lexicon	mentioned	no	yes
Wong and Mooney (2007)	grammar formalism with a maximum-entropy model	lexicon	no	yes	no
Branavan et al. (2009)	reinforcement learning	lexicon and edit distance	no	yes	no
Lim and Lee (2010)	rule-based with semantic web pattern sand similarity measures	lexicon and ontology	potentially	yes	no
Kwiatkowski et al. (2010)	Probabilistic CCG	lexicon and categories	potentially*	yes	no

Table 3.4: Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 1).

Work	Approach	Linguistic Features	Explanability	Gold standard evaluation	User centred evaluation
Tellex et al. (2011)	Statistical model (L-BFGS algorithm)	lexicon, ontology and similarity metrics	no	yes	no
Giordani and Moschitti (2012)	rule-based with SVM	lexicon and weight schema	no	yes	no
Artzi and Zettlemoyer (2013)	rule-based with probabilistic model	lexicon and categories	potentially*	yes	no
Gulwani and Marron (2014)	rule-based	lexicon and data schema	potentially	yes	no
Li and Jagadish (2014)	rule-based	lexicon, pos-tag and dependency tree	potentially	yes	no
Gvero and Kuncak (2015)	rule-based with statistical Model (PCFG)	lexicon	potentially	yes	no
Hemachandra et al. (2015)	Statistical model (RaoBlackwell theorem)	lexicon	no	yes	no
Wang et al. (2015)	Statistical model (AdaGrad)	lexicon / category	no	yes	no
Quirk et al. (2015)	Statistical model (log-linear probabilistic model)	lexicon	no	yes	no
Neelakantan et al. (2015)	RNN and other probabilistic models	word embeddings	no	yes	no
Beltagy and Quirk (2016)	feed-forward neural networks	word and character n-grams and Brown cluster	no	yes	no
Liu et al. (2016)	latent attention in neural networks	one-hot vector	no	yes	no
Mei et al. (2016)	recurrent neural networks	one-hot vector	no	yes	no
Dong and Lapata (2016)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no

Table 3.5: Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 2).

Work	Approach	Linguistic Features	Explanability	Gold standard evaluation	User centred evaluation
Ling et al. (2016)	LSTM-based encoder-decoder neural network	C2W-based embedding vector	no	yes	no
Yin and Neubig (2017)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no
Zhong et al. (2017)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no
Can and Yuret (2018)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no
Bunel et al. (2018)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no
Dong and Lapata (2018)	LSTM-based encoder-decoder neural network	one-hot vector	no	yes	no

Table 3.6: Overview of the methods so far discussed. Potentially explainable methods marked with *, explanation is applied only to part of the methods (Part 3).

nical perspective, but also focusing on the end users. In this sense, there is an expectation that new models, in addition to delivering better performance, also bring mechanisms to allow non-technical users to understand their rationale.

This pressure, however, is not clearly reflected in the related work. Among the research present in our survey, only the work of Little and Miller (2006) delivers explainability as a native feature in the parsing of natural language commands. However, by using criteria like the ones included in the classification of Lipton (2016) (see Section 2.5), some approaches can be labelled as *potentially explainable*, since, although the authors didn't address explainability in their research, the proposed methods offer a straightforward method of explanation. For more on explainability, please refer to Section 2.5.

3.4.3 Evaluation

With a few exceptions from the 80s, works are always somehow evaluated, usually by either a gold standard test collection, that represents the majority of the cases, or based on a user experiment, as the research of Biermann et al. (1983) and Little and Miller (2006). Gold standard test collections guarantee easier reproducibility, using generally well-known metrics such as precision, recall and accuracy, depending on the task. User-centred evaluation, however, can bring a subjective evaluation from a practical point of view, which is especially valuable when assessing the explainability aspect.

3.5 Exposing the Gap

Provided a large enough data set is available, end-to-end encoder-decoder models have been showing high performance in several tasks. However, such a data set is not always available.

Attempts to overcome this restriction with the generation of synthetic data as in the work of Neelakantan et al. (2015) leads to undesirable biases as language expressivity is much higher than its examples in the train and test samples, reducing the real complexity of the task (Shin et al., 2019). While synthetic data sets can serve to analyse theoretical limitations and properties of a learning method (Bermeitinger et al., 2019), their benefit in learning how to decode user language is very limited. We can conclude intuitively that a learning process trained on a synthetically generated data set is in fact modelling the generation function behind the synthesis process.

Our thesis focuses on studying the development of a semantic parser for a task with a restricted amount of training data. Given the implications, as a requirement, our approach does not rely on synthetically generated samples to expand the training data. Additionally, we also aim to produce an

explanation model giving special attention to study the human factors regarding preferences on the type of explanations considering the background of the user, in addition to analyse the user perception on the importance of explainability with regard to the performance.

3.6 Summary

Parsing of natural language commands is an active task dating back to the origins of research in computer science. Throughout the time, the field has evolved from rule-based approaches, mainly focused on string matching, to more sophisticated learning models using modern neural network architectures.

However, newer models come with a strong requirement: the task needs to offer a reasonable large training set from which a neural model can learn. This requirement, however, in many circumstances cannot be achieved, given particular difficulties and the high costs that data collecting has. A common approach to overcome this limitation is the generation of synthetic data. The technique, however, tends to only repeat a known pattern behind the data generator method, which frequently doesn't represent in large extent the complexity of the task and the broadness of the user representation.

The analysis of the literature reveals a gap in methods that can learn from small data sets, whose complexity are high, without relying on synthesising data, given the already proved high risks of bias. This forms the base of the method we propose in the coming chapters.

Part III

Designing an Explainable Semantic Parser

A Semantic Parser for End-User Development

Contents

4.1	Introduction	61
4.2	Re-engineering the Learning Task	63
4.3	Componentising the Semantic Parsing	64
4.4	The Proposed Architecture	66
4.4.1	Command Segmenter	69
4.4.2	Shallow Parser	70
4.4.3	Type Inferencer	71
4.4.4	Predicate-Argument Structure	73
4.4.5	API Filter	73
4.4.6	Function Call Generator	75
4.4.7	Intent Classifier	77
4.4.8	Ranker	77
4.5	Summary	78

4.1 Introduction

By surveying the field, Chapter 3 presented an overview of the current state-of-the-art architecture for the semantic parsing of natural language commands. Typically, this architecture is defined by an end-to-end encoder-decoder-based machine learning model that interprets the task as a translation from *natural language commands* to *function calls*. Figure 4.1 depicts the structure of such architecture, presenting its main components and the relationship between them.

In our research, the first goal is to analyse the performance of end-to-end encoder-decoder architectures in our target data set. As further detailed in Chapter 6, several models based on this architecture do not succeed to solve the target problem due to the small training set available. Our experiments evaluated variations of the end-to-end networks considering different types of input data and variations in the architecture as using attention mechanisms (Vaswani et al., 2017). All the evaluated models delivered an f1-score of 0.

Considering our target task, solving the problem using an end-to-end approach is especially challenging because of its expected output. In addition to identifying the correct function signature, the model needs to simultaneously select chunks of tokens from the input text, and associate them to the correct set of parameters. Our thesis concentrates on finding an alternative approach capable of dealing with the given task, assuming the small size of the data set as a strong requirement.

Following this introductory section, Section 4.2 briefly analysis the structure of the end-to-end learning model in terms of output complexity and suggests re-engineering the task to simplify the learning goal. With this change, the small training data available will be used exclusively to determine to which degree a given function call represents the intent of a natural language command. Changing the learning task comes with the price of adding additional components in the new proposed approach, whose motivation and role are presented in Section 4.3. Section 4.4 describes our proposed architecture, defined in a multi-component fashion, where the new learning task plays a central role in the final ranking function. Finally, Section 4.5 summarises this chapter, highlighting the main aspects of our

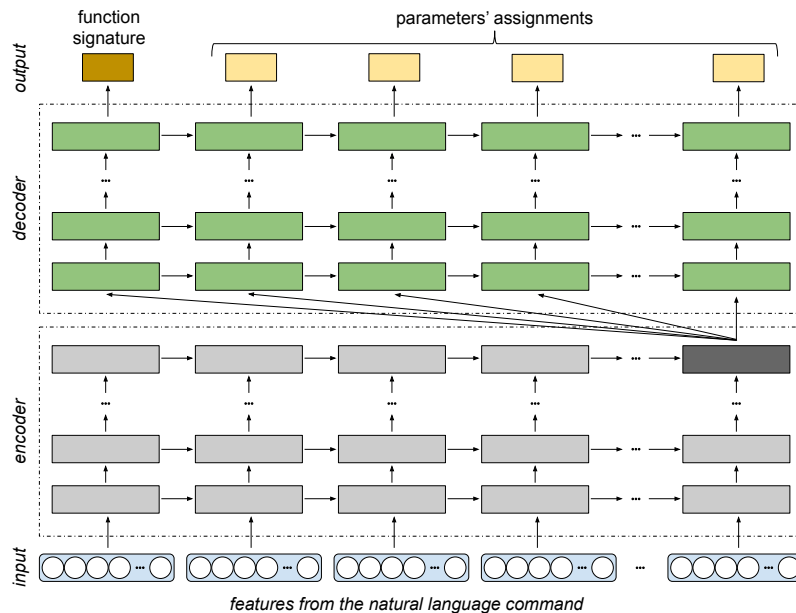


Figure 4.1: A typical method to solve this task is the encoder-decoder machine learning model in the sequence-to-sequence fashion (Sutskever et al., 2014; Ling et al., 2016; Gulcehre et al., 2016). The grey and green boxes represent LSTM cells and the *parameters' assignments* are one-hot vectors associating chunks of the input text to the parameters.

approach.

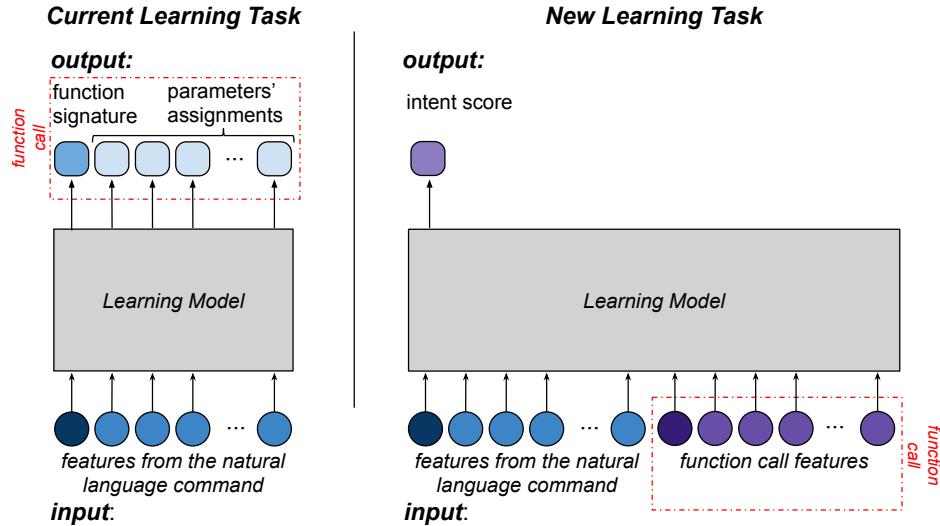


Figure 4.2: The current learning task receives as input a set of features from the *natural language command*, whereas the output defines the intent *function signature* together with their *parameters' values*. On its turn, the new learning task represents a model that receives as input features a pair of a natural language command and a function call, outputting a score that defines the level of representativeness of the user intent ranging from 0 to 3 whose meaning is defined in Table 4.1.

4.2 Re-engineering the Learning Task

To overcome the restriction on the size of the training data, the main strategy is to break down the solution, dividing the responsibilities among smaller components. Thus, based on the intuition that the smaller the training data, the simpler the learning task (Halevy et al., 2009), the data set is used to learn a more targeted piece of information. Hence, instead of inducing the function call directly, predicting at once both the function signature and its parameters' values, as it is stated now, in the new approach we re-engineer the task to classify the function calls, letting the new model be responsible for evaluating them as representative or not of the user intent.

Figure 4.2 compares the input and output data of the *current learning task* to the *new remodelled learning task*. For instance, let the excerpt below be a natural language command:

“Write to newton@example.com asking him to take a look at

the NYT today”

In the first task our learning model receives as input the features from this natural language command aiming at producing the following intended function call, where `send` and `email` is a function signature defined in the API Knowledge Base, and both `message` and `to_address` are parameters associated to this function signature:

```
function name: send an email
params: message = "take a look at the NYT today"
        to_address = "newton@example.com"
```

The proposed learning task, however, operates at the level of function calls, which, together with the natural language command, become the input of the learning model, where the expected output is a score denoting the degree the given function call represents the user intent of the given natural language command.

The change in the structure of the task has two advantages. First, it simplifies the object of learning, from a large set of variables, whose range of the function signature for example is at the scale of thousands, to a single output ranging from 0 to 3 as detailed in Table 4.1. Secondly, this approach increases the training data as we need to generate candidates of function calls to train the model.

Intent Score	Meaning
0	<i>wrong</i> function signature
1	<i>right</i> function signature with <i>wrong</i> parameters
2	<i>right</i> function signature with <i>partially right</i> parameters
3	<i>right</i> function signature with <i>right</i> parameters

Table 4.1: Range of *intent scores* and their respective meaning.

For example, given the natural language command “*Exchange 1000 Chilean Pesos to Euro*”, the new approach first generates a set of function call candidates, as depicted in Table 4.2, and then classifies each of them according to its relevance represented by an intent score, which is a central feature in the final ranking method. This new approach, however, demands the support of extra components able to generate the set of function call candidates from the natural language command and API Knowledge Base.

4.3 Componentising the Semantic Parsing

The main characteristics of a complex system are the multiplicity and heterogeneity of its components. In the context of computational semantics,

building such systems requires the coordination of different NLP components frequently organised as a chain, whose integration often depends on different tools and resources of less granularity, such as language resources and structured knowledge bases (Sales et al., 2018a; Desolda et al., 2017).

The purpose of the componentisation is to support the re-engineering of the learning task, reserving the available training data to determine the intent score of a function call in relation to the natural language command.

Essentially, the componentisation aims at defining the semantic parser as a pipeline composed of a specialised shallow parser that produces a predicate-argument structure from the natural language command, followed by a ranking model based on a distributional model. This architecture takes advantage of external resources to shallow parse the command, enriching their compound part with additional semantic data, before matching to a compatible function signature. For example, tokens of the natural language command that represent a named entity are labelled as such. Other tokens that represent the description of the target function are also highlighted as such. Those cases of labelling can be performed by the composition of a grammar parser and a thesaurus, as shown in the coming sections. This strategy guarantees that the final learning task has more features to classify the degree of representativeness of the function calls.

Figure 4.3 illustrates, at the bottom of the pyramid, resources that store

Function Calls	Intent Score
Convert File [<i>file = Chilean_Pesos, output_format = Euro</i>]	0
Make a Payment [<i>invoice = 1000, method = Chilean_Pesos</i>]	0
Currency Convert [<i>from_amount = Chilean_Pesos, from = Euro, to = 1000</i>]	1
Currency Convert [<i>from_amount = Euro, from = Chilean_Pesos, to = 1000</i>]	2
Currency Convert [<i>from_amount = 1000, from = Chilean_Pesos, to = Euro</i>]	3

Table 4.2: List of possible function calls for the command “*Exchange 1000 Chilean Pesos to Euro*”, where the last row represents the intent score from the user perspective. The calls are generated considering a selected set of function signatures, which can be classified according to its intent score as *wrong function signature* (0), *right function signature with wrong parameters* (1), *right function signature with partially right parameters* (2) or *right function signature with right parameters* (3).

both linguistic and common-sense knowledge, such as manually built thesauri like **WordNet** (Miller, 1995) and **VerbNet** (Schuler, 2005), the knowledge base **DBpedia** (Auer et al., 2007) and word embedding models (Camacho-Collados and Pilehvar, 2018). Under this umbrella, there are also linguistic tools trained from manually curated data such as POS taggers (Manning, 2011) and dependency parsers (Bohnet, 2010).

At the next level, the pyramid has the list of low-level components, which are built from the composition of those linguistic tools and common-sense knowledge to address fundamental tasks of natural language processing such as *named entity recognition* and *semantic relatedness*. Finally, at the top of the pyramid, the complex multi-component semantic system emerges from the coordination of the seven coarser-grained components, which are also built from the elements presented in lower levels.

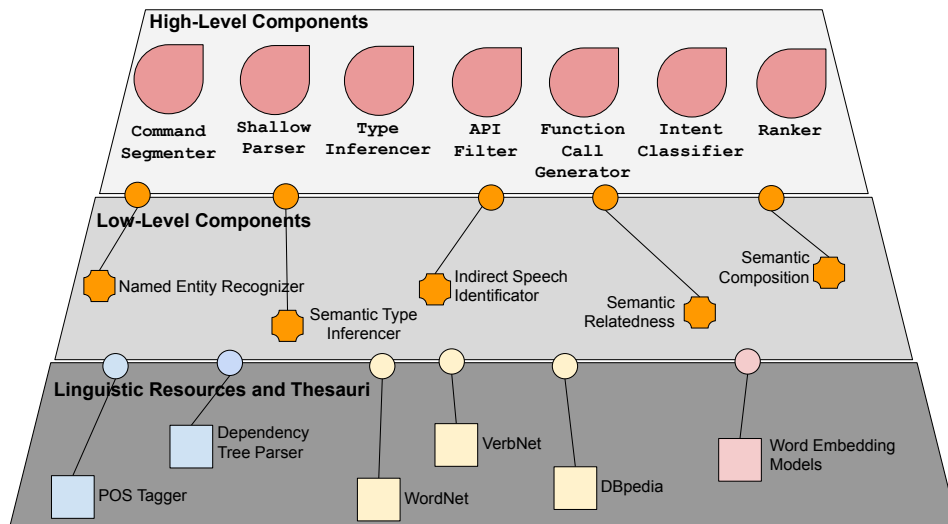


Figure 4.3: The top level of the pyramid shows the high-level components, which represent the main elements from which our proposed architecture is built. Immediately below, there are the low-level components, representing some fundamental NLP tasks, such as named entity recognition and semantic relatedness. At the bottom, we have linguistic tools and thesauri of common-sense knowledge.

4.4 The Proposed Architecture

Our proposed architecture is composed of seven main components, which are developed as a composition of lower-grained natural language models and linguistic resources. Figure 4.4 depicts the execution pipeline of our proposed architecture, showing a functional example of a natural language

command in the form of an *if-then* recipe. The first component in the pipeline is the **Command Segmenter**, responsible for splitting the *protasis*, i.e. the conditional clause, and the *apodosis*, i.e. the main clause. As described in Section 1.3, commands can appear in the form of an *if-then* recipe or not. When only one single action is submitted, the **Command Segmenter** just forwards the command. From now onwards, each sentence of the command is analysed individually. The second component, the **Shallow Parser**, identifies the *function descriptor* and the set of *command objects*, constituting a lightweight representation of the natural language command in the form of a predicate-argument structure. Before serving as input for the **API Filter** to identify the relevant set of function signatures from the **API Knowledge Base**, the predicate-argument structure is enriched with semantic types by the **Type Inferencer**. Next, the **Function Call Generator** analyses the features generated so far to generate a set of potential function calls, which, together with the predicate-argument structure serves as input for the **Intent Classifier** to obtain the intent score. In the final step, the **Ranker** component lists to the final user the most likely set of function calls.

