University of California
Santa Barbara

Towards Democratizing Data Science with Natural Language Interfaces

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Yu Su

Committee in charge:

Professor Xifeng Yan, Chair
Professor Ambuj Singh
Professor Amr El Abbadi

December 2018

The Dissertation of Yu Su is approved.

Professor Ambuj Singh

Professor Amr El Abbadi

Professor Xifeng Yan, Committee Chair

May 2018

Towards Democratizing Data Science with Natural Language Interfaces

by

Yu Su

To Huan,
For the company in the dark, at the dawn, and ever after.

Acknowledgements

I would like to thank my advisor, Xifeng Yan, without whom nothing in this dissertation is possible. He did not give up on me in the early days when nothing seemed going in the right direction. He guided me into the fantastic field of machine learning and natural language processing. His advice, encouragement, and criticism over my whole PhD life have largely shaped the way I do research and beyond, and the things I have learned from him will continue to be invaluable in my future academic career.

I am also in great debt to my committee members, Ambuj Singh and Amr El Abbadi. They provided me invaluable feedback at every stage of my graduate study. Although not officially on my PhD committee, I would also like to give special acknowledgements to Brian Sadler, whose advice over the years has greatly helped me shape my research path and become a better researcher.

I would like to thank my mentors and collaborators: Mudhakar Srivatsa, Sue Kase, Michelle Vanni, Wen-Tau Yih, Ahmed Hassan Awadallah, Patrick Pantel, Ryen White, Michael Gamon, Mark Encarnacion, Madian Khabsa, Miaosen Wang, William Yang Wang, Shengqi Yang, Fangqiu Han, Semih Yavuz, Izzeddin Gur, Keqian Li, Hanwen Zha, Wenhu Chen, and Xin Wang.

I would like to express my deepest gratitude to my wife, Huan Sun, for her company and encouragement. I also thank my parents, my sister and my brother for their love and support.

The research in this dissertation is funded in part by the Army Research Laboratory under cooperative agreements W911NF09-2-0053, NSF IIS 0954125, and NSF IIS 1528175, and Microsoft Research.

Curriculum Vitæ

Yu Su

Education

2012 - 2018	Ph.D. in Computer Science, University of California, Santa Barbara.
2008 - 2012	B.E. in Computer Science and Technology, Tsinghua University.

Experience

10/2018 -	Researcher, Microsoft Semantic Machines
08/2018 -	Visiting Assistant Professor, Dept. of CSE@The Ohio State University
06/2017 - 09/2017	Research Intern, Microsoft Research, Redmond
06/2016 - 09/2016	Research Intern, Microsoft Research, Redmond
09/2015 - 11/2015	Visiting Researcher, U.S. Army Research Laboratory
06/2015 - 09/2015	Research Intern, IBM T.J. Watson Research Center
06/2013 - 06/2018	Research Assistant, University of California, Santa Barbara
09/2012 - 06/2013	Teaching Assistant, University of California, Santa Barbara
07/2011 - 09/2011	Undergraduate Research Assistant, University of Notre Dame

Selected Publications

AAAI 2019	Xin Wang, Jiawei Wu, Da Zhang, Yu Su, William Yang Wang. Learning to Compose Topic-Aware Mixture of Experts for Zero-Shot Video Captioning.
ICDM 2018	Keqian Li, Hanwen Zha, Yu Su, Xifeng Yan. Concept Mining via Embedding.
EMNLP 2018	Wenhu Chen, Jianshu Chen, Yu Su, Xin Wang, Dong Yu, Xifeng Yan and William Yang Wang. XL-NBT: A Cross-lingual Neural Belief Tracking Framework.
EMNLP 2018	Semih Yavuz, Izzeddin Gur, Yu Su and Xifeng Yan. What It Takes to Achieve 100% Condition Accuracy on WikiSQL.
ACL 2018	Izzeddin Gur, Semih Yavuz, Yu Su, Xifeng Yan. DialSQL: Dialogue Based Structured Query Generation.
SIGIR 2018	Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, Ryen White. Natural Language Interfaces with Fine-Grained User Interaction: A Case Study on Web APIs.
NAACL 2018	Yu Su*, Honglei Liu*, Semih Yavuz, Izzeddin Gur, Huan Sun, Xifeng Yan. Global Relation Embedding for Relation Extraction. (*: equal contribution)

SDM 2018	Keqian Li, Hanwen Zha, Yu Su, Xifeng Yan. Unsupervised Neural Categorization for Scientific Publications.
CIKM 2017	Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, Mark Encarnacion. Building Natural Language Interfaces to Web APIs.
EMNLP 2017	Yu Su, Xifeng Yan. Cross-domain Semantic Parsing via Paraphrasing.
EMNLP 2017	Jie Zhao, Yu Su, Ziyu Guan, Huan Sun. An End-to-End Deep Framework for Answer Triggering with a Novel Group-Level Objective.
EMNLP 2017	Semih Yavuz, Izzeddin Gur, Yu Su, Xifeng Yan. Recovering Question Answering Errors via Query Revision.
EMNLP 2016	Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, Xifeng Yan. On Generating Characteristic-rich Question Sets for QA Evaluation.
EMNLP 2016	Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, Xifeng Yan. Improving Semantic Parsing via Answer Type Inference.
SDM 2016	Yu Su, Fangqiu Han, Richard E. Harang, Xifeng Yan. A Fast Kernel for Attributed Graphs.
WWW 2016	Huan Sun, Hao Ma, Xiaodong He, Wen-Tau Yih, Yu Su, Xifeng Yan. Table Cell Search for Question Answering.
KDD 2015	Yu Su, Shengqi Yang, Huan Sun, Mudhakar Srivatsa, Sue Kase, Michelle Vanni, Xifeng Yan. Exploiting Relevance Feedback in Knowledge Graph Search.

Awards and Honors

2018	Best Distinguished Graduate Student Lecture, UCSB
2016	SDM Travel Award, SIAM
2012	LINK Lab Fellowship, UCSB
2012	Outstanding Graduate Award, Tsinghua University
2009 - 2011	Metrit-based Fellowships, Tsinghua University
2008	Outstanding Freshman Fellowship, Tsinghua University

Abstract

Towards Democratizing Data Science with Natural Language Interfaces

by

Yu Su

Data science has the potential to reshape many sectors of the modern society. This potential can be realized to its maximum only when data science becomes democratized, instead of centralized in a small group of expert data scientists. However, with data becoming more massive and heterogeneous, standing in stark contrast to the spreading demand of data science is the growing gap between human users and data: Every type of data requires extensive specialized training, either to learn a specific query language or a data analytics software. Towards the democratization of data science, in this dissertation we systematically investigate a promising research direction, natural language interface, to bridge the gap between users and data, and make it easier for users who are less technically proficient to access the data analytics power needed for on-demand problem solving and decision making.

One of the largest obstacles for general users to access data is the proficiency requirement on formal languages (e.g., SQL) that machines use. Automatically parsing natural language commands from users into formal languages, natural language interfaces can thus play a critical role in democratizing data science. However, a pressing question that is largely left unanswered so far is, how to bootstrap a natural language interface for a new domain? The high cost of data collection and the data-hungry nature of the mainstream neural network models are significantly limiting the wide application of natural language interfaces.

The main technical contribution of this dissertation is a systematic framework for bootstrapping natural language interfaces for new domains. Specifically, the proposed framework consists of three complimentary methods: (1) Collecting data at a low cost via crowdsourcing,

(2) leveraging existing NLI data from other domains via transfer learning, and (3) letting a bootstrapped model to interact with real users so that it can refine itself over time. Combining the three methods forms a closed data loop for bootstrapping and refining natural language interfaces for any domain.

The developed methodologies and frameworks in this dissertation hence pave the path for building data science platforms that everyone can use to process, query, and analyze their data without extensive specialized training. With such AI-powered platforms, users can stay focused on high-level thinking and decision making, instead of overwhelmed by low-level implementation and programming details — "*Let machines understand human thinking. Don't let humans think like machines*."

Contents

Chapter 1

Introduction

The past decades have been witnessing the revolution of global digitalization, and subsequently, the meteoric growth of digital data. Take the health domain as an example. The coverage of electronic health record has reached 86.9% as of 2015 [1], the cost of whole genome sequencing has remarkably decreased to less than 1,000 dollars [2], and mobile health monitoring via devices like Fitbit is generating a huge amount of multi-dimentional personal health data. Such data explosion is happening in almost every sector of the modern society, and has a great potential to enable answering many questions that can not be possibly answered before. For example, a doctor may quickly find similar patients under different criteria based on electronic medical records. With the growth of data comes the great need of capabilities to get insights from data, which has led to the new field of data science.

Data science has the potential to reshape many sectors of the modern society, but this potential can be realized to its maximum only when data science becomes democratized, instead of centralized in a small group of expert data scientists. However, with data becoming more massive and heterogeneous, standing in stark contrast to the spreading demand is the growing gap between end users and data: Every type of data requires extensive specialized training, either to learn a specific query language or a data analytics software. How can we *democratize*

data science, i.e., bridge the gap between users and data, and make it easier for users who are less technically proficient to access the data analytics power needed for on-demand decision making? This is the main theme of this dissertation.

One main obstacle for general users to access data and data analytics power is the gap between the natural language used for communication between humans and the formal languages used by machines. On-demand decision making requires the agility to interact with data in a process of dynamically generating questions (hypotheses), as and when needed, and quickly getting answers. However, non-expert users can hardly enjoy such agility. Learning formal languages like SQL (Structured Query Language) takes a large amount of time. On the other hand, whereas canned forms (e.g., pre-defined templates on graphical user interfaces (GUIs)) shield users from programming, they lack the necessary expressive flexibility for ad-hoc data querying and analytics. The gap between user and data is growing further with the rapidly increasing data volume and heterogeneity. For example, modern knowledge bases like Google Knowledge Graph contains thousands of entity and relation types, tens of millions of entities, and hundreds of billions of relational facts about entities; writing a formal query for such complex data is a challenge even for programming-proficient users, let alone users who are novice in programming. It may also not be economical to write ad-hoc, one-off formal queries to meet every dynamically generated information need.

In this dissertation, we study *natural language interfaces* (NLIs), a type of intelligent systems that can understand natural language requests from users and automatically transduce into the corresponding formal languages that machines can understand. NLIs serve as a unified and easy-to-use interface between human users and machines, and are not constrained to a pre-defined set of functionalities like GUIs. With NLIs, users can simply express their requests, either querying some data or calling some data analytics services, in natural language and directly get the results back, which significantly lowers the bar for data science.

Research on natural language interfaces has spanned several decades. Early rule-based

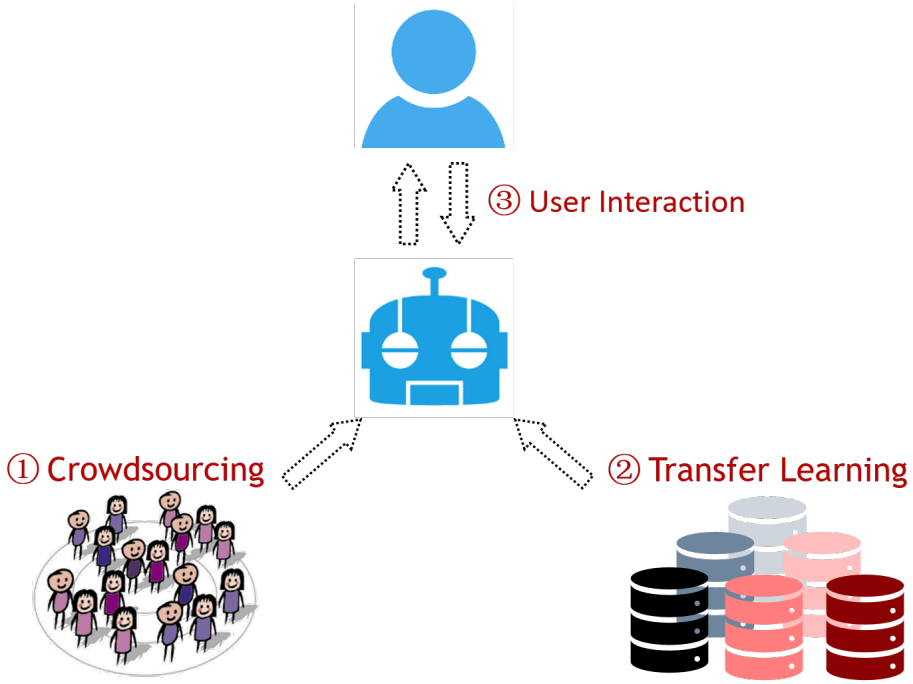


Figure 1.1: A systematic framework for bootstrapping natural language interfaces.

NLIs [3, 4, 5] employ carefully designed rules to map natural language questions to formal meaning representations like SQL queries. While having a high precision, rule-based systems are brittle when facing with language variations, and usually only admit inputs from a restricted vocabulary. The rise of statistical models [6, 7, 8], especially the ongoing wave of neural network models [9, 10], has enabled NLIs to be more robust to language variations. Such systems, under the re-branded name of semantic parsing or question answering, allow users to formulate commands with greater flexibility and have shown a wide success. Throughout this dissertation we will use natural language interface, semantic parsing, and question answering interchangeably.