Considering the formal description of the task of semantic parsing of natural language commands in Section 1.3, the components can be formalised as:

1. **Command Segmenter:** In the first step, represented by Equation 4.1, the model identifies whether the natural language command (c) is defined in the form of an *if-then* recipe, to split it into two sentences ($c_s \mid s \in [0, 1]$);
2. **Shallow Parser:** In the second step, represented by Equation 4.2, for each sentence, the model reduces the natural language command (c_s) to a predicate-argument structure composed of an *function descriptor* (d) and a *set of command objects* (o_1, \dots, o_k);
3. **Type Inferencer:** Then, each command object is analysed by an inference type function (Equation 4.3) that, associate it with a semantic type when appropriate. For example, $\delta(\text{"dollar"}) = \text{CURRENCY}$ and $\delta(\text{"john@domain.com"}) = \text{EMAIL}$;
4. **API Filter:** Next, the **API Filter**, represented by Equation 4.4, selects a set of function signatures \hat{F} , where $\hat{F} \subset F$, $|\hat{F}| \ll |F|$ and $(\forall \hat{f} \in \hat{F} \mid \hat{f} \approx c_s)$, in the sense that the cardinality of the selected subset is significantly smaller than the original one, and those elements are semantically related to the natural language command. This step reduces the search space, maximising the probability of matching the parameters;
5. **Function Call Generation:** Further, the selected set of function signatures \hat{F} is combined with the *command objects* (o_1, \dots, o_k) to

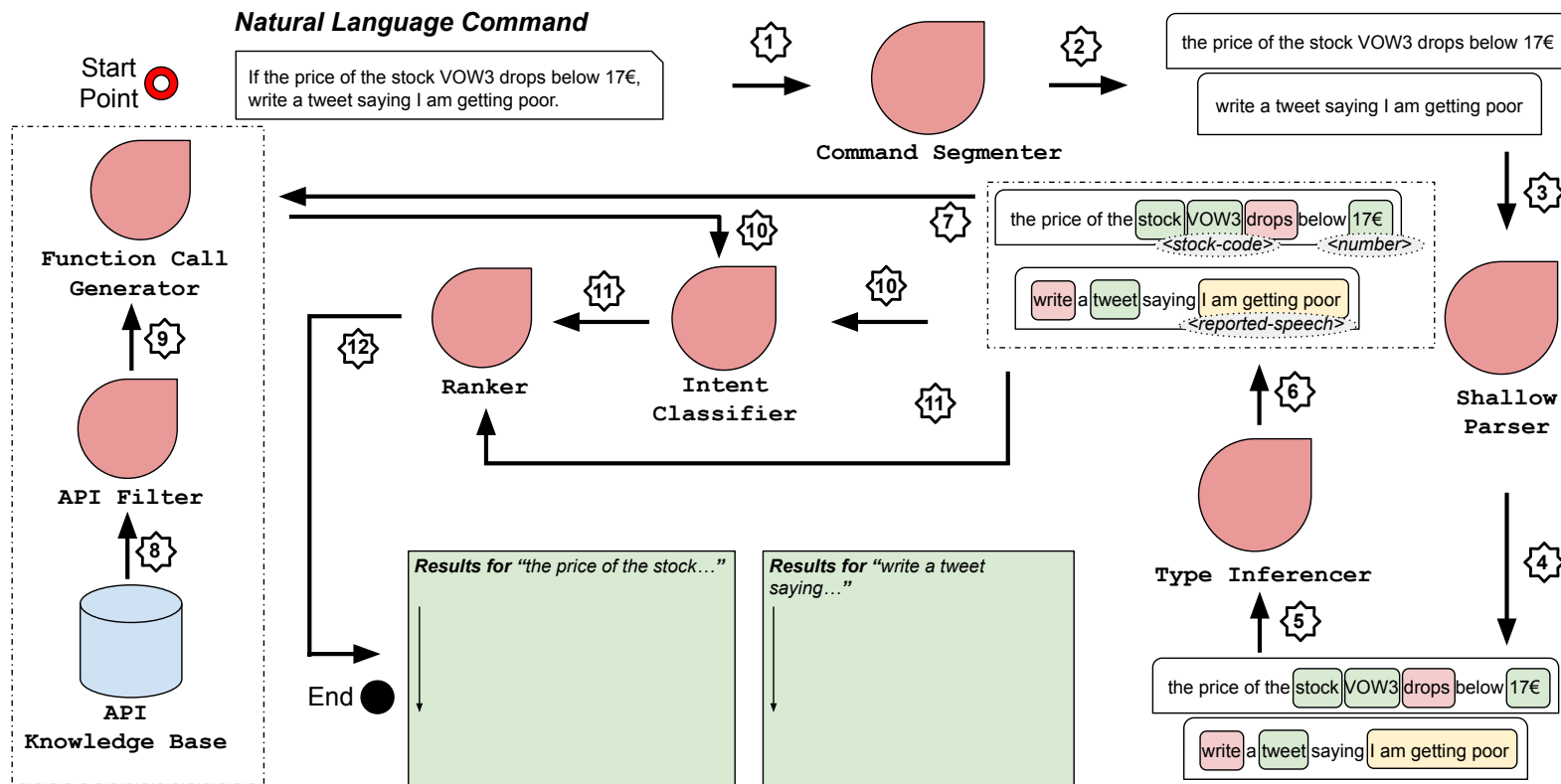


Figure 4.4: The architecture, from a software perspective, of the proposed semantic system to interpret natural language commands.

generate the set of function calls, representing them according to the feature set described in Section 4.4.6;

6. **Intent Classifier:** Then, the function calls are classified according to their representativeness of the user intent in line with the scores defined in Table 4.1, which is represented in Equation 4.6;
7. **Ranker:** Our semantic parser ends with the ranking function that produces the ranking score from the features generated in the previous steps, listing the set of relevant function calls to the final user. The **Ranker** is represented by Equation 4.7, which is defined in Section 4.4.8.

The seven components are formalised in the following equations:

$$\kappa(c) = [c_s \mid s \in [0, 1]] \quad (4.1)$$

$$\sigma(c_s) = (d, o_1, \dots, o_k) \quad (4.2)$$

$$\delta(o_1, \dots, o_k) = (t_1, \dots, t_k) \quad (4.3)$$

$$\rho(F, d, o_1, \dots, o_k) = \hat{F} \quad (4.4)$$

$$features(d, O, \hat{F}) = Z \quad (4.5)$$

$$classify(Z) = \varkappa \quad (4.6)$$

4.4.1 Command Segmenter

The **Command Segmenter** aims at identifying and splitting compound sentences that are mapped to more than one function signature in the **API Knowledge Base**. In our target data set, it represents the case of *if-then* clauses as in the example shown in Figure 4.4. Our implementation follows a method based on the context-free grammar described in Figure 4.5 defined by Weigelt et al. (2018), which operates on chunk tags from a grammar parser, where *if-keywords* and *then-keywords* come from a curated list of terms.

conditional \rightarrow **if-clause then-clause**
if-clause \rightarrow *if-keyword* NP VP
then-clause \rightarrow *then-keyword* NP VP
then-clause \rightarrow *then-keyword* VP
then-clause \rightarrow NP VP
then-clause \rightarrow VP

Figure 4.5: Adapted context-free grammar from Weigelt et al. (2018) describing the basic structures, where *NP* and *VP* stand respectively for noun phrase and verb phrase.

4.4.2 Shallow Parser

The **Shallow Parser** is responsible for identifying the (i) *function descriptor* and (ii) the set of *command objects* based on an approach that considers the syntactic functions of the constituents of the natural language command and the presence of named entities. The *function descriptor* is the minimal subset of tokens present in the command that allows identifying the target function signature in the **API Knowledge Base**. Figure 4.4 shows the output of the **Shallow Parser**, where the function descriptors are highlighted in pink, while the command objects are green and yellow. A *command object* represents a potential descriptor or value of a parameter. For example, in the conditional command “*if the price of the stock VOW3 drops below 17€*”, “*drops*” acts as a function descriptor, whereas “*stock*”, “*VOW3*” and “*17€*” represents the set of command objects. Equation 4.2 defines $\sigma(c_s)$, where c_s is the natural language command, d is the *function descriptor* and o_1, \dots, o_k are the *command objects*.

We implemented the **Shallow Parser** based on an explicit grammar defined by dependency relations and POS-tags. Table 4.3 shows the rules used to identify the command objects.

Be a dependency tree defined as a tuple $H = (V, E, \phi)$, where:

- V is a finite set of nodes each of them representing tokens;
- $E \subseteq V \times V$ is a finite set of edges, where e_{origin} represents the node in the origin and e_{dest} the node in the destination of an edge $e \in E$;
- $\phi : E \rightarrow C$ assigns a label from C to each edge.

In addition to the rules described in Table 4.3, chains of tokens that comply with the POS-tag patterns $(JJ) * (NNP)^+$ and $(JJ) * NN$, are also included in the set of command objects, where JJ , NNP and NN represent respectively adjectives, proper nouns and nouns.

The extraction of the *function descriptor* is also dependent on linguistic features, which is summarised in Algorithm 1.

Condition	CO	Qualifier
$\phi(e) = (poss \mid amod) \wedge \tau(e_{origin}) \neq PRP$	e_{origin}	e_{dest}
$\phi(e) = (nn) \wedge \tau(e_{dest}) \neq NNP \wedge \tau(e_{origin}) \neq PRP$	e_{origin}	e_{dest}
$\phi(e) = (nsubj) \wedge \tau(e_{origin}) \neq PRP$	e_{origin}	e_{dest}
$\phi(e) = (pobj \mid aposs) \wedge \tau(e_{origin}) \neq PRP$	e_{dest}	e_{origin}

Table 4.3: Tree-based rules to identify command objects (CO) and their respective qualifiers, where τ represents the POS-tag of the element $e \in V$.

To illustrate, Table 4.4 shows examples of natural language commands and their representation in the form of a predicate-argument structure composed of the function descriptor and the set of command objects.

In addition to identifying the command objects, the *Shallow Parser* can also associate a *qualifier* to them, which serves as an expanded string to identify the matching between a command object and its parameter. For instance, in the command “*Exchange 1000 Chilean Pesos to Euro*”, *Chilean Pesos* and *Euro* can only be differentiated in the sentence if we consider the preposition *to*. The qualifiers are defined concomitantly with the command objects themselves according to the rules described in Table 4.3.

4.4.3 Type Inferencer

For each command object, we also associate a *semantic type*, which is assigned by a named entity recogniser. Additionally, in our context, *reported speeches* frequently act as the *content* or *message* of a command, as in the example “*write a tweet saying I am getting poor*”, whose expression “*I am getting poor*” is expected to be associated to the message.

Algorithm 1 Extracting the function descriptor, where τ represents the POS-tag of the element $e \in V$

```

1: procedure GETFUNCTIONDESCRIPTOR
2:    $q \leftarrow$  natural language command
3:    $S \leftarrow$  tokens present in command objects
4:    $I \leftarrow$  [‘TO’, ‘IN’, ‘.’]
5:    $T \leftarrow$  tokenize( $q$ )
6:    $r \leftarrow$  “”
7:   for  $t \in T$  do
8:     if  $\tau(t) \notin I \wedge (t \notin S \vee \delta(t) = \emptyset)$  then
9:        $r \leftarrow r + “ ” + t$ 
   return  $r$ 

```

Natural Language Command	
Exchange 1000 Chilean Pesos to Euro	
Send file.doc to sandra@mail.com	
Find an image of the Sputnik-1 on Flickr	
Descriptor	Set of Command Objects
<i>Exchange</i>	(1000, Chilean Pesos, Euro)
<i>Send</i>	(file.doc, sandra@mail.com)
<i>Find image</i>	(image, Sputnik-1, Flickr)

Table 4.4: Examples of natural language commands with their respective function descriptors (d) and set of command objects (o_1, o_2, \dots, o_k) expected to be produced by σ .

Reported Speech Detector

We developed a reported speech identification mechanism based on the technique proposed by Krestel et al. (2008), which operates in two execution cases. The first represents the trivial case when part of the command is quoted. As quotation doesn't always represent a reported speech, it is later analysed by the **Named Entity Recogniser (NER)** to identify whether its content represents another semantic type.

The second case uses Krestel et al. (2008)'s strategy, which, similarly to the conditional detection from Weigelt et al. (2018), is centred on a set of curated keywords: the *reported verbs*. In its mechanism, after identifying a reported verb, the detector analyses the components of the sentence considering its *rection*, i.e. the relationship between a verb and its dependants. For instance, given the command:

“write a tweet saying I am getting poor”

After identifying the reported verb *say*, according to its verb rection, the **Reported Speech Detector** can extract the reported speech by collecting the *reduced non-finite verbal modifier*, which is identified by a dependency grammar parser, corresponding in this example to the expression *“I am getting poor”*. Table 4.5 lists the reported verbs supported by Krestel et al. (2008)'s approach.

Named Entity Recognition

Typical named entity recognisers usually identify people, organisations and places in texts. More comprehensive recognisers also tag dates and expressions of time (Nadeau and Sekine, 2007). In addition to the identification of those common types, our domain demands specific types, such as *currency*, *user* and *email*.

For this reason, we developed our own recogniser, whose implementation combines heuristics with a gazetteer mainly fed by DBpedia instances, where, given a command, it searches for the longest chain of tokens that maps to an element in the gazetteer, ignoring those tokens that are part of a reported speech.

Table 4.6 lists examples of types of entities identified by our component. Types marked with one asterisk are identified by regular expression, while the others use the NER. Types marked with double asterisks are identified by the Reported Speech Detector.

4.4.4 Predicate-Argument Structure

From this step onward, the commands are represented by predicate-argument structures composed of the function description and the set of command objects together with their semantic types and qualifiers, whose foremost goal is to help to measure the semantic relatedness of a given command and a function call. At this level, the challenge relies on calculating the degree of representativeness of the function signature with regard to the natural language command. Figure 4.6 shows the predicate-argument structure of the command “*Send file.doc to sandra@mail.com*”.

Our strategy aims at projecting both commands and function signatures on two semantic hyperspaces, where the first represents a general projection of the commands and function signatures, whereas the second represents a hyperspace for the command objects and parameters.

4.4.5 API Filter

The goal of the filter function ρ (Equation 4.4) is to restrict the set of function signatures to those semantically relevant for the natural language

according	blame	decline	mention	stress
accuse	charge	deny	note	suggest
acknowledge	cite	describe	order	tell
add	claim	disagree	predict	testify
admit	complain	disclose	promise	think
agree	concede	estimate	recall	urge
allege	conclude	explain	recommend	warn
announce	confirm	fear	reply	worry
argue	contend	hope	report	write
assert	criticise	insist	say	observe
believe	declare	maintain	state	

Table 4.5: The list of reported verbs supported by the approach designed by Krestel et al. (2008).

Location	Day*	Music	User*
Product	Person	Telephone Number*	Hashtag*
Language	Email*	Currency	File*
Time*	Sports Team	Number*	Reported Speech**

Table 4.6: Example of semantic types identified by the Named Entity Recogniser. Types marked with an asterisk are identified purely by regular expression, whereas those with double asterisks are identified by the Reported Speech Detector.

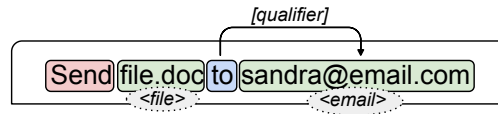


Figure 4.6: A natural language command, where “Send” represents the *function description*, and both “file.doc” and “sandra@email.com” are command objects, whose semantic types are respectively $\langle \text{file} \rangle$ and $\langle \text{email} \rangle$.

command. It is meant to reduce the complexity in generating and analysing function calls and avoiding adding more noise in the final ranking system.

The filter function operates in the **Predicate Hyperspace**, which is defined by a word embedding model on which both the predicate-argument structure and the function signatures from the **API Knowledge Base** are projected as depicted in Figure 4.7. The command and the function signatures are each represented as one vector, from which a semantic relatedness measure can be calculated.

The rationale behind this approach relies on a broader notion of semantic relatedness. For instance, identifying that the words *problem* and *issue* may have equivalent meanings in some contexts, even when these words are not related to each other in a thesaurus, is a practical example of the role that semantic approximation plays in the proposed system. Such approximation ability seeks to overcome the divergence between the vocabulary used by the user and that used to describe the function signatures in the **API Knowledge Base**, known as the *problem of vocabulary mismatch* as discussed in Section 3.3.1. Word embedding provides a mechanism to address this problem (Turney and Pantel, 2010), where a semantic relatedness metric can be calculated from a geometric measure.

The filter function defines the set of relevant function signatures (\hat{F}) by calculating a geometric measure in the hyperspace. The vector representation for both commands and function signatures are defined as a composition of the individual token’s vectors. The representation of the function signature considers its name, provider and parameters’ names, whereas the

command considers the natural language query and the semantic types from the command objects.

The filtering function is defined as a distance measure in a nearest neighbours fashion, where the n closest functions are selected. Figure 4.7 shows a simulated representation in two dimensions of the Predicate Hyperspace where a command and a set of function signatures are depicted.

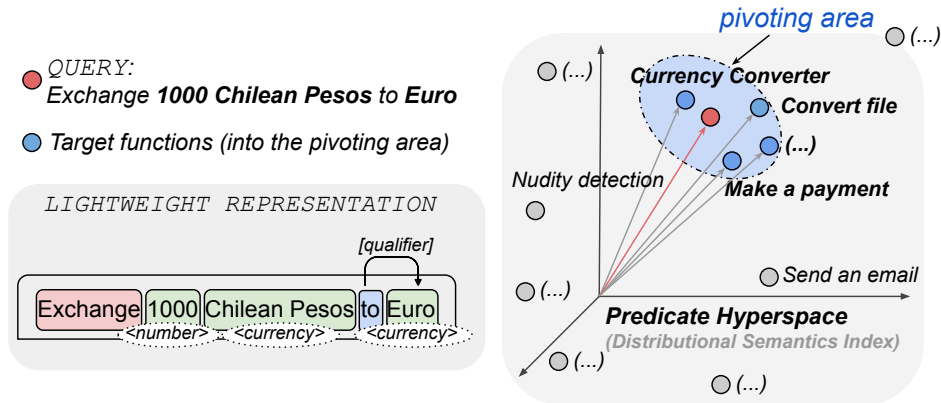


Figure 4.7: The natural language command “Exchange 1000 Chilean Pesos to Euro” is projected on the Predicate Hyperspace in which the set of function signatures is also represented. The pivoting area, represented by a blue ellipsis, determines the set of relevant function signatures for the given command.

4.4.6 Function Call Generator

Based on the predicate-argument structure, the model generates potential function calls by combining the set of command objects and the list of function signatures. For each of the selected set of relevant function signatures, we instantiate a Predicate Hyperspace on where the *command objects* and the *function parameters* are also projected based on the semantic representation of their names, qualifiers and types. The semantic representation is again generated from a word embedding model, using the same compositional method defined for the function signature in the previous hyperspace. Based on this projection, our model generates the features of the set of potential function calls, which are listed in the bullet points below. Each function call candidate corresponds to a certain combination of *function parameters* and *command objects*. Considering the relation *many-to-many* between *function parameters* (i) and *command objects* (j), each pair of function signature-command generates a set of function calls resulting from the permutation iP_j . Table 4.2 shows examples of generated function calls.

Figure 4.8 depicts the simulated example in two dimensions considering

the set of relevant function signatures defined by the *pivoting area* in Figure 4.7.

Aligning parameters and generating features

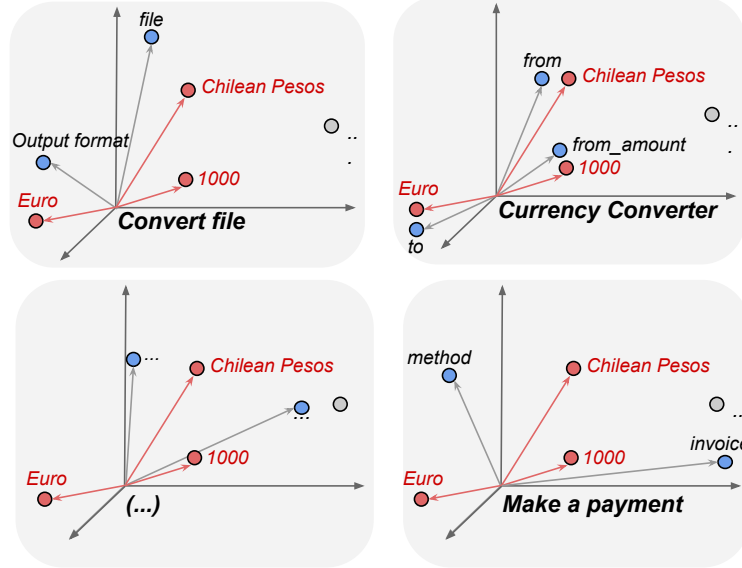


Figure 4.8: The natural language command “Exchange 1000 Chilean Pesos to Euro” is parsed to generate a predicate-argument structure, whose command objects are then projected on **Predicate Hyperspaces**. Each hyperspace represents a function signature selected by the **API Filter**, on which the function parameters are also projected.

The model represents each function call by the following list of features:

- $\cos(\vec{d}, \vec{n})$: The semantic relatedness between the *function description* (d) and the *function name* (n);
- $\max_{0 \leq j \leq m} \cos(\vec{o}_j^{literal}, \vec{l})$: The maximum semantic relatedness between the command objects (o) and the provider (l);
- $\cos(\vec{o}_j^{literal}, \vec{p}_i)$: The set of semantic relatedness between the pairs composed of the command objects (o) and function parameters (p);
- $\cos(\vec{o}_j^{type}, \vec{p}_i)$: The set of semantic relatedness between the pairs composed of the semantic type of the command object (o) and the function parameter (p);
- $den(p_i)$: The set of the densities of the function parameters, which represents the inverse term-frequency in the function signature’s vocabulary set.

These semantic relatedness scores are used as input features to identify jointly the most relevant function signature and the best configuration of parameters' values.

4.4.7 Intent Classifier

The role of the **Intent Classifier** is to measure, according to the scale below described, whether a given function call represents the intent of the natural language command. For each function call, the **Intent Classifier** provides a classification as:

- (i) wrong function signature (score 0);
- (ii) right function signature with wrong parameters (score 1);
- (iii) right function signature with partially right parameters (score 2);
- (iv) right function signature with right parameters (score 3).

This small, but discriminative set of classes works as a data augmentation method, as it enables the existence of many training instances of the same class, even considering the small training data set the task offers.

Our proposed architecture defines both the input feature set and the expected output of the **Intent Classifier**, but allowing the instantiation of different types of learning methods to build it.

In our experiment settings, we assess the performance of different classifiers, such as those based on neural networks, random forest or support vector machines, as described in Chapter 6.

4.4.8 Ranker

Equation 4.7 defines the ranking score function (rs), where c represents the natural language command, f represents a function signature, \odot represents a given combination of parameters and command objects defined in the function call, and α represents a large number to guarantee function calls with a higher intent score (\varkappa) are always ranked above those with lower scores.

$$rs(c, f) = \cos(\vec{n}, \vec{d}) + \max_{j=1}^k (\cos(l, o_j)) + \sum_{i=1, j=1}^{n, k} \odot (\cos(\vec{p}_i, \delta(\vec{o}_j))) + (\alpha * \varkappa) \quad (4.7)$$

In simple terms, the ranker function is defined to guarantee that function calls are sorted first by its intent scores and then by the sum of their features.

4.5 Summary

The restriction on the size of the training data prevents us to use an end-to-end approach, where a unique machine learning model is able to identify the function signature, as well as simultaneously selecting chunks of tokens from the input text to associate them with the correct set of parameters. As an alternative, we propose a multi-component system able to restructure the task to take advantage of the training data.

Our semantic parsing approach relies on the notion of predicate-argument embedding, where functions and parameters are embedded in distributional vector spaces, in which geometric operators such as distance and density are used to measure the semantic relatedness between a predicate-argument structure of the command and the function calls.

The training data is mostly applied to build a classifier able to provide an intent score, which together with other semantic features serves as input for a ranking model to sort the relevant function calls according to their likelihood to represent the user intent.