However, most of existing studies on natural language interfaces have focused on better modeling with the goal to improve the parsing accuracy, and usually assume the data needed to train the parsing model is readily available. This is the largest challenge faced by practitioners of NLI technology, and is significantly limiting its wide application. For example, if

someone wants to build an NLI to electronic medical records to support ad-hoc search requests from doctors, how can she get the annotated data needed to train an NLI model? While data collection and annotation is a widely-faced challenge in machine learning, it is particularly challenging for NLI because it requires annotators to be familiar with formal languages. This becomes even more challenging for the mainstream neural network models because they are much more data-hungry than traditional machine learning models.

The main technical contribution of this dissertation is to propose a systematic framework for NLI bootstrapping in new domains (Figure 1.1). Specifically, the proposed framework consists of three complimentary methods: (1) Collecting data at a low cost via crowdsourcing, (2) leveraging existing NLI data from other domains via transfer learning, and (3) letting a bootstrapped model to interact with real users so that it can refine itself over time. Next we introduce each step in more details.

1.1 NLI Data Collection via Crowdsourcing

Crowdsourcing is a scalable way for data collection for machine learning models. However, it is challenging to collect data for NLI, which consists of natural language utterance and logical form pairs, via crowdsourcing because most crowd workers do not understand logical forms/formal languages. In Chapter 2 we present a semi-automated framework to enable NLI data collection via crowdsourcing. Specifically, we first automatically generate logical forms, and have domain experts to convert these logical forms into *canonical utterances* in natural language, which are a little clumsy but understandable by crowd workers. The job of crowd workers is then just to paraphrase the canonical utterances into a more natural way. We also propose novel techniques to improve the data quality from this framework. We apply to framework to generate a large dataset for knowledge-based question answering (KBQA), which has been widely used to train and evaluate KBQA systems.

1.2 Cross-domain NLIs

Considering that there already exists many domains with NLI training data, a natural idea for bootstrapping an NLI for a new domain to leverge the existing data to learn for the new domain, a typical transfer learning setting. However, this is challenging for NLI because different domains involve different entities and predicates, and can even be defined by different formal languages. In Chapter 3 we present the first study on cross-domain NLI, and we propose a novel method based on paraphrasing to make it possible to do transfer learning for NLI.

1.3 Interactive NLIs

So far we have introduced how to boostrap an NLI via crowdsourcing and transfer learning. An NLI bootstrapped this way may deliver reasonable performance, but it can still be limited in various ways because the data distribution it has seen so far is inevitably different from the true data distribution it will get from real users. Therefore, it is critical for enabling NLIs to interact with real users so that it can continue improving itself over time. In Chapter 4 we discuss innovative interaction mechanisms for NLIs.

In summary, in this dissertation we present a systematic framework to bootstrap NLIs for new domains, which is of both theoretical and practical interests. Along the way we have also successfully developed NLIs for a wide range of data types such as relational databases [11, 12], knowledge bases [13, 14, 15, 16], semi-structured web tables [17], and web APIs [18, 19]. It paves the way for wide application of NLI technologies.

Chapter 2

NLI Data Collection via Crowdsourcing

2.1 Introduction

Factoid question answering (QA) has gained great attention recently, owing to the fast growth of large knowledge bases (KBs) such as DBpedia [20] and Freebase [21], which avail QA systems of comprehensive and precise knowledge of encyclopedic scope [22, 8, 23, 24, 25, 26, 27, 28, 29, 30, 9, 31, 32, 33, 34]. With the blossoming of QA systems, evaluation is becoming an increasingly important problem. QA datasets, consisting of a set of questions with ground-truth answers, are critical for both comparing existing systems and gaining insights to develop new systems.

Questions have rich *characteristics*, constituting dimensions along which question difficulty varies. Some questions are difficult due to their complex semantic structure (*"Who was the coach when Michael Jordan stopped playing for the Chicago Bulls?"*), while some others may be difficult because they require a precise quantitative analysis over the answer space (*"What is the best-selling smartphone in 2015?"*). Many other characteristics shall be considered too, e.g., what topic a question is about (questions about common topics may be easier to answer) and how many answers there are (it is harder to achieve a high recall in case of

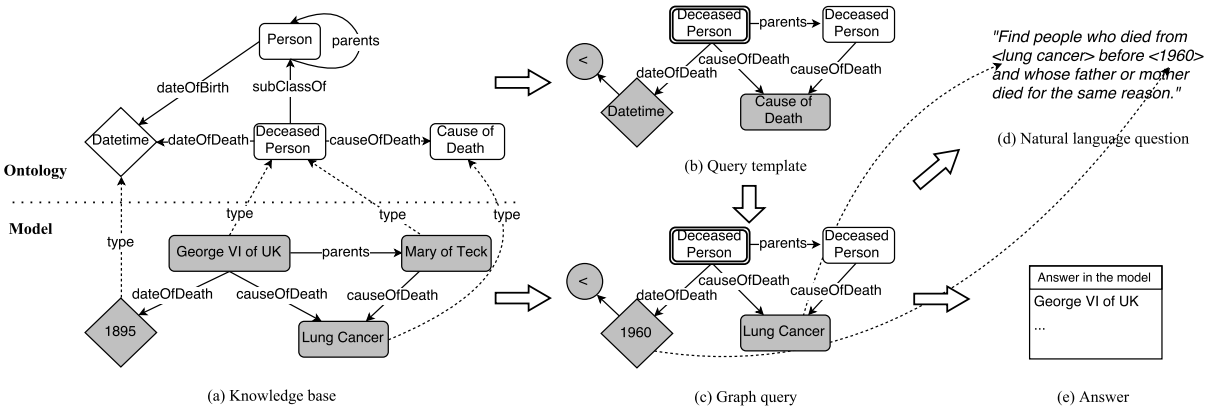


Figure 2.1: Running example of our framework. Graph queries are first generated from a knowledge base. After refinement (not shown), graph queries are sent to human annotators and converted into natural language questions. Answers are collected from the knowledge base.

multiple answers). Worse still, due to the flexibility of natural language, different people often describe the same question in different ways, i.e., paraphrasing. It is important for a QA system to be robust to paraphrasing.

A QA dataset explicitly specifying such question characteristics allows for fine-grained inspection of system performance. However, to the best of our knowledge, none of the existing QA datasets [35, 8, 23, 36, 37, 38] provides question characteristics. In this work, we make the first attempt to generate questions with explicitly specified characteristics, and examine the impact of various question characteristics in QA.

We present a semi-automated framework (Figure 2.1) to construct QA datasets with characteristic specification from a knowledge base. The framework revolves around an intermediate *graph query* representation, which helps to formalize question characteristics and collect answers. We first automatically generate graph queries from a knowledge base, and then employ human annotators to convert graph queries into questions.

Automating graph query generation brings with it the challenge of assessing the quality of graph queries and filtering out bad ones. Our framework tackles the challenge by combining

structured information in the knowledge base and statistical information from the Web. First, we identify *redundant components* in a graph query and develop techniques to remove them. Furthermore, based on the frequency of entities, classes, and relations mined from the Web, we quantify the *commonness* of a graph query and filter out too rare ones.

We employ a semi-automated approach for the conversion from graph query to natural language question, which provides two levels of paraphrasing: Common lexical forms of an entity (e.g., "*Queen Elizabeth*" and "*Her Majesty the Queen*" for `ElizabethII`) mined from the Web are used as entity paraphrases, and the remaining parts of a question are paraphrased by annotators. As a result, dozens of paraphrased questions can be produced for a single graph query.

To demonstrate the usefulness of question characteristics in QA evaluation, we construct a new dataset with over 5,000 questions based on Freebase using the proposed framework, and extensively evaluate several QA systems. A couple of new findings about system performance and question difficulty are discussed. For example, different from the results based on previous QA datasets [39], we find that semantic parsing in general works better than information extraction on our dataset. Information extraction based QA systems have trouble dealing with questions requiring aggregation or with multiple answers. A holistic understanding of the whole question is often needed for hard questions. The experiments point out an array of issues that future QA systems may need to solve.

2.2 Related Work

Early QA research has extensively studied problems like question taxonomy, answer type, and knowledge sources [40, 41, 35]. This work mainly targets at factoid questions with one or more answers that are guaranteed to exist in a KB.

A few KB-based QA datasets have been proposed recently. QALD [36] and FREE917 [23]

contain hundreds of hand-crafted questions. QALD also indicates whether a question requires aggregation. Both based on single Freebase triples, SIMPLEQUESTIONS [37] employ human annotators to formulate questions, while Serban et al. [38] use a recurrent neural network to automatically formulate questions. They are featured by a large size, but the questions only concern single triples, while our framework can generate questions involving multiple triples and various functions. Wang et al. [42] generate question-answer pairs for closed domains like basketball. They also first generate logical forms (λ-DCS formulae [43] in their case), and then convert logical forms into questions via crowdsourcing. Logical forms are first converted into canonical questions to help crowdsourcing workers. Different from previous works, we put a particular focus on generating questions with diversified characteristics in a systematic way, and examining the impact of different question characteristics in QA.

Another attractive way for QA dataset construction is to collect questions from search engine logs [44]. For example, WEBQUESTIONS [8] contains thousands of popular questions from Google search, and Yih et al. [45] have manually annotated these questions with logical forms. However, automatic characterization of questions is hard, while manual characterization is costly and requires expertise. Moreover, users' search behavior is shaped by search engines [46]. Due to the inadequacy of current search engines to answer advanced questions, users may adapt themselves accordingly and mostly ask simple questions. Thus questions collected in this way, to some extent, may still not well reflect the true distribution of user information needs, nor does it fully exploit the potential of KB-based QA. Collecting answers is yet another challenge for this approach. Yih et al. [45] show that only 66% of the WEBQUESTIONS answers, which were collected via crowdsourcing, are completely correct. On the other hand, although questions generated from a KB may not follow the distribution of user information needs, it has the advantage of explicit question characteristics, and enables programmatic configuration of question generation. Also, answer collecting is automated without involving human labor and errors.

2.3 Background

2.3.1 Knowledge Base

In this work, we mainly concern knowledge bases storing knowledge about entities and relations in the form of triples (simply *knowledge bases* hereafter). Suppose $\mathcal{E}$ is a set of entities, $\mathcal{L}$ a set of literals ($\mathcal{I} = \mathcal{E} \cup \mathcal{L}$ is also called *individuals*), $\mathcal{C}$ a set of classes, and $\mathcal{R}$ a set of directed relations, a knowledge base $\mathcal{K}$ consists of two parts: an *ontology* $\mathcal{O} \subseteq \mathcal{C} \times \mathcal{R} \times \mathcal{C}$ and a *model* $\mathcal{M} \subseteq \mathcal{E} \times \mathcal{R} \times (\mathcal{C} \cup \mathcal{E} \cup \mathcal{L})$. In other words, an ontology specifies classes and relations between classes, and a model consists of *facts* about individuals. Such knowledge bases can be naturally represented as a directed graph, e.g., Figure 2.1(a). Literal classes such as `Datetime` are represented as diamonds, and other classes are rounded rectangles. Individuals are shaded. We assume relations are typed, i.e., each relation is associated with a set of *domain* and *range* classes. Facts of a relation must be compatible with its domain and range constraints. Without loss of generality, we use Freebase (June 2013 version) in this work for compatibility with the to-be-tested QA systems. It has 24K classes, 65K relations, 41M entities, and 596M facts.

2.3.2 Graph Query

Motivated by the graph-structured nature of knowledge bases, we adopt a graph-centric approach. We hinge on a formal representation named *graph query* (e.g., Figure 2.1(c)), developed on the basis of Yih et al. [9] and influenced by λ-DCS [43].

Syntax. A graph query q is a connected directed graph built on a given knowledge base $\mathcal{K}$. It comprises three kinds of nodes: (1) *Question node* (double rounded rectangle), a free variable. (2) *Ungrounded node* (rounded rectangle or diamond), an existentially quantified variable. (3) *Grounded node* (shaded rounded rectangle or diamond), an individual. In addition, there are *functions* (shaded circle) such as $<$ and `count` applied on a node. Nodes are typed, each

associated with a class. Nodes are connected by directed edges representing relations. Entities on the grounded nodes are called *topic entities*.