In Chapter 6, we evaluate and discuss variations of this architecture to evaluate the contribution of each component to the final result, comparing them to a set of baselines. Before this evaluation, we present in Chapter 5 an explanation model for the proposed semantic parser, setting the bases for an additional user-centred study presented in Chapter 7.

Explanation Model of a Multi-Component Semantic Parser

Contents

5.1	Introduction	79
5.2	Levels of Interpretability	80
5.3	The Explanation Model	81
5.3.1	Shallow Parser Rules & Syntactic Tree Layers	81
5.3.2	Word Embedding Layer	83
5.3.3	The Ranking & Classification Layer	83
5.3.4	The Comparative Explanation	85
5.4	Summary	85

5.1 Introduction

Aligned with our proposed approach, a large gamut of natural language understanding systems require the complex coordination of multiple components, *e.g.* POS-tagger, syntactic parsing, named entity recogniser and task-oriented machine learning models, where each component explores a large spectrum of resources and learning methods (Burgess, 2018).

While delivering a human-interpretable explanation for a single component is challenging, the problem is aggravated in the context of multi-component systems (Lipton, 2016; Burgess, 2018). In contrast to atomic approaches showing a single unified explanation, multi-component models have the challenge of providing explanations presenting how each individual component works and how they are integrated to deliver the system's results.

This chapter defines a hierarchical explanation model for the proposed semantic parser, exploring a hierarchical representation in an increasing degree of technical depth.

5.2 Levels of Interpretability

The hierarchical model defines three different levels of interpretability.

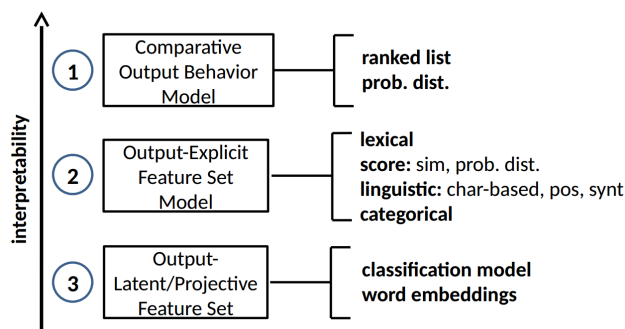


Figure 5.1: The three levels of interpretability defined in a hierarchical model.

1. **Comparative Output Behavior Model:** Displays the output in contrast to other alternative top-k outputs. It provides the user with the *aggregate* understanding of either the classification or similarity aspects computed by the model. It outputs a *ranked list* ordered by a *score* or by a *probability distribution*. *Background knowledge required by the user:* expected output behaviour of the classifier.
2. **Output-Explicit Feature Set Model:** Displays the set of relevant features associated with a specific output class or score. These sets of features can be associated with four types: (i) lexical (naturally interpretable input), (ii) numerical (corresponding to a probability distribution or score), (iii) linguistic (affixes, POS, syntactic elements) and (iv) categorical (derived categories). *Background knowledge required by the user:* meaning of the features.
3. **Output-Latent / Projective Feature Set:** Corresponds to a projection-based visualisation of the impact of one or more features in the model. For word embeddings projections, it can be defined as 2D similarity models, while for machine learning based models it can be defined by correlation or attention-based models. *Background knowledge required by the user:* for word embeddings, the similarity model, for classifiers, the notion of the relevance of a feature for a classification task.

Figure 5.1 summarises the layers of the hierarchical model. Different layers of the model are interpretable for different end-user profiles. Layer (1) requires the understanding of the final task which the system should

address (i.e. the expected behaviour of the application). Typically, it consists of the final aggregation layer for a classifier, matcher or ranker. Layer (2) describes the relationship between features and the output. The interpretability of this layer is dependent on the understanding of the role of linguistic features in the classification (e.g. lexical categories, syntactic relations). Layer (3) is the deepest level and requires the understanding of the dynamics of a classifier, ranker (the notion of different discriminative weights associated with different features) as well as the understanding of the expected behaviour of a word embedding (clusters of similar terms).

Although our explanation model targets a general audience, from a certain depth, unavoidably the user needs some sort of technical knowledge to understand the explanation. Even from this level, the explanation should be chosen in such a manner that the effort and technical knowledge to understand it is minimised in the initial technical layers. The complexity increases gradually, up to the point in which a deep knowledge is mandatory to understand the information.

5.3 The Explanation Model

We instantiate the explanation model in the proposed semantic parser, for which, given a natural language command, it returns a list of function calls that potentially represents the command intent as depicted in Figure 5.2.



Figure 5.2: A command in natural language and a list of potential function calls representing the user intent.

Table 5.1 depicts the components of the semantic parser along with their respective input features and mechanism of work.

The model presents seven explanations grouped by three layers focused on the components’ behaviour and their particular input features. As expected in a heterogeneous architecture, each component operates under a different method, requiring different types of inputs.

5.3.1 Shallow Parser Rules & Syntactic Tree Layers

The first layer describes the **Shallow Parser** and the **Type Inferencer**. The explanations show the rules activated (*i*) to identify the command ob-

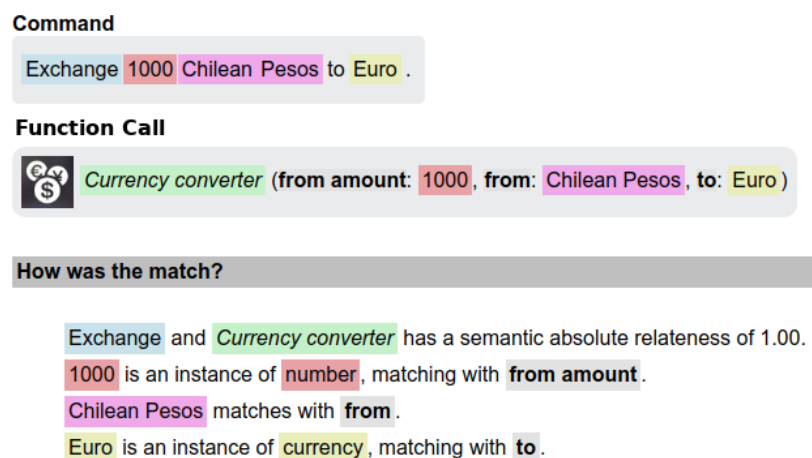


Figure 5.3: Explanations of the Shallow Parser and the Type Inferencer.

jects, *(ii)* to generate multi-word objects, and *(iii)* to identify the semantic types, highlighting the tokens and features involved in the process, as shown in Figure 5.3. The second layer depicts the features on which the rules operate, namely the syntactic tree and the part of speech (POS) of each token. Figure 5.4 shows a natural language command and both the set of POS-tags and the dependency tree associated with its tokens. These layers aim at showing the connection between the linguistic features and main concepts of the parsing system, whose interpretability is dependent on the understanding of the role of linguistic features in the classification.

Component	Features	Mechanism
Shallow Parser	command	rule-based
Type Inferencer	function descriptor, set of command objects	gazetteer
Intent Classifier	word embedding model, semantic relatedness, density	Random Forest
Ranker	relevance classification	ranking model

Table 5.1: Components present in the semantic parser along with their input features and mechanism of work. Each component uses also the features described in the predecessor components.

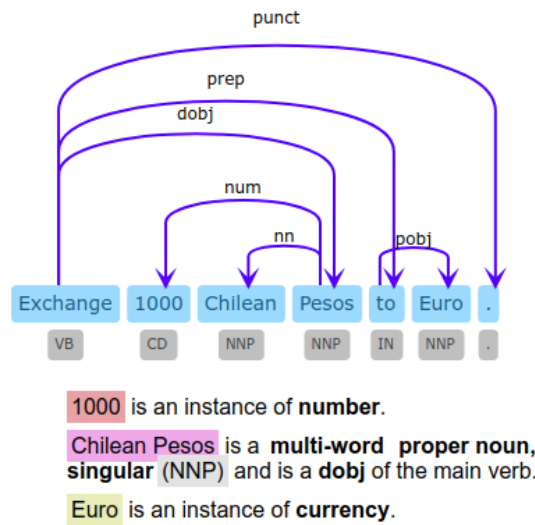


Figure 5.4: The second layer of the explanation model describing the linguistic features and highlighting the relationship between them and the command objects.

5.3.2 Word Embedding Layer

The matching process relies on the semantic relatedness scores, which represent the degree of semantic similarity the function descriptor and command objects have in relation to the function signature. The semantic relatedness is calculated from a word embedding model, which represents terms as vectors in a high-dimensional space. The explanation provides a cluster-based visualisation using t -SNE (van der Maaten and Hinton, 2008), where it plots the semantic elements that play a role in the matching process from both the command and the function signature as shown in Figure 5.5. The cosine between the points represents the degree of semantic relatedness in a *post-hoc* explanation fashion.

5.3.3 The Ranking & Classification Layer

The lower level is devoted to the most technical explanations which shows the mathematical expression that defines the final ranking score of the function call along with the features used in both the expression itself and in the `Intent Classifier`. To simplify the model to non-technical users, we reduced Equation 4.7 to Equation 5.1 in which all elements in the expression are represented by z , the vector of all features. Additionally, this level also presents the trained *Random Forest* classifier, showing the relevance of each feature in the final classification using the visualisation proposed by Welling et al. (2016), called *Random Floor*. Petkovic et al. (2018) also propose a

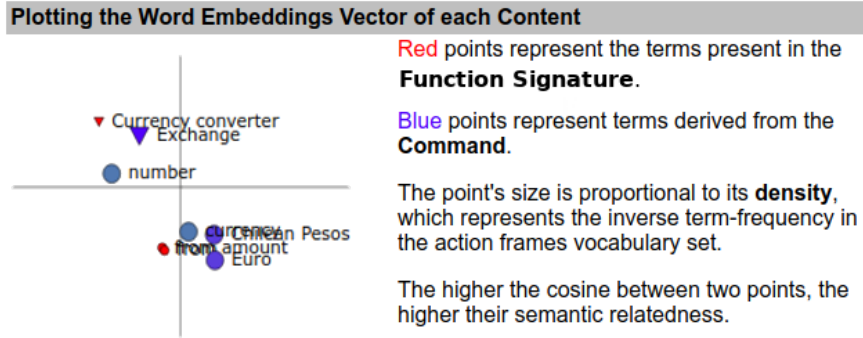


Figure 5.5: Plot of the elements from the command and function signature in which the cosine between the points represents the semantic relatedness.

visual representation for Random Forest models which makes use of a set of charts. Welling et al. (2016)'s proposal, however, is more condensed, which is the reason why we decided to apply it in our explanation model.

In the *Random Floor* chart, X-axis are variable values (in our experiment, the score of the semantic relatedness) and Y-axis the corresponding cross validated feature contributions, whereas the colour gradient in all plots establishes a parallel of the instances among them.

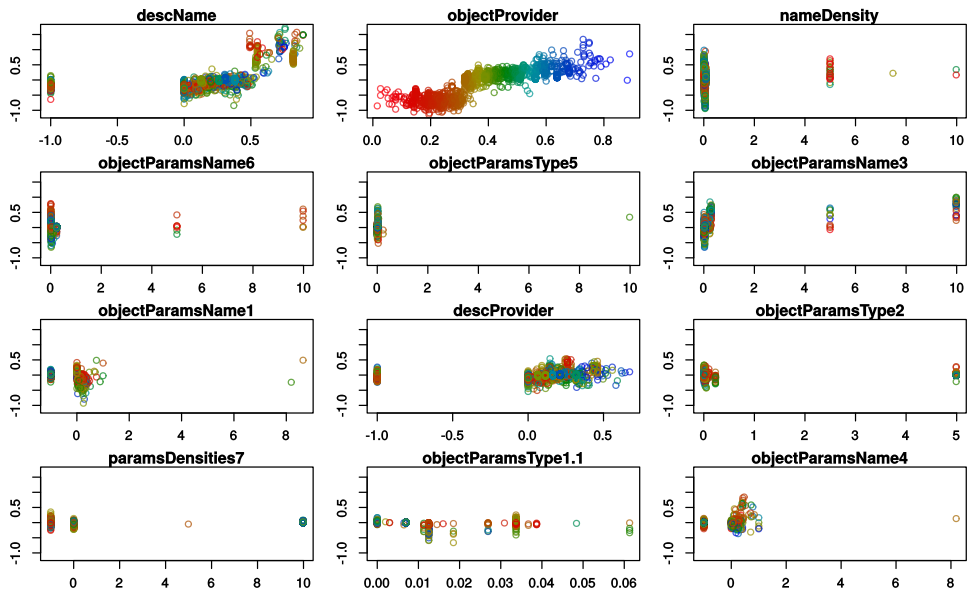


Figure 5.6: The relevance of some features in the Intent Classifier using the *Forest Floor* visualisation method.

Figure 5.6 shows the *Random Floor* representation model, depicting the feature correlations for the *Random Forest* classifier trained in the context

$$rs(c, f) = \sum_{i=0}^{|z|} (z_i) + 1000 * \tau \tag{5.1}$$

of this experiment. the `objectProvider` (the semantic relatedness of the function’s provider) is display at the centre on the top representing the most important feature for the prediction.

5.3.4 The Comparative Explanation

According to Jeroen and Erik (2002), explanations can be factual (*plain-fact*) or comparative, being the last subdivided into *property contrast*, *object contrast* and *time contrast*. All the explanations we presented so far are classified as *plain-fact*, since it acts in the context of one unique function call. In all layers, the explanations describe in detail aspects and properties of a unique function call. The user can infer from those explanations, for example, why this object matched with that parameter, why the semantic relatedness between the function descriptor and the function name has that specific value or why the function call received that specific ranking score. However, identifying the reason the function call A is ranked higher than the function call B demands a comparison between them. In this explanation, we allow the user to make such a comparison, whose intuition is to aggregate visual elements that can support the user to compare and contrast function calls with an eyesight. For this purpose we put side-by-side the instantiation of each function call, the projection of the word embedding vectors, the score expression and feature list.


Figure 5.7 shows the comparison between the function calls “*currency converter*” and “*make a payment*”. The comparative explanation reuses elements from the factual explanations, such as the ranking expression, the word embedding plot and the list of features.


5.4 Summary

Although usually claimed to have broad natural language capabilities, any natural language understanding system is limited to the rules it models or the set of latent patterns present in its training data sets. In this context, explanations can serve as a mechanism to communicate the real abilities and limitations of an intelligent system, allowing the users to adapt their writing style to favour the system’s performance.

We instantiated different explanation paradigms in the multi-component semantic parsing system targeting the end-user programming task. In Chapters 6 and 7, the system and the set of supporting explanation mechanisms

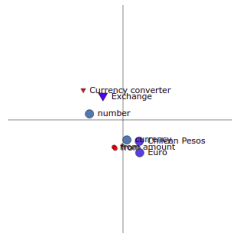
are respectively evaluated in the context of users with different ranges of background in linguistics and machine learning.

 **Currency converter** (from amount: 1000, from: Chilean Pesos, to: Euro)

 **make a payment** (invoice: 1000, method: Euro)

Score Expression and Features

$$\text{Action Score: } -12.61 = \text{relevance} * 1000 + \sum z$$



Red points represent the terms present in the **Function Signature**

Blue points represent terms derived from the **Command**.

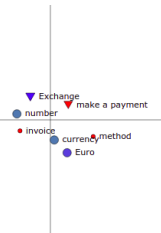
The point's size is proportional to its **density**, which represents the inverse term-frequency in the action frames vocabulary set.

The higher the cosine between two points, the higher their semantic relatedness.

Hide Features

Score Expression and Features

$$\text{Action Score: } -13.69 = \text{relevance} * 1000 + \sum z$$



Red points represent the terms present in the **Function Signature**

Blue points represent terms derived from the **Command**.

The point's size is proportional to its **density**, which represents the inverse term-frequency in the action frames vocabulary set.

The higher the cosine between two points, the higher their semantic relatedness.

Hide Features

Feature Values

Feature	Value
descNameRelatedness	-1.00
nameDensity	0.00
descProviderRelatedness	-1.00
maxObjectProviderRelatedness	0.36
objectParamName0	0.00
objectParamName1	0.03
objectParamName2	0.00
objectParamName3	-1.00
objectParamName4	-1.00
objectParamName5	-1.00
objectParamName6	-1.00
objectParamType0	-1.00
objectParamType1	-1.00
objectParamType2	-1.00
objectParamType3	-1.00
objectParamType4	-1.00
objectParamType5	-1.00
objectParamType6	-1.00
paramsDensities0	0.02
paramsDensities1	0.03
paramsDensities2	0.10
paramsDensities3	0.00
paramsDensities4	0.00
paramsDensities5	0.00
paramsDensities6	0.00

Feature Values

Feature	Value
descNameRelatedness	-1.00
nameDensity	0.05
descProviderRelatedness	-1.00
maxObjectProviderRelatedness	0.23
objectParamName0	0.00
objectParamName1	0.07
objectParamName2	-1.00
objectParamName3	-1.00
objectParamName4	-1.00
objectParamName5	-1.00
objectParamName6	-1.00
objectParamType0	-1.00
objectParamType1	-1.00
objectParamType2	-1.00
objectParamType3	-1.00
objectParamType4	-1.00
objectParamType5	-1.00
objectParamType6	-1.00
paramsDensities0	0.00
paramsDensities1	0.00
paramsDensities2	0.00
paramsDensities3	0.00
paramsDensities4	0.00
paramsDensities5	0.00
paramsDensities6	0.00

Figure 5.7: The comparative explanation shows the function calls *currency converter* and *make a payment* in the context of the natural language command *Exchange 1000 Chilean Pesos to Euro*.

Part IV

Experiments and Validation

The Evaluation of the Semantic Parser

6.1 Introduction

In this thesis, we separate the evaluation of the parser mechanism, which is presented in this chapter, from the user-centred evaluation of the explanation method, described in Chapter 7. The rationale behind this decision is to more clearly identify the performance of each contribution.

We start this chapter presenting the target data set with samples and statistics. Next, as both baseline models and the proposed architecture represents tokens as word embeddings, we present **Indra**, a spin-off tool acting as a centralised semantic relatedness function and vector repository. **Indra** was built to simplify the experimentation with word embeddings, especially regarding the composition of vectors and the identification of nearest neighbours.

The first experiment batches concentrate on the end-to-end encoder-decoder baseline models, which show their failure to address the given task with the presented data set, due to the small size of the training data.

Section 6.4 is devoted to the evaluation of the proposed architecture. We analyse variations in the implementations of both components **API Filter** and **Intent Classifier**, namely *TF-IDF* and *Nearest Neighbours* for the filtering function, and *Random Forest*, *Support Vector Machine* and *Multilayer Perceptron* for the final classification. Additionally, the experiment covers also scenarios where no filtering function or classifier is applied to help us identify the contribution of each component in the final solution.

The chapter finishes with a detailed analysis of the results and a short summary.

6.1.1 The Data Set

The target task is defined by the data set derived from the Task 11 of the SemEval 2017, named *End-User Development using Natural Language* (Sales et al., 2017, 2018a). The data set is composed of 185 natural language commands and 2005 function signatures. There are two types of commands. The first type is *if-then* recipe, which is composed of a *protasis*, i.e. the conditional clause, and the *apodosis*, i.e. the main clause. In this type of

command each clause is associated to a function signature. The second type is the solo direct command, which is associated to a unique function signature. Function signatures are also of two types. **Triggers** comprehend function signatures meant to map to the conditional part of the if-then command. **Actions** represent the set of function signatures meant to map to the main clause of the if-then commands and to direct commands. We borrow the nomenclature from the IFTTT platform¹, from which the function signatures came. The data set comprehending the mappings of natural language commands to function calls and the set of function signatures is available at <https://rebrand.ly/nlc-dataset>. Table 6.1 shows some examples of function signatures present in the knowledge base.

Function name	Provider	Parameters
Create a status message	<i>Facebook</i>	status message
Currency converter	<i>null</i>	from amount, from, to
Open garage door	<i>Garageio</i>	Which door
Create an issue	<i>GitHub</i>	repository, title, body
Create new contact	<i>Google Contacts</i>	full name, email...

Table 6.1: Examples of function signatures present in the KB.

Table 6.2 summarises the data set regarding its size. The numbers show the unbalance between command samples and the total number of available function signatures, which give a flavour of the complexity of the task.

Commands			Functions		
<i>conditional</i>	<i>direct</i>	<i>total</i>	<i>trigger</i>	<i>action</i>	<i>total</i>
34	151	185	1225	780	2005

Table 6.2: Number of commands and function signatures present in the data set, according to their respective types.

6.2 Indra

Vector space models usually comprise large data sets, whose manipulation can easily raise performance issues. As word embedding and semantic relatedness play a central role in our architecture, we implemented a self-contained tool whose twofold goal is to simplify the experimentation while guaranteeing high performance. Given the general academic interest in vector space models, we spinned out the effort as a library called *Indra* (Sales et al., 2018b). *Indra* can be used in the form of a local library or a web

¹<http://ifttt.com>

service receiving calls by REST standards. The project is now realised as an open-source² under the MIT license.

Among other functionalities, such as providing different types of semantic metrics and exploring multi-linguality with machine translation (Freitas et al., 2016), four functions are of higher importance for our proposed architecture: obtaining a vector representation of individual tokens; providing a composed representation for multi-token strings; calculating semantic relatedness of vectors; and providing an efficient method for the identification of nearest neighbours. The four functions are briefly presented in the following sections.

6.2.1 Word Embedding Vector

Indra allows easy access to the vector representation of single tokens. It can serve several word embedding models, in different languages, built from different text corpora and using different types of algorithms. For instance, by just changing parameters, *Indra* gives access to models built by *Skip Gram* (Mikolov et al., 2013), *Global Vectors* (Jeffrey Pennington, 2014), *Explicit Semantic Analysis* (Gabrilovich and Markovitch, 2007) or *Latent Semantic Analysis* (Dumais et al., 1988), over different text corpora, such as *Google News* or *Wikipedia*. The simplicity of experimenting with such variations helps, for example, to identify the best combination of algorithm and corpus for a given problem (Sales et al., 2017).