Semantics. Graph query is a strict subset of λ-calculus. For example, the graph query in Figure 2.1(c) can be written in λ-calculus (an existentially quantified variable is imposed by $<$):

$$\begin{aligned}\lambda x.\exists y.\exists z.&\texttt{type}(x, \texttt{DeceasedPerson})\\ &\wedge \texttt{type}(y, \texttt{DeceasedPerson})\\ &\wedge \texttt{type}(z, \texttt{Datetime}) \wedge \texttt{parents}(x, y)\\ &\wedge \texttt{causeOfDeath}(x, \texttt{LungCancer})\\ &\wedge \texttt{causeOfDeath}(y, \texttt{LungCancer})\\ &\wedge \texttt{dateOfDeath}(x, z) \wedge < (z, \texttt{1960}).\end{aligned}$$

The *answer* of a graph query q, denoted as $[\![q]\!]_{\mathcal{K}}$, can be easily obtained from $\mathcal{K}$. For example, if $\mathcal{K}$ is stored in a RDF triplestore, then q can be automatically converted into a SPARQL query and run against $\mathcal{K}$ to get the answer. Compared with Yih et al. [9], graph queries are not constrained to be tree-structured, which grants us a higher expressivity. For example, linguistic phenomena like anaphora (e.g., Figure 2.1(d)) become easier to model.

2.4 Automatic Graph Query Generation

Our framework proceeds as follows: (1) Generate *query templates* from a knowledge base, ground the templates to generate graph queries, and collect answers (this section). (2) Refine graph queries to retain high-quality ones (Section 2.5). (3) Convert graph queries into questions via crowdsourcing (Section 2.6).

We now describe an algorithm to generate the query template shown in Figure 2.1(b) (excluding the function for now). For simplicity, we will focus on the case of a single question

node. Nevertheless, the proposed framework can be extended to generate graph queries with multiple question nodes. The algorithm takes as input an ontology (Figure 2.1(a)) and the desired number of edges. All the operations are conducted in a random manner to avoid systematic biases in query generation. The `DeceasedPerson` class is first selected as the question node. We then iteratively grow it by adding neighboring nodes and edges in the ontology. In each iteration, an existing node is selected, and a new edge, which might introduce a new node, is appended to it. For example, the relation `causeOfDeath`, whose domain includes `DeceasedPerson`, is first appended to the question node, and then one of its range classes, `CauseOfDeath`, is added as a new node. When a node with the class `CauseOfDeath` already exists, it is possible to add an edge without introducing a new node. The same relation or class can be added multiple times, e.g., "*parent of parent*".

Topic entities like `LungCancer` play an important role in a question. A query template contains some template nodes that can be grounded with different topic entities to generate different graph queries. We randomly choose a few nodes as template. It may cause problems. For example, grounding one node may make some others redundant. We conduct a formal study on this in Section 2.5.1.

Functions such as counting and comparatives are pervasive in real-life questions, e.g., "*how many*", "*the most recent*", and "*people older than*", but are scarce in existing QA datasets. We incorporate functions as an important question characteristic, and consider nine common functions, grouped into three categories: counting (`count`), superlative (`max`, `min`, `argmax`, `argmin`), and comparative ($>$, $\geq$, $<$, $\leq$). More functions can be incorporated in the future. We randomly add functions to compatible nodes in query templates. In the running example, the $<$ function imposes the constraint that only people who passed away before a certain date should be considered. Each query will have at most one function.

We then ground the template nodes with individuals to generate graph queries. A grounding is valid if the individuals conform with the class of the corresponding template nodes, and the

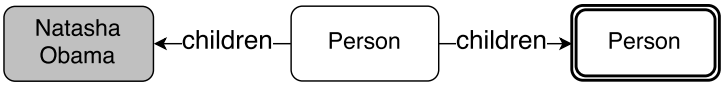


Figure 2.2: Mutual exclusivity example. Entities on different nodes should be different.

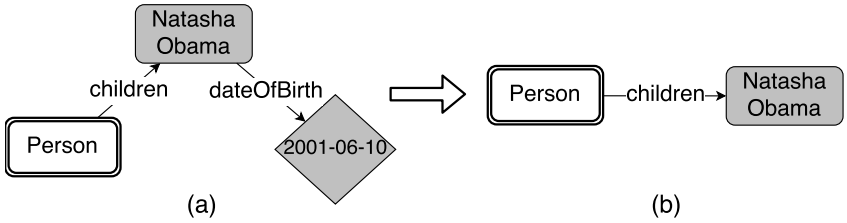


Figure 2.3: Query minimization example: (a) Graph query with redundant components. (b) Graph query after minimization.

resulted answer is not empty. For example, by grounding `CauseOfDeath` with `LungCancer` and `Datetime` with `1960`, we get the graph query in Figure 2.1(c). A query template can render multiple groundings.

Finally, we convert a graph query into a SPARQL query and execute it using Virtuoso Open-Source 7 to collect answers. We further impose *mutual exclusivity* in SPARQL queries, that is, the entities on any two nodes in a graph query should be different. Consider the example in Figure 2.2, which is asking for the siblings of Natasha Obama. Wihout mutual exclusivity, however, Natash Obama herself will also be included as an answer, which is not desired.

2.5 Query Refinement

Since graph queries are randomly generated, some of them may not correspond to an interesting question. Next we study two query characteristics, *redundancy* and *commonness*, based on which we provide mechanisms for automatic query refinement.

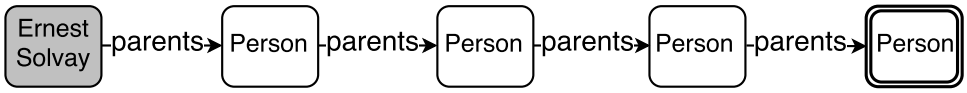


Figure 2.4: Uncommon query example. It is uncommon to ask for somebody's great-great–grandparents.

2.5.1 Query Redundancy and Minimization

Some components (nodes and edges) in a graph query may not effectively impose any constraint on the answer. The query in Figure 2.3(a) is to "*find the US president whose child is Natasha Obama,* ***and Natasha Obama was born on 2001-06-10***". Intuitively, the bold-faced clause does not change the answer of the question. Correspondingly, the `dateOfBirth` edge and the date node are redundant. As a comparison, removing any component from the query in Figures 2.3(b) will change the answer. Formally, given a knowledge base $\mathcal{K}$, a component in a graph query q is *redundant* iff. removing it does not change the answer $[\![q]\!]_{\mathcal{K}}$.

Redundancy can be desired or not. In a question, redundant information may be inserted to reduce ambiguity. In Figure 2.3(a), if one uses "*Natasha*" to refer to `NatashaObama`, there comes ambiguity since it may be matched with many other entities. The additional information "*who was born on 2001-06-10*" then helps. Next we describe an algorithm to remove redundancy from queries. One can choose to either only generate queries with no redundant component, or intentionally generate redundant queries and test QA systems in presence of redundancy.