6.2.2 Composition Vector

In a vector space model, each vector represents an individual token. The definition of tokens in the corpus depends on a pre-defined choice in the tokenisation during the counting/training process of building the word embedding model. Frequently, the tokenisation strategy targets splitting the text in tokens that represents single words in the target language.

In addition to obtaining vectors for single tokens, our approach requires the composition of them to generate the representation of multi-token strings. For instance, when projecting the function name *currency converter* on the *Predicate Hyperspace*, the model needs to obtain the two vectors representing respectively *currency* and *converter*, and combine them to generate a final representation. This step requires the use of a compositional method.

Indra offers a built-in function that detects the occurrence of multi-token strings and automatically applies a compositional function according to the parameters. Currently, the tool supports four compositional methods (average, simple sum, distinct sum, l2-norm sum) (Sales et al., 2017).

²<https://github.com/Lambda-3/Indra>

The final vector representation can then be used to both calculate the semantic relatedness and identify the nearest neighbours.

6.2.3 Semantic Relatedness

Semantic relatedness is the core feature used in the `Intent Classifier`. Calculating it at scale demands optimisation to guarantee a reasonable execution time. `Indra` integrates this functionality with both single and composite vectors in a multi-threading fashion to improve the performance. The service allows the use of twelve relatedness metrics (AbsoluteCosine, AlphaSkew, Chebyshev, CityBlock, Cosine, Dice, Euclidean, Jaccard2, Jaccard, JensenShannon, Pearson, Spearman).

6.2.4 Nearest Neighbours

To find the n closest points to a given query vector in a multi-dimensional space that considers thousands of points is a computationally intensive task, which justifies why many researchers opt for approximate approaches (Rehurek, 2014). In the case of `Indra`, we incorporate the `annoy` method, developed by *Spotify*, which wins out the popular `Gensim` (Rehurek and Sojka, 2010) in performance (Rehurek, 2014). The function is also able to use the composition functions in the query vector.

The integration of the aforementioned features makes `Indra` a recommended tool to speed up both the development and execution phases of any experiment setting that depends on semantic relatedness, nearest-neighbours functions, or correlated measures.

6.3 The Encoder-Decoder Baseline Models

As described in Chapter 3, the encoder-decoder neural network model is the typical state-of-the-art approach for semantic parsing. This type of model, however, demands large data sets to demonstrate its efficiency. To prove its shortcomings to deal with the problem we consider in our research, we instantiated two encoder-decoder models, namely a sequence-to-sequence recurrent neural network architecture and an attention-based architecture, which are presented below.

6.3.1 The Sequence-to-Sequence Baseline

We implemented a sequence-to-sequence neural network composed of LSTM cells as the recurrent unit in the style of Sutskever et al. (2014), which is represented in Figure 6.1. The training process was designed to receive a matrix $Z \in \mathbb{R}^3$ containing the set of function calls. Each function call is represented by the function descriptor and the set of command objects

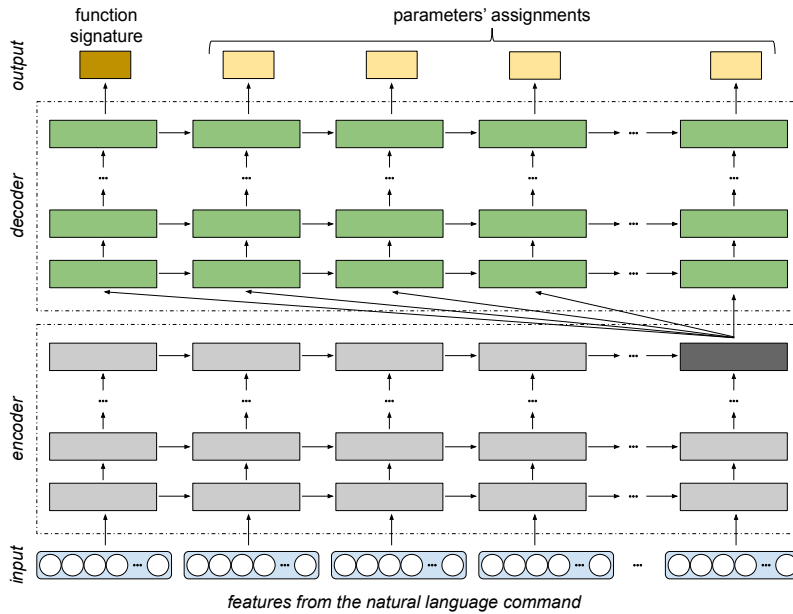


Figure 6.1: A typical method to solve this task is the encoder-decoder machine learning model in the sequence-to-sequence fashion (Ling et al., 2016; Gulcehre et al., 2016; Sutskever et al., 2014). The grey and green boxes represent LSTM cells and the *parameters' assignments* are one-hot vectors associating chunks of the input text to the parameters.

expressed as 300-length vectors generated by the skip-gram model generated over the Google News corpus.

The training process was conducted in two steps. First, we encoded each function signature present in the `API Knowledge Base` as input, in order to make the model aware of all of them. We used the function name as function description and the parameter names as command objects. Secondly, we used the training data in a 10-fold cross-validation fashion.

We evaluated different network architectures varying the number of layers (1 to 3), nodes (1 to 3 times the input size), dropouts (0, 0.3, 0.5), epochs (up to 500), learning rates (0.001, 0.003, 0.01, 0.03, 0.1) and batch sizes (100%, 50% and 25% of the training data). All the evaluated models delivered an f1-score of 0.

6.3.2 The Attention-Based Baseline

An important improvement in the encoder-decoder model is the advent of the `Transformer` neural network architecture, which defines a new model that uses neither recurrent nor convolutional layers, but a new architecture composed solely of attention functions (Vaswani et al., 2017).

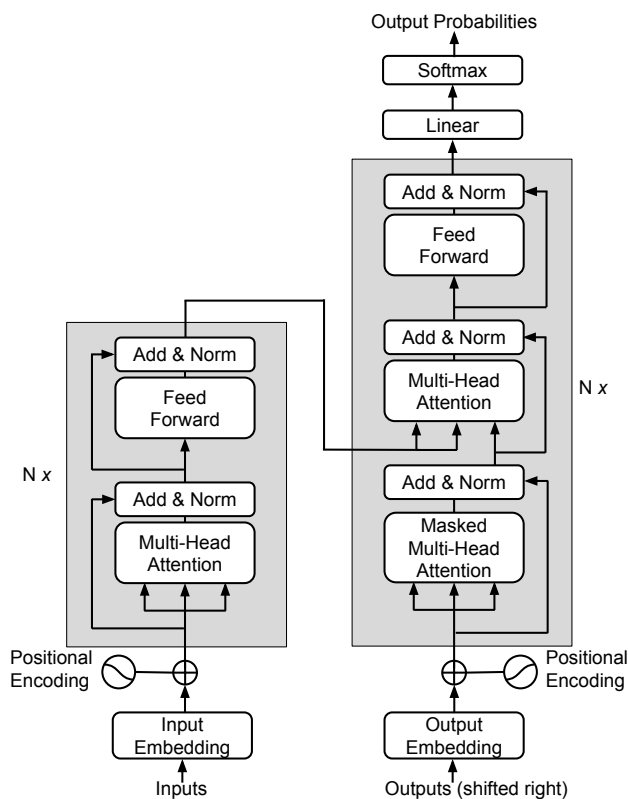


Figure 6.2: The Transformer model architecture defined by Vaswani et al. (2017), where the left block represents the encoder and the right block the decoder.

For our experiments, we implemented the original architecture model defined by Vaswani et al. (2017). Its encoder is composed of N identical layers, each made up of a pair of a multi-head self-attention mechanism and a fully connected feed-forward neural network. The decoder has also N identical layers, varying from the encoder by the addition of an extra multi-head self-attention mechanism, as depicted in Figure 6.2.

The training process follows the same two-step methods applied in the sequence-to-sequence architecture.

We also evaluated different network architectures varying the number of layers (1, 3 and 6). The output of the sub-layers of the models, the hyper parameters of the multi-head attention mechanism and the feed-forward networks follow the default values defined in the original paper ($d_{model} = 512, h = 8, d_k = d_v = d_{model}/h = 64, d_{ff} = 2048$). Once again, all the evaluated models delivered an f1-score of 0.

6.3.3 Baseline Analysis

The task requires the identification of the target function call over a set of thousands of function signatures, besides the correct map of the parameters values. As a consequence, this needs multiple samples per instance to deliver a reasonable performance.

In the case of our data set, however, there is a significant number of function signatures without a single instance, and among those used in the commands, the average number of instances per function signature lies at 3.5, which is very low.

End-to-end encoder-decoder models are well-known for delivering good performance, given they have access to a reasonable large training data set (Halevy et al., 2009). An intuitive explanation for the failure of those experiments in the given data set is the low rate of number of training samples with respect to the number of classes.

6.4 Evaluating the Proposed Architecture

Our experimental setting aims at evaluating the proposed architecture considering different types of implementation for our two main components: the `API Filter` and the `Intent Classifier`.

6.4.1 API Filter Settings

The `API Filter` assumed three implementations. In the first case, ρ implements the identity function. This configuration means that no filter is applied and aims at measuring the relevance of the filtering step. Secondly, a natural candidate for a filtering function is the TF/IDF weighting scheme, which selects the target function signatures considering the overlap of their vocabulary with the query, i.e the natural language command. Our experiment show the TF/IDF filtering function on average limits the number of target function signatures to 10. The third approach uses a nearest neighbours method to select the 50 closest function signatures, when projecting the natural language command on the `Predicate Hyperspace` defined by the word embedding model. This type of function is not limited to the overlapping of the vocabulary with the query, but expands their relations to the latent notion of semantics defined by the word embedding model.

6.4.2 Intent Classifier Settings

With regard to the `Intent Classifier`, we evaluated three learning methods: *Random Forest*, *Support Vector Machine* and a simple *Multilayer Perceptron Neural Network* (MLP). We calibrated each learning configuration identifying their optimal hyper-parameters by grid searching. For Random

Forest, the *number of estimators* ranged into (100, 300, 1000, 3000, 10000), the *maximum number of features* assumed the *sqrt* or the *log2* of the total available. For the Support-Vector-Machine classifier the grid search was applied considering the *kernel* varying into *linear*, *sigmoid* and *polynomial* (with 2, 3, 5 degrees) and *gamma* into the log-space $(-9, 3, 3)$, keeping a fixed $C = 1$. Finally, for the MLP network, we evaluated under the same variation specified for the sequence-to-sequence model. The classifiers receive the input features as described in Section 4.4.6.

Our experiments used the skip-gram model generated over the Google News data set as the word embedding model (Mikolov et al., 2013). As mentioned, to speed up the development and execution time, we used the **Indra** word embedding server (Sales et al., 2018b). To fix the problem of imbalanced classes in the training data, we applied random majority under-sampling with replacement, making use of the **imbalanced-learn** library (Lemaître et al., 2017).

We evaluated the architecture purposely with off-the-shelf implementations for both the **API Filter** and the **Intent Classifier**. This decision seeks to highlight the relevance of the model architecture and the feature selection to the final solution.

6.5 Results & Discussion

Classifiers	Scenario	Identity	TF/IDF	Nearest Neighbours
RF	<i>TOP-10</i>	0.4217	0.6594	0.6825
	<i>TOP-50</i>	0.7549	0.7778	0.8551
SVM	<i>TOP-10</i>	0.0476	0.4298	0.3608
	<i>TOP-50</i>	0.2280	0.6224	0.6232
MLP	<i>TOP-10</i>	0.0738	0.4798	0.3944
	<i>TOP-50</i>	0.3110	0.7015	0.7292
None	<i>TOP-10</i>	0.0380	0.4071	0.3119
	<i>TOP-50</i>	0.1737	0.5989	0.6680

Table 6.3: Recall for Random Forest (RF), Support Vector Machine (SVM) and Multilayer Perceptron Neural Network (MLP) for different filter functions evaluated in the TOP-10 and TOP-50 scenarios. The last line (None) describes the results when no classifier is used.

Tables 6.3 and 6.4 show the results of the proposed approach in different combinations of filter functions and classifiers, measured respectively in relation to the *recall* and *mean reciprocal rank* (MRR). The evaluation was carried out in two scenarios: the first considers the function calls ranked

Classifiers	Scenario	Identity	TF/IDF	Nearest Neighbours
RF	<i>TOP-10</i>	0.1264	0.2878	0.3038
	<i>TOP-50</i>	0.1426	0.2932	0.3120
SVM	<i>TOP-10</i>	0.0099	0.1901	0.1207
	<i>TOP-50</i>	0.0123	0.1978	0.1187
MLP	<i>TOP-10</i>	0.0187	0.2063	0.1323
	<i>TOP-50</i>	0.0311	0.2171	0.1514
None	<i>TOP-10</i>	0.0115	0.1626	0.0954
	<i>TOP-50</i>	0.0175	0.1700	0.1147

Table 6.4: Mean Reciprocal Rank for Random Forest (RF), Support Vector Machine (SVM) and Multilayer Perceptron Neural Network (MLP) for different filter functions evaluated in the TOP-10 and TOP-50 scenarios. The last line (None) describes the results when no classifier is used.

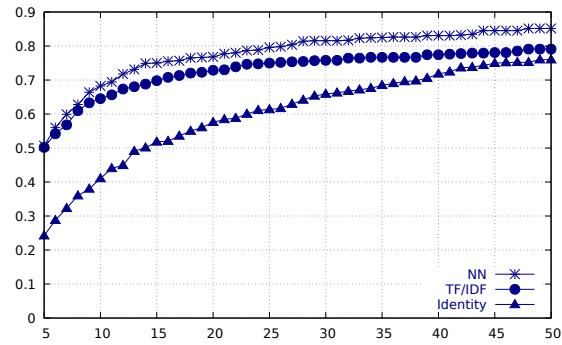
up to the 10th position, whereas the second, up to the 50th. In the experiments, we assumed that only one function call corresponded to the target answer. This assumption makes precision a redundant indicator since it can be derived from the recall.

6.5.1 API Filter

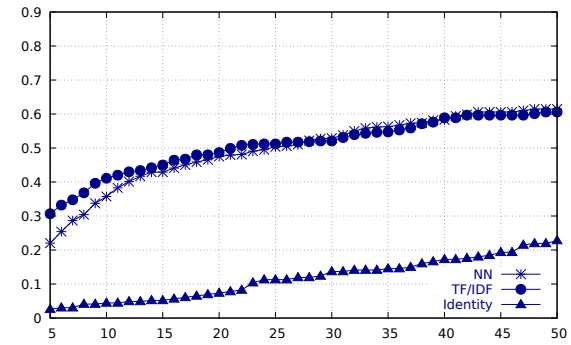
The application of the *identity function* as an **API Filter** represents the absence of a pivoting area in the sense that the full **API Knowledge Base** is considered in the classification step. Given the simplicity of the feature set and the straightforwardness of the matching model, the results show that the filter function plays a key role in increasing the performance. The experiments in which the identity function is present consistently deliver lower recall, as shown in Figure 6.3. Even when combined with a Random-Forest-based **Intent Classifier**, where the identity function had a more competitive performance in recall, Figure 6.4a shows that the target function calls are significantly lower ranked. As expected, an overall look at Figure 6.4 shows that the identity filter function consistently places the target function

	TF/IDF	Nearest Neighbours
# of function signatures	31.73	50.98
missed by the pivoting area	15.55%	6.84%
maximum recall	84.45%	93.16%

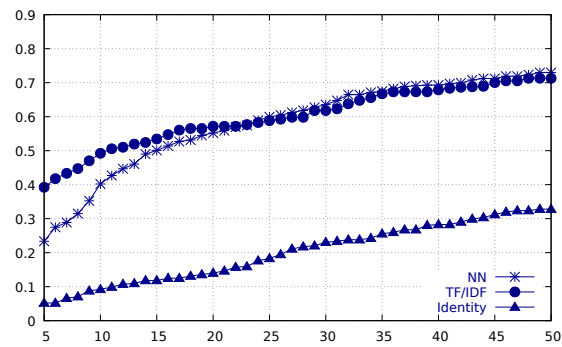
Table 6.5: The average number (*#*) of *function signatures* into the pivoting area and how many functions are missed in percentage.



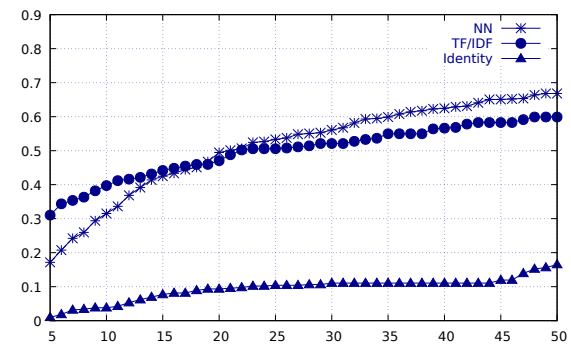
(a) Random Forest



(b) SVM

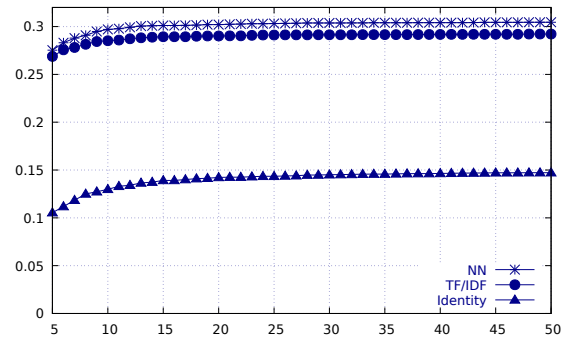


(c) MLP

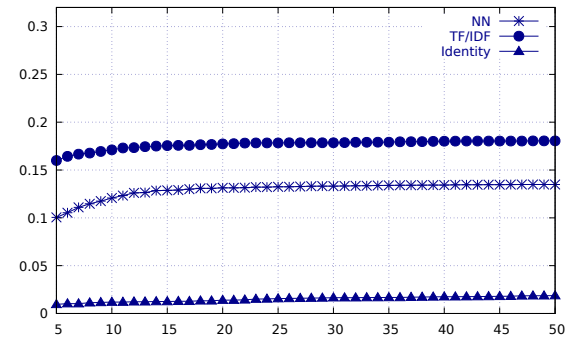


(d) None

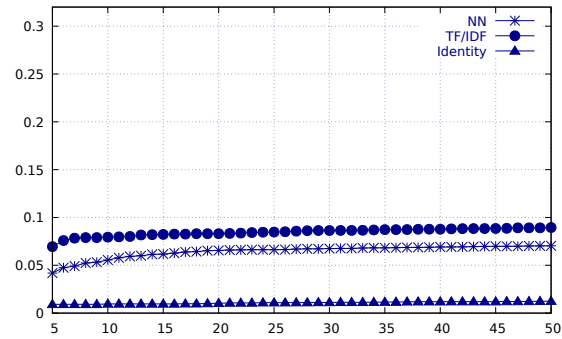
Figure 6.3: The charts show the behaviour of the recall for the Random Forest(a), SVM(b) and MLP(c) considering the three filter functions. The fourth chart (d) describes the behaviour of the ranking system when the classification component of the ranking equation is ignored.



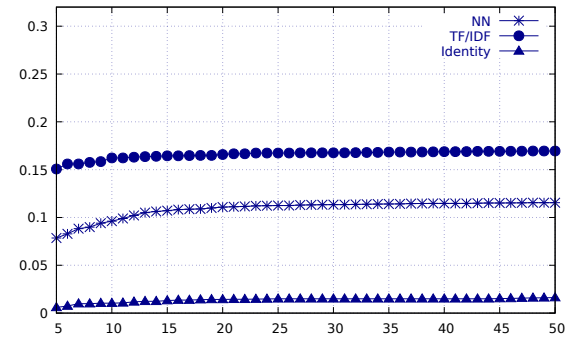
(a) Random Forest



(b) SVM



(c) MLP



(d) None

Figure 6.4: The charts show the behaviour of the mean reciprocal rank for the Random Forest(a), SVM(b) and MLP(c) considering the three filter functions. The fourth chart (d) describes the behaviour of the ranking system when the classification component of the ranking equation is ignored.

calls in lower ranks regardless of the `Intent Classifier`.

The other two filter functions have similar results in recall, with a slight advantage of Nearest Neighbours over TF/IDF. This is explained by the fact that the TF/IDF cut out relevant function signatures more often than Nearest Neighbours as shown in Table 6.5. Whereas Nearest Neighbours allows the `Intent Classifier` to evaluate on around 50 function calls, the TF/IDF filter reduces this search space to around 31. This stronger cut reduces the potential maximum recall of TF/IDF to around 84%, compared to around 93% of the Nearest Neighbours approach.

6.5.2 Classifiers

Random Forest is by far the best classifier considering either recall or MRR in all of the evaluation scenarios, whereas the SVM and MLP classifiers perform similarly in relation to the recall. Our assumption is that the set of semantic relatedness features shows a high level of independence, producing many sub-optimal areas that can be better avoided by a tree-based learning model. This assumption is reinforced considering the scenarios in conjunction with the identity function, in which the higher volume of data tends to generate more sub-optimal spaces and represents exactly the scenario where the other classifiers show lower performance. The MRR values in the Random Forest chart (a) means that on average the target function calls are placed between the 3rd and 4th positions.

We also evaluated a scenario in which the relevance classification is ignored, i.e. the rank function becomes exclusively the sum of the features without any learning process as depicted in Equation 6.1.

$$\sum_{i=0}^n (z_i) \quad (6.1)$$

Figures 6.3d and 6.4d show respectively its recall and MRR. Considering the recall, ignoring the classification component of the ranking equation delivers results equivalent to the SVM, given its ineffective classification as discussed in our further analysis.

We also analysed the effectiveness of the classifier in isolation and its impact on the results. Figures 6.5, 6.6 and 6.7 show the confusion matrices of the three classifiers.

The Random Forest classifier shows a high accuracy in the classification of the 0-class function calls. This ability helps in removing non-related calls from the top positions. Classes 1 and 2, which represent the correct function calls but without all the correct parameter assignments, are frequently classified as 0.

SVM is the worst-performing model in the correct identification of the 0-class, where more than 50% of them are misclassified as 3. Additionally, it

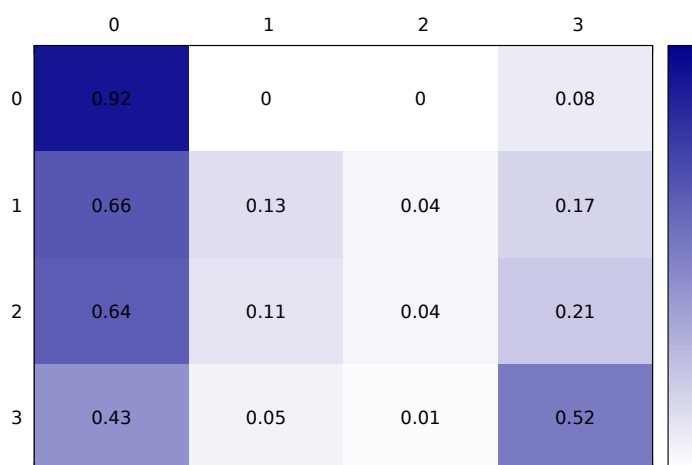


Figure 6.5: The confusion matrix in percent of the Random Forest classifiers considering the evaluation scenario with the nearest neighbours filter function.

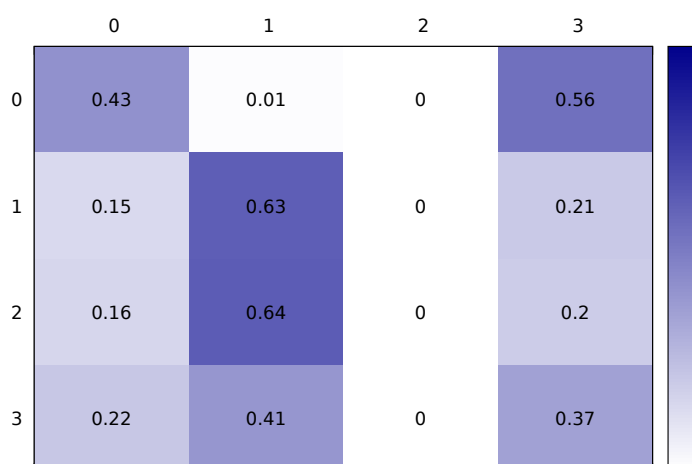


Figure 6.6: The confusion matrix in percent of the SVM classifiers considering the evaluation scenario with the nearest neighbours filter function.

poorly identifies the 3-class, which led to the low accuracy. Given the gains and the losses, it is on average as if no filter was applied as shown in the comparison of Figures 6.3b and 6.3d.

Regarding the MLP classifier, while having high accuracy in classifying correctly the 0-class, the other classes are generally all classified as 3-class. The excessive number of mis-classifications of 1 and 2 classes explains its poor result, performing worse than the scenario without a classifier.

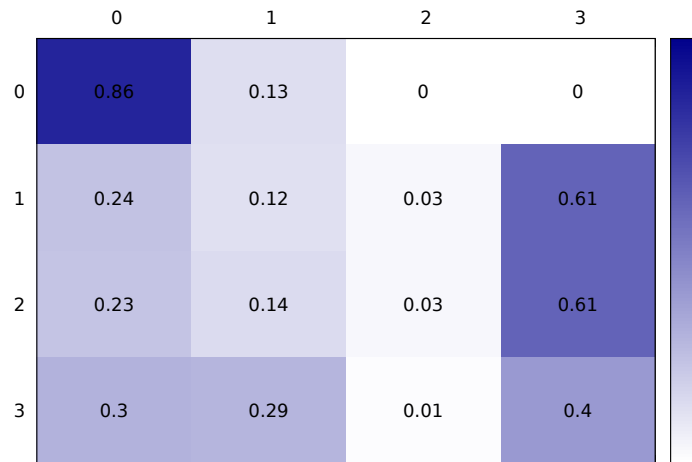


Figure 6.7: The confusion matrix in percent of the MLP classifiers considering the evaluation scenario with the nearest neighbours filter function.

Considering that our evaluation is focused only on the correct matching function with the correct set of parameters, classifying 1 and 2 with any class different than 3 does not create high impact on the ranking. In reality, classifying only as a boolean (`true` for the actual 3-class and `false` for the others classes) would be ideal. However, after some experiments showing better results, we opted for making the current fined-grained classification, since the full matching spectrum is better identified.

6.6 Summary

The proposed semantic parsing method maps natural language commands to function calls considering a large and heterogeneous `API Knowledge Base` under a restricted set of annotated data. The proposed semantic parsing method operating with a `Nearest Neighbours API Filter` and a `Random Forest Intent Classifier` was able to solve up to 68% of the commands considering the TOP-10 evaluation scenario, and up to 85% when considering the TOP-50, with MRR scoring around 0.3, which means that the target function calls are placed on average at the 3rd or 4th positions.

Our experiments also show that many instances of both sequence-to-sequence and attention-based machine learning models were unable to present successful results in the task. As briefly debated, an intuitive explanation is the lack of a large-enough training set.

The Evaluation of the Explanation Model

Contents

7.1	Introduction	105
7.2	The Experimental Setting	106
7.2.1	Participants	106
7.2.2	General Instructions	107
7.3	Evaluation Criteria	107
7.3.1	Mental Models	108
7.3.2	Type of Explanations	109
7.3.3	Accuracy <i>vs.</i> Explainability	109
7.4	Results & Discussion	110
7.4.1	Mental Models	110
7.4.2	Relevance of the Explanations	111
7.4.3	The Accuracy <i>vs.</i> Explainability Dilemma	112
7.4.4	Concerns About the Design of the Experiment	113
7.4.5	Considerations about Completeness	114
7.4.6	Participants' Comments and Impressions	114
7.5	Summary	115

7.1 Introduction

We present in this chapter the evaluation of the explanation model presented in Chapter 5 instantiated in the proposed multi-component semantic parsing system. The evaluation setting is designed to analyse the different types of explanations with different levels of abstraction and technical depth.

Three research questions guided our experimental analysis:

- To what extent are users able, irrespective of their technical background, to improve their mental models by associating the linguistic features from the explanations to the system's behaviour?
- How does the knowledge in machine learning affect the preference for technical explanations? and

- To what degree are users willing to favour explainability over accuracy given a certain context?

7.2 The Experimental Setting

The evaluation was carried out in a simulated use of the semantic parser, in which the system presents the user's natural language commands, and a suggested list of function calls as depicted in Figure 7.1. During the experiment, we asked the participants to go over a sub-set of twelve pre-configured natural language commands and their corresponding lists of 3 to 5 potential function calls as a result of the execution of the parser. This simplified scenario is intended to help the users focus exclusively on the command matching and the explanations (when available), thus avoiding the distraction of having to deal with a larger list of options to decide upon.



Figure 7.1: A command in natural language and a list of potential function calls representing the user intent.

7.2.1 Participants

We recruited 66 adult participants from our professional networks whose unique requirement was to be fluent in English. The set of participants is composed of 26 females and 40 males, with an age range between 20 and 49. They reported their level of knowledge in *machine learning* (ML) and *English grammar* (EG) according to the same scale suggested by Amos Azaria (2016), to which we attributed a score from 1 to 6 respectively:

1. *none*;
2. *very little*;
3. *some background from high school*;
4. *some background from university*;
5. *significant knowledge, but mostly from other sources*;

6. *bachelor with a major or minor in the related topic.*

The participants were divided randomly into the *control group*, composed of 31 participants, which were given access to the system without the explanation, and the *treatment group*, composed of 29 participants, with access to the explanation. The random division longed for balancing the number of participants with and without ML knowledge in each group. Table 7.1 shows the number of participants in each group.

Knowledge Level	1	2	3	4	5	6
Machine Learning	14	6	1	17	9	19
English Grammar	0	0	11	18	27	10

Table 7.1: Distribution of the participants according to their knowledge in machine learning and English Grammar.

7.2.2 General Instructions

We introduced the experiment to the participants by exposing its main goals and the expected procedures in the task. We highlighted that the idea behind the parser is to allow a user to find suitable functions and their parameters from their commands expressed in natural language, regardless of their technical knowledge. We asked them to select the correct function call for each pre-configured command, while examining the tool to infer how it works. For the users that participated in the treatment group, we encouraged them to examine the explanations, which shows how the system maps commands to the function calls, and also encouraged them to see the comparative explanation using the proper buttons.

We estimated a dedicated participant would take less than 15 minutes to complete the task without the explanation, whereas the time increases to around 30 minutes for the case with the explanation. Appendix A details the complete questionnaire filled by the participants.

7.3 Evaluation Criteria

In our evaluation, we examined three main aspects of the impact of the explanation on the user:

- Firstly, how the explanation affects the mental models of the end users;
- Secondly, which types of explanations can better serve the needs of the end users, considering their background knowledge; and finally,
- How end users perceive the dilemma of accuracy versus explainability.

We elaborate each of them in the following subsections.

7.3.1 Mental Models

A mental model is a cognitive representation of the external world to support the human reasoning process (Jones et al., 2011). The closer a person’s cognitive representation to the external world, the higher the understanding and the ability to take decisions about it (Johnson-Laird, 1983). In our task, the “external world” is represented by the semantic parsing system, and we evaluate the user’s mental model by assessing whether the presented explanations help the user to understand the system’s mechanisms. We designed a set of questions to measure whether the user realised the correct influence of linguistic features in the overall performance of the parser in both the **Shallow Parser** and the other components of the ranking model, such as the **Intent Classifier**. Given a contextual command, the participants were asked to judge affirmative sentences on the following Likert 7-point scale (Likert, 1932):

1. *strongly disagree*;
2. *disagree*;
3. *somewhat disagree*;
4. *neither agree nor disagree*;
5. *somewhat agree*;
6. *agree*;
7. *strongly agree*.

We evaluated three aspects of the **Shallow Parser**: (i) the role of proper nouns, (ii) the importance of the correct spelling and use of grammar and (iii) the verb mood (indicative *vs.* imperative).

Capital Letters Matter

Proper nouns in general start with a capitalised letter in English. As proper nouns define a command object, we want to identify to what extent users figure out the impact of this feature in the system’s performance. After giving a contextual command, we asked the participants to judge the veracity of sentences like:

*“Writing ‘Swiss Francs’ with capital
letter increases the system comprehension”*

Full Sentence Instead of Keywords

Incomplete sentences can introduce errors in the part-of-speech tagger and grammar tree parser, which, on the other hand, leads to wrong interpretation of the objects. In this task, we present grammatically incomplete commands similarly to many information retrieval systems, as in a keyword-search style, to highlight the importance of grammatically correct sentences. After giving a contextual command, we asked the participants to judge the veracity of sentences like:

*“Writing a set of keywords for the command
has the same result as grammatically correct sentences”*

Give Orders instead of Polite Requests

About verb mood, we presented to the participants commands written as questions and in the indicative form. After giving a contextual command, we asked the participants to judge the veracity of sentences like:

“Starting by ‘I would like’ increases the system comprehension”

7.3.2 Type of Explanations

We also asked the participants to evaluate the relevance of each explanation presented in the model, also, on the Likert 7-point scale. The questions had the following template:

- *I found helpful to see how parameters’ values are identified.*
- *I found helpful to see the comparison explanation.*
- *I found helpful to see the scoring expression.*

Given the different nature of the explanations, such as, targeting simulation capability or visualisation, we decided to opt for an open question able to allow the user to grade the helpfulness they believe the explanations had in helping them understand the system.

7.3.3 Accuracy vs. Explainability

Explainable AI models have advanced significantly, however in many cases higher transparency still comes with a cost of accuracy (Lipton, 2016; Adadi and Berrada, 2018). We questioned the participants to what degree they would favour accuracy over interpretability for a semantic parsing, stating that the higher the accuracy, the better the system’s results, while explainability means the ability to understand the underlying systems’ mechanisms of work. To expand the context of the evaluation, we also inquired the participants the same question targeting two further hypothetical contexts:

- in the case of a cancer exam of a relative;
- in the case of a bank loan decision.

The participants answered these questions on a 7-point scale, in which 1 means high explainability and low accuracy, and 7 the opposite, i.e. low explainability and high accuracy.

7.4 Results & Discussion

We associated the answers on the Likert 7-point scale to the interval -3 to 3, where 0 is the neutral answer and 3 represents *strongly agree* when the question reflects a true statement, and *strongly disagree* when it represents a false statement. We also analysed the statistical significance of the results using the *t-test*, which is represented by the variable p .

7.4.1 Mental Models

Table 7.2 presents the results of the mental model assessment. On average, participants in the treatment group give scores 55% higher than those in the control group (1.13 vs. 0.73, $p < 0.05$). The results also demonstrate that knowledge in machine learning and English grammar exhibit a significant positive correlation with the mental model scores in both treatment group ($r = 0.54$ for ML, $r = 0.45$ for EG) and the control group ($r = 0.45$ for ML, $r = 0.46$ for EG). The invariance of the correlation coefficients among the groups and the mental model scores strongly suggest the explanation model helps users to build better mental models.

<i>Metrics</i>	<i>Treatment Group</i>		<i>Control Group</i>	
Average	1.13		0.73	
r (ML)	0.54		0.45	
r (EG)	0.45		0.46	
	<i>Acquainted</i>	<i>Non-</i>	<i>Acquainted</i>	<i>Non-</i>
Avg. (ML)	1.62	0.63	1.07	0.54
Avg. (EG)	1.42	0.62	1.11	0.34

Table 7.2: The results regarding the mental model assessment, presenting the average scores and Pearson correlation coefficient r in relation to machine learning knowledge (r ML) and English grammar knowledge (r EG) for both treatment and control groups. The last two rows represent the average scores grouped by the level of specialisations.

To explicitly present this conclusion, we divided both treatment and control groups into four subgroups according to their knowledge in ML. We considered as acquainted with ML those users that declared having *significant knowledge, but mostly from other sources* or a *bachelor with a major or minor in the topic*. On average, the score of the users acquainted with ML in the treatment group was 1.62, while it was 1.07 in the control group ($p < 0.05$). Although not being the focus of our study, the results concerning EG knowledge present a similar tendency as shown in Table 7.2.

7.4.2 Relevance of the Explanations

The participants show a clear preference for less technical explanations, regardless of their ML background, as shown in Table 7.3. For instance, the relevance score given to the *Shallow Parser Layer*, which shows high score in both groups, is more than five times higher than the score given to the *Intent Classifier*.

Average	Overall	Acquainted	Non-Acquainted	p
Shallow Parser Layer	1.69	2.19	1.19	< 0.001
Syntactic Tree Layer	0.84	1.38	0.31	< 0.1
Word Embedding	0.59	1.19	0.00	< 0.05
Feature List	0.31	-0.19	0.81	< 0.1
Score Expression	0.91	0.38	1.44	< 0.05
Intent Classifier	0.31	0.56	0.06	> 0.35
Comparative	0.78	0.88	0.69	> 0.70

Table 7.3: The average scores of the relevance given to each type of explanation on a Likert 7-point scale (-3 to 3). The second and third columns show respectively the averaged scores for the group of participants acquainted and non-acquainted in machine learning, whereas the last column represents the p-value of the statistical significance (t-test).

Syntactic Tree Layer and *Word Embedding*, however, present significant divergences between the groups (respectively $p < 0.1$ and $p < 0.05$). Although these explanations cannot be considered highly technical, they still demand a sort of domain-specific knowledge to be interpreted. As the users acquainted with ML master the required knowledge, we presume their understanding as straightforward. In contrast, the other group of users that struggle to comprehend them, results in the lower scores.

Feature List and *Score Expression* also present high divergence (respectively $p < 0.1$ and $p < 0.05$). When interviewing the participants, some users not acquainted with ML (participants P21, P26 and P30) acknowledged that

this pair of explanations could serve to check the system’s correctness. Other users acquainted with ML (P5 and P14), however, credited their low interest to the fact that debugging at this stage would allow checking only the last expression, which is an unlikely source of problem.

Participants were unanimous in relation to the *Intent Classifier* explanation, giving comparatively low scores to its relevance. Additionally, the results do not show a statistical significance between the groups ($p > 0.35$), suggesting that the background in ML does not affect the general perception about the explanation. In general, as the explanations become more technical, either the users are unable to understand them or they do not invest enough time to interpret them. Although this hypothesis might not cover the case of the technically competent and knowledgeable users, we presume, however, the lack of interest in this part of the experiment is associated with the lack of incentive to convince them to make the necessary effort to fully understand the explanations.

The score of the *Comparative* explanation in the group acquainted with ML exceeds the average score of its compound parts, suggesting that the users see a value from looking at two distinct function calls simultaneously. Although the same cannot be identified in the group non-acquainted with ML, the fact that there is no statistical significance between the groups ($p > 0.70$) indicates the lack of influence of ML knowledge on the user perception of the importance of the *Comparative* explanation.

7.4.3 The Accuracy vs. Explainability Dilemma

In this part of our analysis, we considered the participants from both the treatment and control groups. Generally, users tend to favour accuracy over explainability, as the score of the four questions are higher than 4 out of 7, as shown in Table 7.4. However, although having a small variability in the average scores, the results allow us to come up with a hypothesis.

The general average score for *command selection tool* and *cancer exam* are equivalent to each other ($p > 0.70$). Diversely, the general score for *bank loan* present statistical significance comparing to the other hypothetical contexts ($p < 0.05$). It means that for *command selection tool* and *cancer exam*, the participants in general would favour accuracy over explainability to a higher degree than when compared to *bank loan*. We conjecture the possible social unfairness of a bank loan decision incentivises the participants to increase their interest in an explanation, whereas, for the other two hypothetical contexts there are, in theory, no incentives for unfairness.

The results also tell us that the participant’s judgements are left almost perfectly uninfluenced by their ML or EG backgrounds, given the close-to-zero correlations shown in lines r (ML) and r (EG). This hypothesis is also confirmed by the lack of statistical significance between the group of users acquainted and not acquainted.

	Command Selection Tool		Cancer Exam		Bank Loan	
General Average	5.35		5.44		4.80	
r (ML)	0.07		-0.09		0.03	
r (EG)	0.01		-0.08		-0.12	
	Ave.	p	Ave.	p	Ave.	p
Acquainted	5.36	> 0.90	5.29	> 0.50	4.54	> 0.30
Non-Acquainted	5.34		5.55		5.00	
Treatment Group	5.09	< 0.1	5.44	> 0.95	4.66	> 0.50
Control Group	5.59		5.44		4.96	

Table 7.4: The results represent the degree to which participants favour accuracy over explainability. The average scores are represented on a 7-point scale, where 1 denotes high explainability and low accuracy, and 7 the opposite.

When we group the participants according to their role in the study, we see a slight difference between the groups for the *command selection tool* with a statistical significance of $p < 0.1$. However, the scores for the *cancer exam* and *bank loan* do not show any significant variance, which might suggest that users get biased only when carrying out a domain-specific experiment.

7.4.4 Concerns About the Design of the Experiment

Although the participants did not effectively write the commands by themselves, the evaluation presented diversified commands from which we can evaluate the users' ability to identify the parser mechanisms of execution to validate our hypothesis that participants exposed to the explanations were in a better position to identify how to write more understandable commands.

We designed this evaluation method after trying others without success. We acknowledge that allowing the user to input their own commands would set a better evaluation scenario. However, when we applied this approach, several problems appeared. When we asked the users to write their own commands based on a scenario description, they were biased by the description itself, reusing its vocabulary and syntactical structures. Additionally, the level of engagement was much lower. So we converged to the current approach.

7.4.5 Considerations about Completeness

At first consideration, the information provided in the explanations might seem complete and sound, in the sense that it explains in concrete terms the underlying mechanisms behind the components.

However, when drilling down through even deeper layers, the user might want to understand how (and why) the system produced some of the generated features such as POS-tags, syntactic structures or word embedding vectors.

As many systems recurrently depend on low interpretable linguistic-level features, it is expected that linguistic features and word-embedding might represent a cut-off point for most application scenarios. However, the same hierarchical model can be recursively applied to feature levels without loss of generality.

7.4.6 Participants' Comments and Impressions

Vocabulary Gap

Although studies show that the *vocabulary gap* is highly present when users interact with information systems (Furnas et al., 1987), some participants (P31, P33 and P55) felt uncomfortable with the variation of terms expressed in data set. In the context of the command *Exchange 1000 Chilean Pesos to Euro*, three participants argued that the verb *exchange* represents more the effective swap of cash, than the calculation of the value rate. This type of variation occurs because it is induced by the data set of the Task 11 of the SemEval 2017.

Neutral Score to Technical Explanation

During the experiment two participants (P19 and P32) said that labelling a more technical explanation as irrelevant could sound as they did not understand (which in fact they did not) and they did not want to be perceived as dummies, reinforcing the argument that non-technical participants tend to give neutral scores to technical explanation.

Provider and Visual Aids

At least 20% of the participants highlighted the importance of the provider to determine the correct function signature to choose. The provider is the company or institution that provides the service or functionality. Icon/provider plays an important role in the identification of the intent function call.