We manage to generate *minimal* queries, for which there exists no sub-query having the same answer. An important theorem, as we prove in [13], is the equivalency of minimality and non-redundancy: A query is minimal *iff.* it has no redundant component. This renders a simple algorithm for query minimization, which directly detects and removes the redundant components in a query. We first examine every edge (in an arbitrary order), and remove an edge if it is redundant. Redundant nodes will then become disconnected to the question node and are thus eliminated. It is easy to prove that the produced query (e.g., Figure 2.3(b)) is

minimal, and has the same answer as the original query.

2.5.2 Commonness Checking

We now quantify the *commonness* of graph queries. The benefits of this study are two-fold. First, it provides a refinement mechanism to reduce too rare queries. Second, commonness is itself an important question characteristic. It is interesting to examine its impact on question difficulty. Consider the example in Figure 2.4, which asks for "*the great-great-grandparents of Ernest Solvay*". It is minimal and logically plausible. Few users, however, are likely to come up with it. Ernest Solvay is famous for the Solvay Conferences, but few people outside the science community may know him. Although `Person` and `parents` are common, asking for the great-great-grandparents is quite uncommon.

A query is more common if users would more likely come up with it. We define the commonness of a query q as its *probability* $p(q)$ of being picked among all possible queries from a knowledge base. The problem then boils down to estimating $p(q)$. It is hard, if not impossible, to exhaust the whole query space. We thus make the following simplification. We break down query commonness by components, assuming mutual independence between components, and omit functions:

$$p(q) = \prod_{i \in \mathcal{I}_q} p(i) \times \prod_{c \in \mathcal{C}_q} p(c) \times \prod_{r \in \mathcal{R}_q} p(r), \tag{2.1}$$

where $\mathcal{I}_q$, $\mathcal{C}_q$, $\mathcal{R}_q$ are the *multi-set* of the individuals, classes, and relations in q, respectively. Repeating components are thus accumulated (c.f. Figure 2.4).

We propose a data-driven method, using statistical information from the Web, to estimate $p(i)$, $p(c)$, and $p(r)$. Other methods like domain-knowledge based estimation are also applicable if available. We start with entity probability $p(e)$ (excluding literals for now). If users mention an entity more frequently, its probability of being observed in a question should be

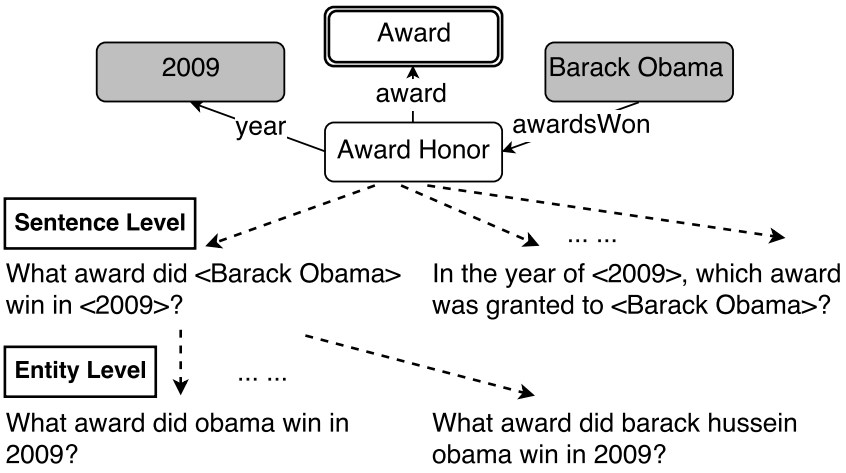


Figure 2.5: Question generation and paraphrasing.

higher. We use a large entity linking dataset, FACC1 [47], which identifies around 10 billion mentions of Freebase entities in over 1 billion web documents. The estimated linking precision and recall are 80-85% and 70-85%, respectively. Suppose entity e has $n(e)$ mentions, then $p(e) = \frac{n(e)}{\sum_{e' \in \mathcal{E}} n(e')}$. For a class c, probability $p(c)$ is higher if it has more frequently mentioned entities. If we use $e \in c$ to indicate e is an instance of c, then $p(c) = \frac{\sum_{e \in c} n(e)}{\sum_{c' \in \mathcal{C}} \sum_{e \in c'} n(e)}$. Estimating $p(r)$ requires relation extraction from texts, which is hard. We make the following simplification: If (e_1, r, e_2) is a fact in the knowledge base, we increase $n(r)$ by 1 if e_1 and e_2 co-occur in a document. This suffices to distinguish common relations from uncommon ones. We then define $p(r) = \frac{n(r)}{\sum_{r' \in \mathcal{R}} n(r')}$. Finally, we use frequency information from the knowledge base to smooth the probabilities, e.g., to avoid zero probabilities. The probability of literals are solely determined by the frequency information from the knowledge base.

2.6 Natural Language Conversion

In order to ensure naturalness and diversity, we employ human annotators to manually convert graph queries into natural language questions. We manage to provide two levels of paraphrasing (Figure 2.5). Each query is sent to multiple annotators for sentence-level paraphras-

	# of edges			Function				$\log_{10}(p(q))$				$\|\mathbf{A}\|$	
	1	2	3	none	count	super.	comp.	$[-40, 30)$	$[-30, 20)$	$[-20, 10)$	$[-10, 0)$	1	> 1
# of graph queries	321	144	35	350	61	42	47	60	135	283	22	332	168
# of questions	3094	1648	424	3855	710	332	269	653	1477	2766	270	3487	1679

Table 2.1: Characteristic statistics. $|\mathbf{A}|$ is answer cardinality.

ing. In addition, we use different lexical forms of an entity mined from FACC1 for entity-level paraphrasing. We provide a ranked list of common lexical forms and the corresponding frequency for each topic entity. For example, the lexical form list for `UnitedStatesOfAmerica` is "*us*" (108M), "*united states*" (44M), "*usa*" (22M), etc. Finally, graph queries are automatically translated into SPARQL queries to collect answers.

Natural language generation (NLG) [48, 38, 49] would be a good complement to our framework, the combination of which can lead to a fully-automated pipeline to generate QA datasets. For example, Serban et al. [38] automatically convert Freebase triples into questions with a neural network. More sophisticated NLG techniques able to handle graph queries involving multiple relations and various functions are an interesting future direction.

2.7 Experiments

We have constructed a new QA dataset, named GRAPHQUESTIONS, using the proposed framework, and tested several QA systems to show that it enables fine-grained inspection of QA systems.

2.7.1 Dataset Construction

We first randomly generated a set of minimal graph queries, and removed the ones whose commonness is below a certain threshold. The remaining graph queries were then screened by graduate students, and a *canonical* question was generated for each query, with each be-

Question	Domain	Answer	# of edges	Function	$\log_{10}(p(q))$	|A|
Find terrorist organizations involved in **September 11 attacks**.	Terrorism	alQaeda	1	none	-16.67	1
The **September 11 attacks** were carried out with the involvement of what terrorist organizations?						
Who did **nine eleven**?						
How many children of **Eddard Stark** were born in **Winterfell**?	Fictional Universe	3	2	count	-23.34	1
Winterfell is the home of how many of **Eddard Stark**'s children?						
What's the number of **Ned Stark**'s children whose birthplace is **Winterfell**?						
In which month does the average rainfall of **New York City** exceed **86** mm?	Travel	March, August ...	3	comp.	-37.84	7
Rainfall averages more than **86** mm in **New York City** during which months?						
List the calendar months when **NYC** averages in excess of **86** millimeters of rain?						

Table 2.2: Example questions and characteristics. Three sentence-level paraphrases are shown for each graph query, with the last one also involving entity-level paraphrasing. Topic entities are bold-faced.

ing verified by at least two students. We recruited 160 crowdsourcing workers from Amazon MTurk to generate sentence-level paraphrases of the canonical questions. Trivial paraphrases (e.g., "*which city*" vs. "*what city*") were manually removed to retain a high diversity in paraphrasing. At most 3 entity-level paraphrases were used for each sentence-level paraphrase.

2.7.2 Dataset Analysis

GRAPHQUESTIONS contains 500 graph queries, 2,460 sentence-level paraphrases, and 5,166 questions[2]. The dataset presents a high diversity and covers a wide range of domains including `People`, `Astronomy`, `Medicine`, etc. Specifically, it contains 148, 506, 596, 376

[2]For each query template, we only generate one graph query, but one can also generate multiple graph queries, and easily get the corresponding questions by replacing the topic entities. This will significantly increase the total number of questions, and can be helpful in training.

Dataset	# of Questions	# of Multi-relation	Function (count/super./comp.)	Commonness	Paraphrase	Multi-answer
GRAPHQUESTIONS (this work)	5166	2072	710 / 332 / 269	+	+	+
WEBQUESTIONSSP [45][1]	4737	2075	2 / 168 / 334	-	-	+
FREE917 [23]	917	229	185 / 0 / 0	-	-	+
Serban et al. [38]	30M	0	0 / 0 / 0	-	-	-
SIMPLEQUESTIONS [37]	108K	0	0 / 0 / 0	-	-	-

Table 2.3: Comparison of QA datasets constructed from Freebase. GRAPHQUESTIONS is the richest in question characteristics.

and 3,026 distinct domains, classes, relations, topic entities, and words, respectively. We evenly split GRAPHQUESTIONS into a training set and a testing set. All the paraphrases of the same graph query are in the same set.

While there are other question characteristics derivable from graph query, we will focus on the following ones: *structure complexity*, *function*, *commonness*, *paraphrasing*, and *answer cardinality*. We use the number of edges to quantify structure complexity, and limit it to at most 3. Commonness is limited to $\log_{10}(p(q)) \geq -40$ (c.f. Eq. 2.1). As shown in Section 2.7.4, such questions are already very hard for existing QA systems. Nevertheless, the proposed framework can be used to generate questions with different characteristic distributions. Some statistics are shown in Table 2.1.

Several example questions are shown in Table 2.2. Sentence-level paraphrasing requires to handle both commands (the first example) and "*Wh*" questions, light verbs ("*Who did nine eleven?*"), and changes of syntactic structure ("*The September 11 attacks were carried out with the involvement of what terrorist organizations?*"). Entity-level paraphrasing tests the capability of QA systems on abbreviation ("*NYC*" for `New York City`), world knowledge ("*Her Majesty the Queen*" for `ElizabethII`), or even common typos ("*Shakspeare*" for `WilliamShakespeare`). Numbers and dates are also common, e.g., "*Which computer operating system was released on Sept. the 20th, 2008?*"

We compare several QA datasets constructed from Freebase, shown in Table 2.3. Datasets

focusing on single-relation questions are of a larger scale, but are also of a significant lack in question characteristics. Overall GRAPHQUESTIONS presents the highest diversity in question characteristics.

2.7.3 Setup

We evaluate three QA systems whose source code is publicly available: SEMPRE [8], PARASEMPRE [25], and JACANA [30]. SEMPRE and PARASEMPRE follow the semantic parsing paradigm. SEMPRE conducts a bottom-up beam-based parsing on questions to find the best logical form. PARASEMPRE, in a reverse manner, enumerates a set of logical forms, generates a canonical utterance for each logical form, and ranks logical forms according to how well the canonical utterance paraphrases the input question. In contrast, JACANA follows the information extraction paradigm, and builds a classifier to directly predict whether an individual is the answer. They all use Freebase.

The main metric for answer quality is the average F1 score, following Berant and Liang [25]. Because a question can have more than one answer, individual precision, recall, and F1 scores are first computed on each question and then averaged. When a system generates no response for a question, precision is 1, recall is 0, and F1 is 0. Average runtime is used for efficiency. Results are shown in percentage. Systems are trained on the training set using the suggested configurations. We use student's t test at $p = 0.05$ for significance test.

2.7.4 Results

Overall Evaluation

Compared with the scores on WEBQUESTIONS (30%-40%), the scores on GRAPHQUESTIONS are lower (Table 2.4). This is because GRAPHQUESTIONS contains questions over a broader range of difficulty levels. For example, it is more diverse in topics; also the scores

System	F1	Time/s
SEMPRE	10.80	56.19
PARASEMPRE	12.79	18.43
JACANA	5.08	2.01

Table 2.4: Overall performance on GRAPHQUESTIONS.

become much closer when excluding paraphrasing (Section 2.7.4).

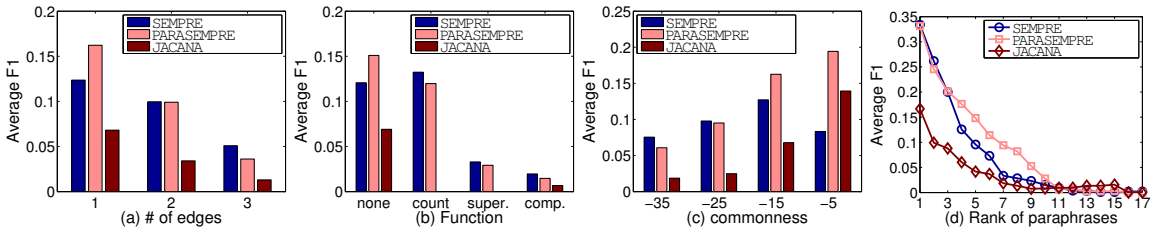
JACANA achieves a comparable F1 score with SEMPRE and PARASEMPRE on WEBQUESTIONS [39]. On GRAPHQUESTIONS, however, SEMPRE and PARASEMPRE significantly outperform JACANA (both $p < 0.0001$). The following experiments will give more insights about where the performance difference comes from. On the other hand, JACANA is much faster, showing an advantage of information extraction. The semantic parsing systems spend a lot of time on executing SPARQL queries. Bypassing SPARQL and directly working on the knowledge base may be a promising way to speed up semantic parsing on large knowledge bases [9].