7.5 Summary

This chapter presented a user-centred analysis of the interpretability of different types of explanations in relation to different types of users in the context of the proposed semantic parser. In the setting involving the complex multi-component system, our experiment showed explanations are an effective method to build mental models in the given task, regardless of the users' technical background.

The experiment also suggested that technical knowledge is boosted with the support of explanations, as the results show higher correlation between machine learning knowledge and the mental model score in the treatment group in comparison to the control group.

Simple explanations associating the input provided by the user to the features of the system are the more useful type of information for a general audience. On the other hand, even low technically explanations, such as plotting a τ -SNE chart or a grammar tree, demands a previous grasp to attract the user interest.

Part V

Conclusions and Perspectives

Conclusions and Future Research

Contents

8.1	Summary	119
8.2	Discussion and Conclusions	121
8.2.1	Semantic Parsing	121
8.2.2	Explanation Model	123
8.3	Revisiting the Principles and Requirements of End-User Programming	124
8.3.1	Motivational Task	124
8.3.2	Principles	125
8.3.3	Challenges	126
8.4	Future Research Directions	129
8.4.1	Enriching the Training Data	129
8.4.2	Further Research on Explainability	130

8.1 Summary

This thesis was dedicated to the task of semantic parsing of natural language commands, proposing a method that has low dependency on training data. Additionally, our research defined a hierarchical explanation model, which was analysed from a user-centred perspective.

The chapters can be summarised as follows:

- After defining the task and the research questions and hypotheses in Chapter 1, we presented two comprehensive surveys in Chapters 2 and 3 describing prominent approaches, data sets and tasks respectively on end-user development and semantic parsing. Additionally, whereas, Chapter 2 introduced the concept of explainable artificial intelligence and discussed its related work in the context of our research, Chapter 3 exposed a gap in the literature, which pointed to semantic parsers with low dependency on large training sets. This gap and the rising importance of explainability of AI methods defined the focus of our research.

- Chapter 4 proposed a multi-component semantic parsing approach composed of a shallow parser that associates natural language commands to predicate-argument structures, integrated to a ranking model supported by semantic relatedness metrics over a word embedding model. The proposed method starts by shallow parsing the natural language command, generating a predicate-argument structure, in which the following components append additional semantic information. The predicate-argument structure stores the natural language command as a tuple composed of a *function descriptor*, which is the minimal subset of tokens present in the command that allows identifying the target function signature in the **API Knowledge Base**, and a set of *command objects*, which represent potential descriptors or parameters or their values. The command objects are later processed to receive extra semantic labels, such as tags of named entities. At the centre of the proposed solution are the semantic hyperspaces, on which both the attributes of the predicate-argument structure and the set of potential function signatures are projected using their word-embedding vectorial representations. Those semantic hyperspaces allow the system to calculate semantic similarity measures as described in Section 4.4.6, which serve as features to classify the *intent score*, i.e. the degree to which a given function call represents the user intent expressed in the natural language command (Section 4.4.7). As a result, the intent score and the set of features feed a final ranking component that defines the relative relevance of the function calls for the final user (Section 4.4.8).
- Chapter 5 defined an explanation model for our parser. The method is organised in a hierarchical fashion, starting with easy-to-read explanations, i.e. those that do not demand technical background, followed by the more advanced ones, to which a proper background can highly favour the understanding. The explanations are associated to the components, and combine both transparency-based and ad-hoc-based methods.
- Chapter 6 was devoted to the evaluation of the semantic parsing, considering a gold standard data set. Before evaluating our proposed method, we demonstrated the low performance of encoder-decoder based architectures to deal with our target task, given the low availability of training data. The experiments evaluated both sequence-to-sequence and attention-based architectures in different settings, varying the number of layers, the input data representation and other sensitive hyper-parameters. All the evaluated models delivered an f1-score of 0. To evaluate our proposed architecture from different points of view, we instantiated several implementations for both the **API Fil-**

ter and the Intent Classifier. For the filtering function, we evaluated TF/IDF, nearest neighbours and the identity function. *Random Forest*, *Support Vector Machine* and *Multilayer Perceptron* were the evaluated implementations for the classifier. The proposed method operating with a nearest-neighbours filter and a Random-Forest classifier achieved the best results with a recall of 0.682 and a mean reciprocal rank of 0.303 for the TOP-10 results over a knowledge base of 2005 distinct function signatures.

- In Chapter 7, we analysed from a user-centred perspective the utility of different types of explanations and the impact of background knowledge on the user preferences. The experiments provide sufficient evidence that explanations are an effective method to build mental models, regardless of the users' technical background. Our experiments suggested that simple explanations associating the input provided by the user to the features of the system are the more useful type of information for a general audience. On the other hand, even low technical explanations, such as plotting a *t-SNE* chart or a grammar tree, demands a previous grasp to attract the user interest. The experiment also suggests technical knowledge is boosted when accompanied by explanations, given its high correlations with mental model scores. This work also assumes that technical explanations are more suitable for debugging tools, as the level of effort necessary to properly understand them demands higher level of attention and reasoning capabilities.
- Finally, in addition to this summary, this chapter discusses the results of the experiments under the light of the hypotheses and the research questions (Section 8.2), revisits the principles and challenges for the development of an effective end-user programming environment (Section 8.3), and concludes presenting the future research directions opened by our research (Section 8.4).

8.2 Discussion and Conclusions

8.2.1 Semantic Parsing

We analysed the performance of our proposed semantic parser focusing on two research questions, which are now discussed in the light of the obtained results.

Research Question 1.1

How suitable are encoder-decoder machine learning models to build a semantic parser for our target task, considering the low

availability of training data?

Experimental support: Evaluation of several end-to-end encoder-decoder neural network architectures based on both sequence-to-sequence models and attention mechanisms for the given task, whose results are described in Section 6.3.

Interpretation: We instantiated a comprehensive set of sequence-to-sequence and attention-based models, with variations in several hyper-parameters as described in Section 6.3. None of the instantiated models were capable to provide any satisfactory output. The experiments confirm the well-known requirement, to which Du et al. (2018) provided the theoretical foundation, that a deep learning machine model depends on large data sets, given the high number of parameters in their weight matrices.

Research Hypothesis 1.2

The combination of a specialised shallow parser with type-inference capabilities and a ranking model supported by relatedness measures on a word embedding model can be used to define a semantic parser for natural language commands under the restriction of low availability of training data.

Experimental support: Our proposed multi-component semantic parsing method operating with a Nearest Neighbours API Filter and a Random Forest Intent Classifier was able to solve up to 68% of the commands considering the TOP-10 evaluation scenario, and up to 85% when considering the TOP-50. The results are shown in detail in Tables 6.3 and 6.4.

Interpretation: The multi-component parser can mitigate the lack of training data by relying on curated linguistic data sets and models, to define a shallow parser, able to produce a predicate-argument representation for the natural language command, associated to a ranking model supported by semantic relatedness metrics over a word embedding model. In our approach, a shallow parser identifies the main linguistic elements to match a natural language command to a function signature (Section 4.4.2). Additionally, both Named Entity Recogniser (Section 4.4.3) and Reported Speech Detector (Section 4.4.3) were able to append semantic labels to the list of potential descriptors or values of parameters. The final configuration optimised the use of the training data, reserving it to solely learn the intent score (Section 4.4.7).

8.2.2 Explanation Model

We analysed our user-centred study on explainability focusing on three research questions, which are now discussed in the light of the obtained results.

Research Question 2.1

To what extent are users able, irrespective of their technical background, to improve their mental models by associating the linguistic features from the explanations to the system's behaviour?

Experimental support: The mental-model experiment with the treatment and control groups to measure whether the users realise the correct influence of linguistic features in the overall performance of the parser, whose results are shown in Table 7.2.

Interpretation: We divided this research question into two parts. The first analyses the ability of the explainable model to improve the mental model of the end users, while the second analyses the impact of the technical background on the user perceptions. The results of the experiments showed that the users exposed to the explanations demonstrated a better understanding of the system's working mechanisms. When analysing subgroups, according to their machine learning and grammatical competence in English, the results consistently demonstrate improvement in the understanding, regardless of the level of knowledge in any of the areas. Additionally, the results suggest that the knowledge in machine learning boosts the explanations in helping the understanding of the system.

Research Question 2.2

How does the knowledge in machine learning affect the preference for technical explanations?

Experimental support: Among the different types of explanations, we asked the participants, grouped as acquainted and non-acquainted to machine learning, the level of helpfulness of each of them, as summarised in Table 7.3.

Interpretation: After been exposed to seven different types of explanations, each of them associated to one component of the architecture, the results showed that the participants prefer explanations that require simple reasoning using logic, plain mathematics and linguistic features. When we analyse the score inter-groups, the results show that those simpler explanations are comparatively better assessed by the participants. The results

concerning the relevance of both comparative explanation and the explanation for the intent classifier present an interesting consistency, showing insignificant statistical divergence between the groups. On the other hand, participants acquainted with machine learning appear to be more satisfied with simpler explanations and are rather sceptical to more technical ones when compared to the group of non-acquainted.

Research Question 2.3

To what degree are users willing to favour explainability over accuracy given a certain context?

Experimental support: We asked a set of questions to both treatment and control groups regarding their preferences on explainability *vs.* accuracy in three scenarios, namely, *a semantic parsing*, in the case of *a cancer exam* of a relative and in the case of *a bank loan decision*, as summarised in Table 7.4.

Interpretation: In general, participants favour accuracy over explainability. There is however a significant distinction between the scenarios. We conjecture that, as a command selection tool is perceived as non-critical and harmless, whereas a cancer screening, despite having a high impact on the subjects' lives, is perceived as technical, and so, less prone to errors, the users tend to favour accuracy over explainability in a more intensive degree when compared to the bank loan scenario. We speculate there is a social awareness of the possible bias of financial agents that incentivises the participants to be more interested in an explanation for a loan decision.

8.3 Revisiting the Principles and Requirements of End-User Programming

In this section, we revisit the principles and requirements for the development of end-user programming environments considering the insights obtained during the development of our research.

8.3.1 Motivational Task

For our analysis, consider the following motivational task, which represents a request to administrative staff in a hypothetical language school.

“Please, send an email to each student from Table 8.1. Calculate their bills, where each class costs €80, converting the price to their preferred currency. For those in the A1 or A2 levels, please also attach a translated message in their mother languages.”

Name	Email	Course Level	# of Classes	Currency	Mother Language
John	john@smith.com	A1	13	Chinese Yuan	Chinese
Kim	kim@korea.com	A2	17	Korean Won	Korean
Tarsila	tarsila@ama.br	B2	10	Brazilian Real	Portuguese
...
Frida	frida@kahlo.mx	A2	15	Mexican Peso	Spanish

Table 8.1: Tabular content within an email message.

Processing this task manually for a reasonable long table can demand a few hours of work, although being easily automated if a proper tool and services are available.

8.3.2 Principles

Writability, *readability* and *expressivity* have been the main criteria to evaluate programming languages (Sebesta, 2012). Contrary to the first thought, we need to resist the temptation of assuming that NL programming would address these issues *by nature*. Indeed, some high-level instructions in natural language look clearer and straightforward, but this is not always the case. For instance, long mathematical expressions described in plain text decrease both readability and writability when compared to the algebraic notation (Dijkstra, 1979). Data manipulation, an aspect neglected in the mainstream end-user discussions, cannot receive proper support from a purely natural language interface as well (Desolda et al., 2017). Looking at EUD as more than a language, but as a platform, allows us to overcome these restrictions by using hybrid approaches as we further describe.

Reliability, also commonly accounting for general programming languages evaluation (Sebesta, 2012), assumes a different sense in the context of NL programming too. As natural language interpreters and parsers need to handle an open lexicon, they always bring a degree of uncertainty (Li and Du, 2017). Being reliable in this context means offering to the end user a graceful mechanism to identify and repair any AI misclassifications (Sales et al., 2018a; Schlegel et al., 2019). For example, when a user types a command and one word is interpreted in a sense different from the user’s intention, the output will be incorrect. Any interactive system supported by AI needs to take into account the need to provide a *graceful repairing* mechanism.

Moreover, the *information overload* that we currently face cannot be ignored (Roetzel, 2018). This phenomenon is not only regarded in books, news, scientific papers and social media posts, but also to *code*. As of 2018,

GitHub¹ hosts more than 100 million repositories, each one holding potentially a library (Jason Warner, 2018). Professional developers already need to make use of specialised blogs and tech social media such as Stack Overflow² to find the intended library or API call. Given its nature, any EUD platform must offer suitable mechanisms to help the user to browse and find the pertinent functions.

Despite integrating data being a fundamental step to automate repetitive tasks, EUD platforms usually are short of any data manipulation tools (Desolda et al., 2017). In addition to deal with structured data sources, a new fundamental requirement is the integration of data in the unstructured form, since by 2022, 93% of the information generated is expected to be unstructured (IDG, 2016), e.g. texts and also semi-structured data such as Table 8.1.

8.3.3 Challenges

An effective EUD tool cannot be only a compiler for a certain syntax but also a platform offering three main features. First, the platform needs to allow end users to find and execute both single and composed high-level functions natively integrated with their service providers. Second, it needs to supply tools to deal with unstructured and semi-structured data. And finally, the platform must offer an embedded explanation mechanism.

The failure of the first natural programming languages may be attributed to the attempt to address general-purpose programming, believing they could act as Java or Python (Biermann et al., 1983). Indeed, as the typical end users think at the level of business concepts and processes, the main advantage of those languages is rather related to their capacity to operate at a higher level of abstraction, connecting coarse-grained pieces of code, a concept well-established in the fields of *Service-oriented Architecture* and *Business Process Management* (Pourmirza et al., 2019). For instance, the reference task lists three coarse-grained functions: *send an email*, *convert currency* and *translate a text*, representing the functions end users see as atomic in that context.

Information Extraction: The Data

The success of spreadsheets, the most popular EUD platform, can be credited to their ability to fuse data and functions gently (Birch et al., 2017). An effective EUD platform needs to have built-in support for data manipulation of complex data types. To a large extent, the unstructured nature of the information we deal daily with forms the main barrier to advance the

¹<https://github.com>

²<https://stackoverflow.com>

automation of our tasks, since demanding the end users to structure large data sets manually cuts significantly the benefit of automation.

In this context, *information extraction* (IE) becomes a key tool in end-user platforms. Usually based on artificial intelligence techniques, IE transforms a text into a knowledge graph from which a function can identify and process its content easier (Cetto et al., 2018).

Search & Run: The Method

In an environment where end-user developers can use services from many sources and providers, finding the appropriate function becomes challenging, resulting in a scenario that represents the *information overload* phenomenon for code.

Inspired by feature-rich applications that relied on searching to prevent the user from navigating with intricate menu hierarchies to find the desired function (Bota et al., 2018), we advocate for a method also focused on search. Furthermore, given the potential scale of millions of functions, we need to power a search engine with semantic capabilities, especially to deal with the vocabulary gap and ambiguity.

As natural languages allow expressing the same idea using not only different words but also different syntactic structures, having an efficient search engine is mandatory to disambiguate related services and correctly deal with the vocabulary gap between the way the users express themselves and the way the service is described in the repository. For instance, a user might write “*exchange money*” to find a service that is described as “*convert currency*” (Sales et al., 2017). Regarding ambiguity, using natural language command as search queries provides not only an initial description for its functions, but also contextual information that might be used as parameters’ names or values, thus serving to disambiguate services sharing similar concepts (Sales et al., 2018a).

After finding the desired function, the end user can execute it. The data produced in each execution needs to be shown to the user but also stored as contextual information to be potentially used by subsequent commands. The end user can then define new functions by “*chaining*” a sequence of previously executed commands, by means of interlinking the output of a command to the input of the subsequent. After receiving a name, the end user can make the new function available to be resolved by the semantic parser. For example, the sequential execution and chaining of *send an email*, *convert currency* and *translate a text* can generate the new function “*notify student*”, which can be reused with less effort.

Such a method favours a gentle learning curve from two aspects. Novice users find easier to understand stateless programming (Krishnamurthi and Fisler, 2019), but to guarantee a pace of learning, the platform needs to allow also a simple transition to writing *stateful code*. When using a method

based on search, the end user can make this transition naturally, as the same mechanism is used for both the execution of a single command and the construction of functions. Secondly, showing functions in a ranked list dialogues with the principle of *graceful repairing*, thus reducing the effect of misunderstanding and performance error of the parser (Sales et al., 2018a). Furthermore, the ranked list also allows simple mechanisms to learn from the user experience without any feedback overloading the user.

Hybrid Interface: The Fusion

None of the interaction modes, i.e. textual language, visual language or programming by example, can address alone all expressivity issues of end-user developers. Rather, an effective EUD platform requires multiple mechanisms to tackle appropriately the diverse types of user interactions. Whereas natural languages cover the identification of coarse-grained functions with a semantically-empowered search mechanism, the platform needs to provide proper mechanisms to describe simple computations and to integrate data.

When defining a new function from a chain of commands, as shown in the reference example, the output of one might mismatch (i.e. data type or format) to the input of the subsequent. This is the typical duty of scripting languages focused on performing simple computations, such as mathematical expressions and string transformations, like those used in spreadsheets applications. The mathematical formalist in this scenario represents a gain in expressivity (Dijkstra, 1964).

Regarding data integration, both visual tools and programming by example can play important roles. On the one hand, visual tools can significantly help end users manipulating and understanding the data. Pane and Myers (2006) showed that users benefit when data is represented in visual cards that may be traversed manually, or by textual filters. Additionally, it also serves as an important debugging mechanism for the (potentially error-prone) information extraction tools. On the other hand, when importing semi-structured data, comprehending table into texts or content following a standard format, users can benefit from programming-by-example techniques with high performance.

User's Idiosyncrasies: The Human Factor

A tool with the impressive 95% of accuracy fails, in average, every 20 uses, and the impreciseness of intelligent systems can lead to users' frustration and lack of interest in the tool (Thomason et al., 2015). Despite the significant improvement in semantic-based technologies in recent years, the level of complexity inherent to natural language requires that solutions be aware of their (still high) likelihood of failure.

To safeguard user's attention, every AI-empowered functionality needs to

provide *graceful repairing* mechanisms to allow proper corrections of misunderstandings in a simple and easy manner. The search approach to identify functions, for example, goes in this direction by giving flexibility to the user to select the correct function among the priorities.

Considering the freedom of expressivity that a natural language allows, the programming model needs to learn the particular user’s writing style. For example, while some users might use complete and grammatically sound sentences to express their commands, others might insist on a keyword-based approach. The same occurs when the user frequently writes a vague word to express a given concept. Learning from the user’s history of use is a key component to deliver quality results.

Explainability: The Transparency

Although much of the debate around explanation for AI systems have been concentrated on decision-making algorithms, mainly guided by the rights “*for meaningful information about the logic involved*” and “*to non-discrimination*” both defined in the European Union’s General Data Protection Regulation (Parliament and Council of European Union, 2016), other types of AI systems can also benefit from explanations. Biran and Cotton (2017) show a set of studies suggesting where users feel much more confident using a system they understand how it works. Explanation for AI has become a point of no return for many intelligent systems.

8.4 Future Research Directions

The research described in this thesis points to the construction of an end-user programming environment mainly supported by a natural language interface, integrated with visual elements, as presented in Section 8.3.

Given the importance of data sets to produce and improve natural language understanding systems, in Section 8.4.1, we also highlight the potential of transfer learning techniques to enlarge training data in general, and alternative methods to improve the quality and reduce the cost to obtain new training data in our target task.

At the end, Section 8.4.2 is devoted to debate future research regarding the explainability of semantic parsing methods.

8.4.1 Enriching the Training Data

We are facing the unprecedented proliferation of learning representation models, where the increasing size of the data sets can be highly beneficial (Du et al., 2018). We foresee three strategies that can be used to improve the quality and reduce the costs of obtaining data for our target task.

Transfer Learning Methods

The high availability of resources from other natural language processing task may favour the improvement of any semantic model when using transfer learning techniques. Transfer learning is the ability to extract knowledge from a data set initially meant for another task (Pan and Yang, 2010).

In addition to other data sets targeting semantic parsing in other domains, data sets associated with question answering or text entailment might be beneficial given two important similarities such as the `SimpleQuestions` data set (Bordes et al., 2015) and the `Boeing-Princeton-ISI Textual Entailment` data set (Silva et al., 2019). Both tasks usually deal with short utterances and, except for the tendency of having imperative mood, the variability in vocabulary and grammar structure tends to be similar.

Collecting as Using

A functional prototype can serve as a source of real-world data from users that accept to use a beta version. Once there is an initial implementation of a minimum viable platform, this strategy of data collection can be immediately applied.

“Don’t Think of a White Elephant”

When a person is asked to suppress a certain thought such as thinking of a white elephant, the psychological effect is in fact making it more likely to surface, which is called *ironic rebound* (Wegner and Schneider, 2003).

A similar effect occurs when asking one to describe a natural language command to *search for a yellow monkey video on YouTube*. By describing the task in natural language, we induce the use of the same vocabulary and grammar structure. One alternative is to describe the task using images, such as Figure 8.1, in which the user is expected to write a chain of commands to search for a product, convert its price to another currency, and finally sending it by `Skype` to someone else. Figure 8.2 presents another possible scenario including searching and posting in social medias.

The method can also open the doors for the collection of more complex composition of commands, that goes beyond single commands or if-then recipe, in line with the revisited principles and challenges described in Section 8.3.

8.4.2 Further Research on Explainability

Concerning explainability, there are two important aspects that deserve our attention for future research: an investigation on the notion of completeness as well as the need for interactive and customised explanations.



Figure 8.1: An example of a multiple-command scenario composed of three commands. The chart induces the user to write a command to search for “iPhone” in Amazon followed by exchanging its price from Euro to US Dollar, and finally sending the calculated price to the user @MyMother in Skype.

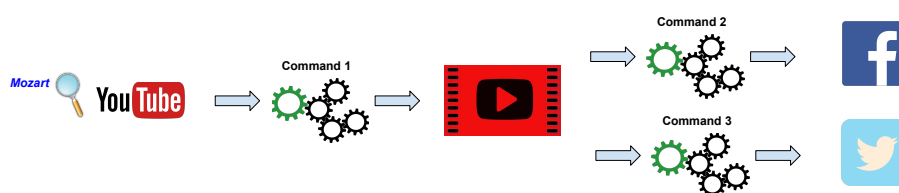


Figure 8.2: A chart describing a chain of a command to search a video on YouTube, followed by a pair of commands to share it on social media platforms.

The Notion of Completeness

In the context of data quality (Pipino et al., 2002), completeness is usually regarded as *intuitive*. For instance, if you have a database of clients, which doesn’t contain an entry for a given known client, the data set isn’t complete. The intuitiveness comes from the simplicity of identifying the whole, and to check if the object of study, in this case the database, covers it *completely*. However, this is not always the case. In a context of knowledge management, when making logic inference over common-sense knowledge, we can assume its completeness only under the theoretic assumption of closed-world (Reiter, 1981). It is not feasible to cover the entire common-sense knowledge in a database.

The notion of completeness for explanations are closer to the second example. There is a gap in the literature regarding the measurement of completeness and the evaluation of soundness of an explanation. Our research, as well as the work of Kulesza et al. (2015), assume the explanations are complete and sound without any metric or experimentation to ground it. From the user perspective, which are the requirements of a given explanation, considering the task and the method, to be considered complete and sound?

Interactive and Customised Explanations

Current explanation models are mostly *static*, in the sense that there is no interactive element for the user (Abdul et al., 2018). Given the lack of interaction, as a side effect, the models define a unique message to the users. However, users observe the world and systems with different eyes (Jones et al., 2011).

Aligned with the challenges of the field of Human Computer Interaction (HCI) (Abdul et al., 2018), there is a research space to advance the field towards customised explanations. This research direction points to define explanation models that offer interactive mechanism based on *space explorations* (Vermeulen, 2014) or *conceptual models for implicit interactions* (Ju, 2015) where the end users navigate and adapt the explanations to better help their understanding.

Our explanation model partially addresses this issue by its hierarchical model, offering different levels of explanation aiming at satisfying different kinds of users. The challenge, however, is to allow that all layers of explanations present interactive elements to allow the end users to adapt the explanation to different perspectives.

Appendices

APPENDIX A

Questionnaire

This Appendix describes the questionnaire used in the user evaluation of the explanation model. It was equally applied to both target and control group, except for Section A.2, which was exclusively applied to the first group.

A.1 Mental Models

For each scenario below defined, consider the commands and choose one option for each statement, following the Likert 7-point scale (Likert, 1932):

1. *strongly disagree*;
2. *disagree*;
3. *somewhat disagree*;
4. *neither agree nor disagree*;
5. *somewhat agree*;
6. *agree*;
7. *strongly agree*.

Scenario 1.1: For the command: I'd like, please, to exchange 100 euro to swiss francs.

Questions	1	2	3	4	5	6	7
Writing the command in the imperative form increases the system comprehension.							
Writing "Euro" with capital letter increases the system comprehension.							
Changing 100 for "one hundred" increases the system comprehension.							
Writing "Swiss Francs" with capital letters increases the system comprehension.							
Starting by "I'd like" increases the system comprehension.							

Scenario 1.2: For the command: directions new york city to los angeles.

Questions	1	2	3	4	5	6	7
Writing the command as a grammatically correct sentence increases the system comprehension.							
Writing "New York City" with capital letter increases the system comprehension.							
Writing a set of keywords has the same result as grammatically correct sentences.							
Writing "Los Angeles" with capital letters increases the system comprehension.							
Changing "directions" for "distance" makes no difference in the command.							

Scenario 1.3: For the command: Could the system translate doc1.txt from english to portuguese?

Questions	1	2	3	4	5	6	7
Writing the command in the imperative form increases the system comprehension.							
Writing "English" with capital letter increases the system comprehension.							
Writing the command as a question has the same result as the imperative form.							
Writing "Portuguese" with capital letter increases the system comprehension.							

A.2 Types of Explanation

The expression "*strongly disagree*" means you didn't find the given explanation useful at all, whereas "*strongly agree*" means the explanation is highly useful to understand the system behaviour.

Scenario 2.2: I found helpful to see ...

Questions	1	2	3	4	5	6	7
how parameters' values are identified.							
the grammar tree.							
the comparison explanation.							
the representation of word vectors in a chart.							
the importance of features in the relevance classification.							
the score expression.							
the features' values.							

A.3 Accuracy over Interpretability

In our context, the higher the accuracy, the better the system's results, while interpretability means your ability to understand the system's mechanism.

Question 3.1: To what degree would you favour accuracy over interpretability for a command selection tool?

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Table A.1

Question 3.2: To what degree would you favour accuracy over interpretability for ...

	1	2	3	4	5	6	7
a cancer exam of a relative?							
a bank loan decision?							

Table A.2

Bibliography

- Ashraf Abdul, Jo Vermeulen, Danding Wang, Brian Y. Lim, and Mohan Kankanhalli. Trends and trajectories for explainable, accountable and intelligible systems: An hci research agenda. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18*, pages 582:1–582:18, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3174156. URL <http://doi.acm.org/10.1145/3173574.3174156>.
- A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2870052.
- Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL '09*, pages 19–27, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-41-1. URL <http://dl.acm.org/citation.cfm?id=1620754.1620758>.
- Alfred V Aho and Jeffrey D Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, 1 edition, 1972.
- Tom M. Mitchel Amos Azaria, Jayant Krishnamurthy. Instructable intelligent personal. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16*, 2016.
- Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Snderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- Jacob Andreas and Dan Klein. Grounding language with points and paths in continuous spaces. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 58–67, 2014.
- Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1165–1174,

- Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1138. URL <https://www.aclweb.org/anthology/D15-1138>.
- Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.
- Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013. doi: 10.1162/tacl.a.00209. URL <https://www.aclweb.org/anthology/Q13-1005>.
- Yoav Artzi, Dipanjan Das, and Slav Petrov. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1273–1283, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1134. URL <https://www.aclweb.org/anthology/D14-1134>.
- Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. *DBpedia: A Nucleus for a Web of Open Data*, pages 722–735. Springer, Berlin, Heidelberg, 2007.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project. In *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1, COLING '98*, pages 86–90, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. doi: 10.3115/980451.980860. URL <https://doi.org/10.3115/980451.980860>.
- Bruce W. Ballard and Alan W. Biermann. Programming in natural language: “nlc”; as a prototype. In *Proceedings of the 1979 Annual Conference*, ACM '79, pages 228–237, New York, NY, USA, 1979. ACM. ISBN 0-89791-008-7. doi: 10.1145/800177.810072. URL <http://doi.acm.org/10.1145/800177.810072>.
- Antonio Valerio Miceli Barone and Rico Sennrich. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 314–319, 2017.

- Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, 149:101 – 137, 2019. ISSN 0164-1212. doi: <https://doi.org/10.1016/j.jss.2018.11.041>. URL <http://www.sciencedirect.com/science/article/pii/S0164121218302577>.
- Emanuele Bastianelli, Giuseppe Castellucci, Danilo Croce, Luca Iocchi, Roberto Basili, and Daniele Nardi. Huric: a human robot interaction corpus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may 2014. European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4.
- I. Beltagy and Chris Quirk. Improved semantic parsers for if-then statements. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, Berlin, Germany, 2016. URL <http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127577>.
- Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. Representational capacity of deep neural networks: A computing study. *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2019. doi: 10.5220/0008364305320538. URL <http://dx.doi.org/10.5220/0008364305320538>.
- Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.
- Alan W. Biermann, Bruce W. Ballard, and Anne H. Sigmon. An experimental study of natural language programming. *International Journal of Man-Machine Studies*, 18(1):71 – 87, 1983. ISSN 0020-7373. doi: [https://doi.org/10.1016/S0020-7373\(83\)80005-4](https://doi.org/10.1016/S0020-7373(83)80005-4). URL <http://www.sciencedirect.com/science/article/pii/S0020737383800054>.
- Or Biran and Courtenay Cotton. Explanation and justification in machine learning: A survey. In *IJCAI-17 Workshop on Explainable AI (XAI)*, page 8, 2017.
- David Birch, David Lyford-Smith, and Yike Guo. The future of spreadsheets in the big data era. In *Proceedings of the EuSpRIG 2017 Conference “Spreadsheet Risk Management”*, 2017. ISBN: 978-1-905404-54-4.

- Alan F. Blackwell. *End-User Developers – What Are They Like?*, pages 121–135. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60291-2. doi: 10.1007/978-3-319-60291-2_6. URL https://doi.org/10.1007/978-3-319-60291-2_6.
- Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics, COLING '10*, pages 89–97, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=1873781.1873792>.
- Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4349–4357. Curran Associates, Inc., 2016.
- Tracey Booth and Simone Stumpf. End-user experiences of visual and textual programming environments for arduino. In Yvonne Dittrich, Margaret Burnett, Anders Mørch, and David Redmiles, editors, *End-User Development*, pages 25–39, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38706-7.
- Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015. URL <http://arxiv.org/abs/1506.02075>.
- Alessio Bosca, Fulvio Corno, Giuseppe Valetto, and Roberta Maglione. On-the-fly construction of web services compositions from natural language requests. *Journal of Software*, 1(1):40–50, 2006.
- Horatiu Bota, Adam Fourney, Susan T. Dumais, Tomasz L. Religa, and Robert Rounthwaite. Characterizing search behavior in productivity software. In *Proceedings of the 2018 Conference on Human Information Interaction & Retrieval, CHIIR '18*, pages 160–169, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4925-3. doi: 10.1145/3176349.3176395. URL <http://doi.acm.org/10.1145/3176349.3176395>.
- Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.

- Daniel Brooks, Constantine Lignos, Cameron Finucane, Mikhail Medvedev, Ian Perera, Vasumathi Raman, Hadas Kress-Gazit, Mitch Marcus, and Holly Yanco. Make it so: Continuous, flexible natural language interaction with an autonomous robot. In *Proceedings of the 2012 AAAI Workshops*, 2012. URL <https://www.aaai.org/ocs/index.php/WS/AAAIW12/paper/view/5244/5583>.
- Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis, 2018.
- Andrew Burgess. *AI Prototyping*, pages 117–127. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63820-1. doi: 10.1007/978-3-319-63820-1_7. URL https://doi.org/10.1007/978-3-319-63820-1_7.
- José Camacho-Collados and Mohammad Taher Pilehvar. From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788, 2018. doi: 10.1613/jair.1.11259. URL <https://doi.org/10.1613/jair.1.11259>.
- Ozan Arkan Can and Deniz Yuret. A new dataset and model for learning to understand navigational instructions, 2018.
- Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, and Steven M. Hadfield. Raptor: A visual programming environment for teaching algorithmic problem solving. *SIGCSE Bull.*, 37(1):176180, February 2005. ISSN 0097-8418. doi: 10.1145/1047124.1047411. URL <https://doi.org/10.1145/1047124.1047411>.
- R. Caruana, H. Kangarloo, J. D. Dionisio, U. Sinha, and D. Johnson. Case-based explanation of non-case-based learning methods. *Proc AMIA Symp*, pages 212–215, 1999. ISSN 1531-605X. URL <https://www.ncbi.nlm.nih.gov/pubmed/10566351>. D005808[PII].
- Matthias Cetto, Christina Niklaus, André Freitas, and Siegfried Handschuh. Graphene: Semantically-linked propositions in open information extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2300–2311. Association for Computational Linguistics, 2018. URL <http://aclweb.org/anthology/C18-1195>.
- David L. Chen. Fast online lexicon learning for grounded language acquisition. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 430–439, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=2390524.2390584>.

- David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 859–865. AAAI Press, 2011. URL <http://dl.acm.org/citation.cfm?id=2900423.2900560>.
- Howard Chen, Alane Suhr, Dipendra Misra, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Conference on Computer Vision and Pattern Recognition*, 2019.
- Robert A. Cochran, Loris D'Antoni, Benjamin Livshits, David Molnar, and Margus Veanes. Program boosting: Program synthesis via crowdsourcing. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 15, page 677688, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333009. doi: 10.1145/2676726.2676973. URL <https://doi.org/10.1145/2676726.2676973>.
- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL <https://doi.org/10.1023/A:1022627411411>.
- Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 43–48, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075823. URL <https://doi.org/10.3115/1075812.1075823>.
- A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, S. R., and S. Roy. Program synthesis using natural language. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 345–356, May 2016. doi: 10.1145/2884781.2884786.
- Giuseppe Desolda, Carmelo Ardito, Maria Francesca Costabile, and Maristella Matera. End-user composition of interactive applications through actionable ui components. *Journal of Visual Languages & Computing*, 42:46 – 59, 2017. ISSN 1045-926X. doi: <https://doi.org/10.1016/j.jvlc.2017.08.004>. URL <http://www.sciencedirect.com/science/article/pii/S1045926X17300125>.
- E. W. Dijkstra. Some comments on the aims of mirfac. *Commun. ACM*, 7(3):190–, March 1964. ISSN 0001-0782. doi: 10.1145/363958.364002. URL <http://doi.acm.org/10.1145/363958.364002>.

- Edsger W. Dijkstra. On the foolishness of "natural language programming". In *Program Construction, International Summer School*, pages 51–53, London, UK, UK, 1979. Springer-Verlag. ISBN 3-540-09251-X. URL <http://dl.acm.org/citation.cfm?id=647639.760596>.
- Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, 2016.
- Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, 2018.
- Simon S. Du, Yining Wang, Xiyu Zhai, Sivaraman Balakrishnan, Ruslan Salakhutdinov, and Aarti Singh. How many samples are needed to estimate a convolutional or recurrent neural network?, 2018.
- John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- Kais Dukes. Semantic Annotation of Robotic Spatial Commands. In *Language and Technology Conference (LTC)*, 2013. URL http://scholar.google.de/citations?view_op=view_citation&hl=en&user=wmRD1-4AAAAJ&citation_for_view=wmRD1-4AAAAJ:YsMSGLbicyi4C.
- Kais Dukes. Semeval-2014 task 6: Supervised semantic parsing of robotic spatial commands. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 45–53, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University. URL <http://www.aclweb.org/anthology/S14-2006>.
- S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '88*, pages 281–285, New York, NY, USA, 1988. ACM. ISBN 0-201-14237-6.
- Felix Duvallet, Matthew R Walter, Thomas Howard, Sachithra Hemachandra, Jean Oh, Seth Teller, Nicholas Roy, and Anthony Stentz. Inferring maps and behaviors from natural language instructions. In *Experimental Robotics*, pages 373–388. Springer, 2016.
- Empirica. High-Tech Leadership Skills for Europe. Technical report, Empirica, 2017.

- Kurt Englmeier, Javier Pereira, and Josiane Mothe. Choreography of web services based on natural language storybooks. In *Proceedings of the 8th International Conference on Electronic Commerce: The New e-Commerce: Innovations for Conquering Current Barriers, Obstacles and Limitations to Conducting Successful Business on the Internet*, ICEC '06, pages 132–138, New York, NY, USA, 2006. ACM. ISBN 1-59593-392-1. doi: 10.1145/1151454.1151485. URL <http://doi.acm.org/10.1145/1151454.1151485>.
- G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev. Meta-design: A manifesto for end-user development. *Commun. ACM*, 47(9): 3337, September 2004. ISSN 0001-0782. doi: 10.1145/1015864.1015884. URL <https://doi.org/10.1145/1015864.1015884>.
- Rita Francese, Michele Risi, and Genoveffa Tortora. Iconic languages: Towards end-user programming of mobile applications. *Journal of Visual Languages & Computing*, 38:1 – 8, 2017. ISSN 1045-926X. doi: <https://doi.org/10.1016/j.jvlc.2016.10.009>. URL <http://www.sciencedirect.com/science/article/pii/S1045926X1530063X>. SI:In honor of Prof SK Chang.
- André Freitas, Siamak Barzegar, Juliano Efon Sales, Siegfried Handschuh, and Brian Davis. Semantic relatedness for all languages: A comparative analysis of multilingual semantic relatedness using machine translation. In *20th International Conference on Knowledge Engineering and Knowledge Management-Volume 10024*, pages 212–222, 2016.
- G. W. Furnas et al. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, November 1987. ISSN 0001-0782.
- Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- Matt Gardner, Pradeep Dasigi, Srinivasan Iyer, Alane Suhr, and Luke Zettlemoyer. Neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 17–18, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-5006. URL <https://www.aclweb.org/anthology/P18-5006>.
- David J. Gilmore, Karen Pheasey, Jean Underwood, and Geoffrey Underwood. *Learning graphical programming: An evaluation of KidSimTM*,

- pages 145–150. Springer US, Boston, MA, 1995. ISBN 978-1-5041-2896-4. doi: 10.1007/978-1-5041-2896-4_24. URL https://doi.org/10.1007/978-1-5041-2896-4_24.
- Alessandra Giordani and Alessandro Moschitti. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of COLING 2012: Posters*, pages 401–410, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL <https://www.aclweb.org/anthology/C12-2040>.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.
- Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38: 50–57, 2017.
- Giovanni Guida and Carlo Tasso. Nli: a robust interface for natural language person-machine communication. *International Journal of Man-Machine Studies*, 17(4):417 – 433, 1982. ISSN 0020-7373. doi: [http://dx.doi.org/10.1016/S0020-7373\(82\)80042-4](http://dx.doi.org/10.1016/S0020-7373(82)80042-4). URL <http://www.sciencedirect.com/science/article/pii/S0020737382800424>.
- Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany, August 2016. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P16-1014>.
- Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 11, page 317330, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450304900. doi: 10.1145/1926385.1926423. URL <https://doi.org/10.1145/1926385.1926423>.
- Sumit Gulwani. Programming by examples (and its applications in data wrangling). In *Verification and Synthesis of Correct and Secure Systems*. IOS Press, January 2016. URL <https://www.microsoft.com/en-us/research/publication/programming-examples-applications-data-wrangling/>.
- Sumit Gulwani and Mark Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 803–814, New York, NY, USA, 2014.

- ACM. ISBN 978-1-4503-2376-5. doi: 10.1145/2588555.2612177. URL <http://doi.acm.org/10.1145/2588555.2612177>.
- Sumit Gulwani, William R Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, 2012.
- Sumit Gulwani, Mikael Mayer, Filip Niksic, and Ruzica Piskac. Strisynth: synthesis for live programming. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 701–704. IEEE, 2015.
- Mark Guzdial. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, 8(6):1–165, 2015.
- Tihomir Gvero and Viktor Kuncak. Synthesizing java expressions from free-form queries. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015*, pages 416–432, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3689-5. doi: 10.1145/2814270.2814295. URL <http://doi.acm.org/10.1145/2814270.2814295>.
- A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, March 2009. ISSN 1541-1672. doi: 10.1109/MIS.2009.36.
- Sachithra Hemachandra, Felix Duvallat, Thomas M Howard, Nicholas Roy, Anthony Stentz, and Matthew R Walter. Learning models for following natural language directions in unknown environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5608–5615. IEEE, 2015.
- Matthew Henderson, Blaise Thomson, and Jason Williams. The second dialog state tracking challenge. In *Proceedings of SIGDIAL*. ACL Association for Computational Linguistics, June 2014a. URL <https://www.microsoft.com/en-us/research/publication/the-second-dialog-state-tracking-challenge/>.
- Matthew Henderson, Blaise Thomson, and Jason Williams. The third dialog state tracking challenge. In *Proceedings IEEE Spoken Language Technology Workshop (SLT)*. IEEE Institute of Electrical and Electronics Engineers, December 2014b. URL <https://www.microsoft.com/en-us/research/publication/the-third-dialog-state-tracking-challenge/>.

- Gary G Hendrix, Earl D Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems (TODS)*, 3(2):105–147, 1978.
- Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P. Bigham. Instructablecrowd: Creating if-then rules via conversations with the crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 1555–1562, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2892502. URL <http://doi.acm.org/10.1145/2851581.2892502>.
- IDG. Data & Analytics: Landscape in the Enterprise. Technical report, IDG, 2016.
- Jason Warner. Thank you for 100 million repositories. <https://github.blog/2018-11-08-100m-repos/>, 2018. Online; accessed 13 May 2019.
- Christopher Manning Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, October 2014.
- Van Bouwel Jeroen and Weber Erik. Remote causes, bad explanations? *Journal for the Theory of Social Behaviour*, 32(4):437–449, 2002. doi: 10.1111/1468-5914.00197. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-5914.00197>.
- Philip Nicholas Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press, 1983.
- Natalie Jones, Helen Ross, Timothy Lynam, Pascal Perez, and Anne Leitch. Mental models: An interdisciplinary synthesis of theory and methods. *Ecology and Society*, 16(1), Mar 2011.
- Wendy Ju. The design of implicit interactions. *Synthesis Lectures on Human-Centered Informatics*, 8(2):1–93, 2015.
- Neel Kant. Recent advances in neural program synthesis. *CoRR*, abs/1802.02353, 2018. URL <http://arxiv.org/abs/1802.02353>.
- Joohyun Kim and Raymond Mooney. Adapting discriminative reranking to grounded language learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 218–227, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P13-1022>.

- Joohyun Kim and Raymond J. Mooney. Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 433–444, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL <http://dl.acm.org/citation.cfm?id=2390948.2391001>.
- Andrew J Ko, Brad A Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pages 199–206. IEEE, 2004.
- Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, 43(3):21, 2011.
- Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*, HRI '10, pages 259–266, Piscataway, NJ, USA, 2010. IEEE Press. ISBN 978-1-4244-4893-7. URL <http://dl.acm.org/citation.cfm?id=1734454.1734553>.
- Thomas Kollar, Danielle Berry, Lauren Stuart, Karolina Owczarzak, Tagyoung Chung, Lambert Mathias, Michael Kayser, Bradford Snow, and Spyros Matsoukas. The alexa meaning representation language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 177–184, 2018.
- Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. Frame-based editing: Easing the transition from blocks to text-based programming. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE 15, page 2938, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450337533. doi: 10.1145/2818314.2818331. URL <https://doi.org/10.1145/2818314.2818331>.
- Ralf Krestel, Sabine Bergler, and René Witte. Minding the source: Automatic tagging of reported speech in newspaper articles. In *LREC 2008*, 2008. URL http://www.lrec-conf.org/proceedings/lrec2008/pdf/718_paper.pdf.
- Shriram Krishnamurthi and Kathi Fisler. Programming paradigms and beyond. *The Cambridge Handbook of Computing Education Research*, 2019.

- Marek Kubis, Paweł Skórzewski, and Tomasz Zikerkiewicz. EUDAMU at SemEval-2017 task 11: Action ranking and type matching for end-user development. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1000–1004, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/S17-2175>.
- Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th International Conference on Intelligent User Interfaces, IUI '15*, pages 126–137, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3306-1. doi: 10.1145/2678025.2701399. URL <http://doi.acm.org/10.1145/2678025.2701399>.
- Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics, 2010.
- Thomas D. LaToza, W. Ben Towne, Christian M. Adriano, and André van der Hoek. Microtask programming: Building software with a crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST 14*, page 4354, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330695. doi: 10.1145/2642918.2647349. URL <https://doi.org/10.1145/2642918.2647349>.
- Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL <http://jmlr.org/papers/v18/16-365>.
- Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. Coscripiter: Automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08*, pages 1719–1728, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: 10.1145/1357054.1357323. URL <http://doi.acm.org/10.1145/1357054.1357323>.
- Deyi Li and Yi Du. *Artificial intelligence with uncertainty*. CRC press, 2017.
- Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

- Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 639–646, 2010.
- Henry Lieberman. *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Elsevier, 2001.
- Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. *End-User Development: An Emerging Paradigm*, pages 1–8. Springer Netherlands, Dordrecht, 2006. ISBN 978-1-4020-5386-3. doi: 10.1007/1-4020-5386-X_1. URL https://doi.org/10.1007/1-4020-5386-X_1.
- R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.
- JongHyun Lim and Kyong-Ho Lee. Constructing composite web services from natural language requests. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):1 – 13, 2010. ISSN 1570-8268. doi: <https://doi.org/10.1016/j.websem.2009.09.007>. URL <http://www.sciencedirect.com/science/article/pii/S1570826809000493>.
- Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL <https://www.aclweb.org/anthology/L18-1491>.
- Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1057. URL <https://www.aclweb.org/anthology/P16-1057>.
- Zachary C. Lipton. The mythos of model interpretability. In *Proceedings of the ICML 2016 Workshop on Human Interpretability in Machine Learning*, 2016.
- Greg Little and Robert C. Miller. Translating keyword commands into executable code. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 135–144, New York, NY, USA, 2006. ACM. ISBN 1-59593-313-1. doi: 10.1145/1166253.1166275. URL <http://doi.acm.org/10.1145/1166253.1166275>.

- Chang Liu, Xinyun Chen, Richard Shin, Mingcheng Chen, and Dawn Song. Latent attention for if-then program synthesis. In *Advances in Neural Information Processing Systems*, pages 4574–4582, 2016.
- Hugo Liu and Henry Lieberman. Programmatic semantics for natural language interfaces. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1597–1600, New York, NY, USA, 2005a. ACM. ISBN 1-59593-002-7. doi: 10.1145/1056808.1056975. URL <http://doi.acm.org/10.1145/1056808.1056975>.
- Hugo Liu and Henry Lieberman. Metafor: Visualizing stories as code. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, pages 305–307, New York, NY, USA, 2005b. ACM. ISBN 1-58113-894-6. doi: 10.1145/1040830.1040908. URL <http://doi.acm.org/10.1145/1040830.1040908>.
- Gustavo López, Luis Quesada, and Luis A Guerrero. Alexa vs. siri vs. cortana vs. google assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, 2017.
- Robert Elton Maas and Patrick Suppes. Natural-language interface for an instructable robot. *International Journal of Man-Machine Studies*, 22(2):215 – 240, 1985. ISSN 0020-7373. doi: [http://dx.doi.org/10.1016/S0020-7373\(85\)80071-7](http://dx.doi.org/10.1016/S0020-7373(85)80071-7). URL <http://www.sciencedirect.com/science/article/pii/S0020737385800717>.
- Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1475–1482. AAAI Press, 2006. ISBN 978-1-57735-281-5. URL <http://dl.acm.org/citation.cfm?id=1597348.1597423>.
- Bill Z. Manaris and Wayne D. Dominick. Nalige: a user interface management system for the development of natural language interfaces. *International Journal of Man-Machine Studies*, 38(6):891 – 921, 1993. ISSN 0020-7373. doi: <http://dx.doi.org/10.1006/imms.1993.1042>. URL <http://www.sciencedirect.com/science/article/pii/S0020737383710424>.
- Bill Z. Manaris, Jason W. Pritchard, and Wayne D. Dominick. Developing a natural language interface for the unix operating system. *SIGCHI Bull.*, 26(2):34–40, April 1994. ISSN 0736-6906. doi: 10.1145/198125.198137. URL <http://doi.acm.org/10.1145/198125.198137>.
- Christopher D. Manning. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of the 12th International*

- Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I, CICLing'11*, pages 171–189, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-19399-6. URL <http://dl.acm.org/citation.cfm?id=1964799.1964816>.
- Mehdi Manshadi, Carolyn Keenan, and James F. Allen. Using the crowd to do natural language programming. In *Human Computation at AAAI Technical Report WS-12-08*, 2012.
- Mehdi H Manshadi, Daniel Gildea, and James F Allen. Integrating programming by example and natural language programming. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.
- Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 251–258. IEEE, 2010.
- Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.
- Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.
- Hongyuan Mei, Mohit Bansal, and Matthew R Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. URL <https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12522/12021>.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- Franco Milicchio, Rebecca Rose, Jiang Bian, Jae Min, and Mattia Proserpi. Visual programming for next-generation sequencing data analytics. *BioData mining*, 9:16–16, Apr 2016. ISSN 1756-0381. doi: 10.1186/s13040-016-0095-3. URL <https://www.ncbi.nlm.nih.gov/pubmed/27127540>.
- George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995. ISSN 0001-0782.
- Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *Proceedings of the 2018 Conference*

- on *Empirical Methods in Natural Language Processing*, pages 2667–2678, 2018.
- Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016. doi: 10.1177/0278364915602060. URL <https://doi.org/10.1177/0278364915602060>.
- Raymond J. Mooney. Comparative experiments on disambiguation word senses: An illustration of the role of bias in machine learning. In *Proc. Conference on Empirical Methods in Natural Language Processing*, pages 82–91, 1996.
- James H Muller. The great logo adventure. *Computers in the Schools*, 2(2-3):117–125, 1985.
- Brad A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97 – 123, 1990. ISSN 1045-926X. doi: [https://doi.org/10.1016/S1045-926X\(05\)80036-9](https://doi.org/10.1016/S1045-926X(05)80036-9). URL <http://www.sciencedirect.com/science/article/pii/S1045926X05800369>.
- David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, January 2007. Publisher: John Benjamins Publishing Company.
- Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. Service composition for non-programmers: Prospects, problems, and design recommendations. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 123–130. IEEE, 2010.
- Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015. URL <http://arxiv.org/abs/1511.04834>.
- Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584, Nov 2015. doi: 10.1109/ASE.2015.36.
- S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010. ISSN 2326-3865. doi: 10.1109/TKDE.2009.191.

- John F. Pane and Brad A. Myers. *More Natural Programming Languages and Environments*, pages 31–50. Springer Netherlands, Dordrecht, 2006. ISBN 978-1-4020-5386-3. doi: 10.1007/1-4020-5386-X_3. URL https://doi.org/10.1007/1-4020-5386-X_3.
- Parliament and Council of European Union. General Data Protection Regulation (EU) no 2016/679, 2016.
- Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. Mapping natural language commands to web elements, 2018.
- Fabio Paternò and Volker Wulf, editors. *New Perspectives in End-User Development*. Springer, 2017.
- Michael J. Pazzani. Representation of electronic mail filtering profiles: A user study. In *Proceedings of the 5th International Conference on Intelligent User Interfaces, IUI '00*, pages 202–206, New York, NY, USA, 2000. ACM. ISBN 1-58113-134-8. doi: 10.1145/325737.325843. URL <http://doi.acm.org/10.1145/325737.325843>.
- Roy D Pea. Logo programming and problem solving. Technical Report 12, Center for Children and Technology at the Bank Street College of Education, 1987.
- Hila Peleg, Sharon Shoham, and Eran Yahav. Programming not only by example. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1114–1124. IEEE, 2018.
- Vittorio Perera, Tagyoung Chung, Thomas Kollar, and Emma Strubell. Multi-task learning for parsing the alexa meaning representation language. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- Dragutin Petkovic, Russ Altman, Mike Wong, and Arthur Vigil. Improving the explainability of random forest classifier - user centered approach. *Pac Symp Biocomput*, 23:204–215, 2018. ISSN 2335-6936. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5728671/>. 29218882[pmid].
- Leo L Pipino, Yang W Lee, and Richard Y Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.
- Florin-Claudiu Pop, Marcel Cremene, Mircea Vaida, and Michel Riveill. Natural language service composition with request disambiguation. In Paul P. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing*, pages 670–677, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-17358-5.

- Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI '03*, pages 149–157, New York, NY, USA, 2003. ACM. ISBN 1-58113-586-6. doi: 10.1145/604045.604070. URL <http://doi.acm.org/10.1145/604045.604070>.
- Shaya Pourmirza, Sander Peters, Remco Dijkman, and Paul Grefen. Bpms-ra: A novel reference architecture for business process management systems. *ACM Trans. Internet Technol.*, 19(1):13:1–13:23, February 2019. ISSN 1533-5399. doi: 10.1145/3232677. URL <http://doi.acm.org/10.1145/3232677>.
- David Price, Ellen Riloff, Joseph Zachary, and Brandon Harvey. Naturaljava: a natural language interface for programming in java. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 207–211. ACM, 2000.
- P. J. Price. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, pages 91–95, 1990. URL <http://www.aclweb.org/anthology/H90-1020>.
- Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, Beijing, China, July 2015. URL <http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127514>.
- Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1105. URL <https://www.aclweb.org/anthology/P17-1105>.
- Partha Pratim Ray. A survey on visual programming languages in internet of things. *Scientific Programming*, 2017, 2017.
- Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, May 2016.
- Radim Rehurek. Performance shootout of nearest neighbours: Querying, 2014. URL <https://rare-technologies.com/performance-shootout-of-nearest-neighbours-querying/>.
- Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

Raymond Reiter. On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier, 1981.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pages 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL <http://doi.acm.org/10.1145/2939672.2939778>.

Roberto Rodri-gues Filho, Alexander Wild, and Barry Porter. Code synthesis in self-improving software systems. In *Proceedings of the International Workshop of Self-Improving System Integration*. IEEE, 2019.

Peter Gordon Roetzal. Information overload in the information age: a review of the literature from business administration, business psychology, and related disciplines with a bibliometric approach and framework development. *Business Research*, Jul 2018. ISSN 2198-2627. doi: 10.1007/s40685-018-0069-z. URL <https://doi.org/10.1007/s40685-018-0069-z>.

David J. Rogers and Taffee T. Tanimoto. A computer program for classifying plants. *Science*, 132(3434):1115–1118, 1960. ISSN 0036-8075. doi: 10.1126/science.132.3434.1115. URL <https://science.sciencemag.org/content/132/3434/1115>.

Senjuti Basu Roy, Martine De Cock, Vani Mandava, Swapna Savanna, Brian Dalessandro, Claudia Perlich, William Cukierski, and Ben Hamner. The microsoft academic search dataset and kdd cup 2013. In *Proceedings of the Workshop for KDD Cup 2013*. ACM - Association for Computing Machinery, August 2013. URL <https://www.microsoft.com/en-us/research/publication/the-microsoft-academic-search-dataset-and-kdd-cup-2013-workshop-for-kdd-cup->

Juliano Efon Sales, Siegfried Handschuh, and André Freitas. Semeval-2017 task 11: End-user development using natural language. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 556–564, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/S17-2092>.

Juliano Efon Sales, André Freitas, and Siegfried Handschuh. An open vocabulary semantic parser for end-user programming using natural language. In *12th IEEE International Conference on Semantic Computing, ICSC 2018, Laguna Hills, CA, USA, January 31 - February 2, 2018*, pages 77–84, 2018a. doi: 10.1109/ICSC.2018.00020. URL <https://doi.org/10.1109/ICSC.2018.00020>.

- Juliano Efsou Sales, Leonardo Souza, Siamak Barzegar, Brian Davis, André Freitas, and Siegfried Handschuh. Indra: A word embedding and semantic relatedness server. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018b. European Language Resources Association (ELRA).
- Jean E. Sammet. The use of english as a programming language. *Commun. ACM*, 9(3):228–230, March 1966. ISSN 0001-0782. doi: 10.1145/365230.365274. URL <http://doi.acm.org/10.1145/365230.365274>.
- Jordy Sangers, Flavius Frasincar, Frederik Hogenboom, and Vadim Chepegin. Semantic web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11):4660 – 4671, 2013. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2013.02.011>. URL <http://www.sciencedirect.com/science/article/pii/S0957417413001279>.
- Mark Santolucito, Drew Goldman, Allyson Weseley, and Ruzica Piskac. Programming by example: Efficient, but not “helpful”. In *9th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2018)*, 2018.
- Viktor Schlegel, Benedikt Lang, Siegfried Handschuh, and André Freitas. Vajra: step-by-step programming with natural language. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 30–39. ACM, 2019.
- Karin Kipper Schuler. *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2005. AAI3179808.
- Robert W. Sebesta. *Concepts of Programming Languages*. Pearson, 10th edition, 2012. ISBN 0273769103, 9780273769101.
- J. Segal. Some problems of professional end user developers. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, pages 111–118, Sep. 2007. doi: 10.1109/VLHCC.2007.17.
- Andrew D Selbst and Julia Powles. Meaningful information and the right to explanation. *International Data Privacy Law*, 7(4):233–242, 2017.
- Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL <http://arxiv.org/abs/1610.02391>.

- Richard Shin, Neel Kant, Kavi Gupta, Chris Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. Synthetic datasets for neural program synthesis. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=rye0SnAqYm>.
- Chengxun Shu and Hongyu Zhang. Neural programming by example. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- Vivian Silva, Siegfried Handschuh, and Andr Freitas. Recognizing and justifying text entailment through distributional navigation on definition graphs. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. URL <https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16246>.
- Vivian Silva, Andre Freitas, and Siegfried Handschuh. Exploring knowledge graphs in an interpretable composite approach for text entailment. In *Thirty-Third AAAI conference on artificial intelligence*. AAAI Press, 2019.
- Pavel Smutný. Visual programming for smartphones. In *2011 12th International Carpathian Control Conference (ICCC)*, pages 358–361. IEEE, 2011.
- Juha Sorva. Notional machines and introductory programming education. *Trans. Comput. Educ.*, 13(2):8:1–8:31, July 2013. ISSN 1946-6226. doi: 10.1145/2483710.2483713. URL <http://doi.acm.org/10.1145/2483710.2483713>.
- Mark Steedman. *Surface Structure and Interpretation*. The MIT Press, 1996.
- Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0-262-19420-1.
- Simone Stumpf, Vidya Rajaram, Lida Li, Margaret Burnett, Thomas Dietterich, Erin Sullivan, Russell Drummond, and Jonathan Herlocker. Toward harnessing user feedback for machine learning. In *Proceedings of the 12th International Conference on Intelligent User Interfaces, IUI '07*, pages 82–91, New York, NY, USA, 2007. ACM. ISBN 1-59593-481-2. doi: 10.1145/1216295.1216316. URL <http://doi.acm.org/10.1145/1216295.1216316>.
- Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. Building natural language interfaces to web apis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM '17*, pages 177–186, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3133009. URL <http://doi.acm.org/10.1145/3132847.3133009>.

- Ron Sun. Artificial intelligence: Connectionist and symbolic approaches. *International Encyclopedia of the Social & Behavioral Sciences*, 04 2000. doi: 10.1016/B0-08-043076-7/00553-2.
- Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press. URL <http://dl.acm.org/citation.cfm?id=2969033.2969173>.
- Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, pages 466–477, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44795-5.
- Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth J. Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011. URL <http://dblp.uni-trier.de/db/conf/aaai/aaai2011.html#TellexKDWBTR11>.
- The App Association. Six-Figure Tech Salaries. Technical report, The App Association, 2018.
- Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1923–1929. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL <http://dl.acm.org/citation.cfm?id=2832415.2832516>.
- Henry S Thompson, Anne Anderson, Ellen Gurman Bard, Gwyneth Doherty-Sneddon, Alison Newlands, and Cathy Sotillo. The hrc map task corpus: natural dialogue for speech recognition. In *Proceedings of the workshop on Human Language Technology*, pages 25–30. Association for Computational Linguistics, 1993.
- Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January 2010. ISSN 1076-9757.
- Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Menicken, Noah Picard, Diane Schulze, and Michael L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3227–3231, New York, NY, USA, 2016.

- ACM. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858556. URL <http://doi.acm.org/10.1145/2858036.2858556>.
- Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>.
- Jo Vermeulen. *Designing for intelligibility and control in ubiquitous computing environments*. PhD thesis, 2014.
- Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 806–814. Association for Computational Linguistics, 2010.
- E. S. Vorm. Assessing demand for transparency in intelligent systems using machine learning. In *2018 Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–7, July 2018. doi: 10.1109/INISTA.2018.8466328.
- Matthew R. Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth J. Teller. Learning semantic maps from natural language descriptions. In *Robotics: Science and Systems*, 2013.
- Matthew R. Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. A framework for learning semantic maps from grounded natural language descriptions. *International Journal of Robotics Research*, 31(9):1167–1190, August 2014.
- Daniel C Wang. The zephyr abstract syntax description language. In *Proceedings of Conference on Domain-Specific Languages, 1997*. USENIX, 1997.
- Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, 2015.

- Daniel M. Wegner and David J. Schneider. The white bear story. *Psychological Inquiry*, 14(3/4):326–329, 2003. ISSN 1047840X, 15327965. URL <http://www.jstor.org/stable/1449696>.
- S. Weigelt, T. Hey, and V. Steurer. Detection of conditionals in spoken utterances. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 85–92, Jan 2018. doi: 10.1109/ICSC.2018.00021.
- Soren Havelund Welling, Hanne Hoffmann Froelund Refsgaard, Per Bruun Brockhoff, and Line Katrine Harder Clemmensen. Forest floor visualizations of random forests. *ArXiv e-prints*, may 2016. URL <http://arxiv.org/abs/1605.09196>.
- George Westerman and Didier Bonnet. Revamping your business through digital transformation. *MIT Sloan Management Review*, 56(3):1–6, 2015.
- Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 404–413, Metz, France, August 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/W13-4065>.
- Jeffrey Wong. Marmite: Towards end-user programming for the web. In *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pages 270–271. IEEE, 2007.
- Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/P07-1121>.
- Yuk Wah Wong and Raymond J. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL '06*, pages 439–446, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. doi: 10.3115/1220835.1220891. URL <https://doi.org/10.3115/1220835.1220891>.
- Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damjanovic, Brian Davis, Norbert Fuchs, Stefan Hoefler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, Martin Luts, Jonathan Pool, Mike Rosner, Rolf Schwitler, and John Sowa. On controlled natural languages: Properties and prospects. In Norbert E. Fuchs, editor, *Controlled Natural Language*, pages 281–289, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14418-9.

- Chao Xie, Hua Qi, Lei Ma, and Jianjun Zhao. Deepvisual: A visual programming tool for deep learning systems. In *Proceedings of the 27th International Conference on Program Comprehension, ICPC 19*, page 130134. IEEE Press, 2019. doi: 10.1109/ICPC.2019.00028. URL <https://doi.org/10.1109/ICPC.2019.00028>.
- Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, 2017.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.
- Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S Lasecki, and Dragomir Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases, 2019a.
- Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. Sparc: Cross-domain semantic parsing in context. *arXiv preprint arXiv:1906.02285*, 2019b.
- John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2, AAAI'96*, pages 1050–1055. AAAI Press, 1996. ISBN 0-262-51091-X. URL <http://dl.acm.org/citation.cfm?id=1864519.1864543>.
- John M Zelle, Raymond J Mooney, and Joshua B Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In *Machine Learning Proceedings 1994*, pages 343–351. Elsevier, 1994.
- Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence, UAI'05*, pages 658–666, Arlington, Virginia, United States, 2005. AUAI Press. ISBN 0-9749039-1-4. URL <http://dl.acm.org/citation.cfm?id=3020336.3020416>.

- Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.
- J. Zhou, S. Z. Arshad, X. Wang, Z. Li, D. Feng, and F. Chen. End-user development for interactive data analytics: Uncertainty, correlation and user confidence. *IEEE Transactions on Affective Computing*, 9(3):383–395, July 2018. ISSN 1949-3045. doi: 10.1109/TAFFC.2017.2723402.
- Jianlong Zhou, M. Asif Khawaja, Zhidong Li, Jinjun Sun, Yang Wang, and Fang Chen. Making machine learning useable by revealing internal states update - a transparent approach. *Int. J. Comput. Sci. Eng.*, 13(4):378–389, January 2016. ISSN 1742-7185. doi: 10.1504/IJCSE.2016.080214. URL <https://doi.org/10.1504/IJCSE.2016.080214>.