Fine-grained Evaluation

With explicitly specified question characteristics, we are able to further inspect QA systems.

Structure Complexity. We first break down system performance by structure. Answer quality is in general sensitive to the complexity of question structure: As the number of edges increases, F1 score decreases (Figure 2.6(a)). The tested systems often fail to take into account auxiliary constraints in a question. For example, for "*How many children of Ned Stark were born in Winterfell?*" SEMPRE fails to identify the constraint "*born in Winterfell*", so it also considers Ned Stark's bastard son, Jon Snow, as an answer, who was not born in Winterfell. Answering questions involving multiple relations using large knowledge bases remain an open

[2]WEBQUESTIONSSP is WEBQUESTIONS with manually annotated logical forms. Only those with a full logical form are included (4737 / 5810).



(a) structure complexity (b) function (c) commonness (d) paraphrase

Figure 2.6: Performance breakdown. Note that in (c) $x = -5$ indicates the commonness range $-10 \leq \log_{10}(p(q)) < 0$.

problem. The large size of knowledge bases prohibits exhaustive search, so smarter algorithms are needed to efficiently prune the answer space. Berant and Liang [34] point out an interesting direction, leveraging agenda-based parsing with imitation learning for efficient search in the answer space.

Function. In terms of functions, while SEMPRE and PARASEMPRE perform well on `count` questions, all the tested systems perform poorly on questions with superlatives or comparatives (Figure 2.6(b)). JACANA has trouble dealing with functions because it does not conduct quantitative analysis over the answer space. SEMPRE and PARASEMPRE do not generate logical forms with superlatives and comparatives, so they cannot answer such questions well.

Commonness. Not surprisingly, more common questions are in general easier to answer (Figure 2.6(c)). An interesting observation is that SEMPRE's performance gets worse on the most common questions. The cause is likely rooted in how the QA systems construct their candidate answer sets. PARASEMPRE and JACANA exhaustively construct candidate sets, while SEMPRE employs a bottom-up beam search, making it more sensitive to the size of the candidate answer space. Common entities like `UnitedStatesOfAmerica` are often featured by a high degree in knowledge bases (e.g., 1 million neighboring entities), which dramatically increases the size of the candidate answer space. During SEMPRE's iterative beam search, many correct logical

forms may have fallen off beam before getting into the final candidate set. We checked the percentage of questions for which the correct logical form is in the final candidate set, and found that it decreased from 19.8% to 16.7% when commonness increased from -15 to -5, providing an evidence for the intuition.

Paraphrasing. It is critical for a system to tolerate the wording varieties of users. We make the first effort to evaluate QA systems on paraphrasing. For each system, we rank, in descending order, all the paraphrases derived from the same graph query by their F1 score achieved by the system, and then compute the average F1 score of each rank. In Figure 2.6(d), the decreasing rate of a curve thus describes a system's robustness to paraphrasing; the higher, the less robust. All the systems achieve a reasonable score on the top-1 paraphrases, i.e., when a system can choose the paraphrase it can best answer. The F1 scores drop quickly in general. On the fourth-ranked paraphrases, the F1 score of SEMPRE, PARASEMPRE, and JACANA are respectively only 37.65%, 53.2%, and 36.2% of their score on the top-1 paraphrases. Leveraging paraphrasing in its model, PARASEMPRE does seem to be more robust. The results show that how to handle paraphrased questions is still a challenging problem.

Answer Cardinality. SEMPRE and JACANA get a significantly lower F1 score (both $p < 0.0001$) on multi-answer questions (Table 2.5), mainly coming from a decrease on recall. The decrease of PARASEMPRE is not significant (p=0.29). The particularly significant decrease of JACANA demonstrates the difficulty of training a classifier that can predict all of the answers correctly; semantic parsing is more robust in this case. The precision of SEMPRE is high because it generates no response for many questions. Note that under the current definition, the average F1 score is not the harmonic mean of the average precision and recall (c.f. Section 2.7.3).

System	$\|\mathbf{A}\|$	Prec.	Rec.	F1
SEMPRE	1	59.81	16.11	12.68
	> 1	62.38	9.17	6.78
PARASEMPRE	1	17.42	17.58	13.25
	> 1	19.65	17.23	11.82
JACANA	1	14.77	6.56	6.56
	> 1	11.80	1.43	1.98

Table 2.5: Performance breakdown by answer cardinality $|\mathbf{A}|$.

2.8 Conclusion

We proposed a framework to generate characteristic-rich questions for question answering (QA) evaluation. Using the proposed framework, we constructed a new and challenging QA dataset, and extensively evaluated several QA systems. The findings point out an array of issues that future QA research may need to solve.

2.9 Acknowledgements

This research was sponsored in part by the Army Research Laboratory under cooperative agreements W911NF09-2-0053, NSF IIS 0954125, and NSF IIS 1528175. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

Chapter 3

Cross-domain Semantic Parsing

3.1 Introduction

Semantic parsing, which maps natural language utterances into computer-understandable logical forms, has drawn substantial attention recently as a promising direction for developing natural language interfaces to computers. Semantic parsing has been applied in many domains, including querying data/knowledge bases [3, 50, 8], controlling Internet-of-Things (IoT) devices [51], and communicating with robots [52, 53, 54, 55].

Despite the wide applications, studies on semantic parsing have mainly focused on the *in-domain* setting, where both training and testing data are drawn from the same domain. How to build semantic parsers that can learn across domains remains an under-addressed problem. In this work, we study *cross-domain semantic parsing*. We model it as a domain adaptation problem [56], where we are given some *source* domains and a *target* domain, and the core task is to adapt a semantic parser trained on the source domains to the target domain (Figure 3.1). The benefits are two-fold: (1) by training on the source domains, the cost of collecting training data for the target domain can be reduced, and (2) the data of source domains may provide information complementary to the data collected for the target domain, leading to better performance

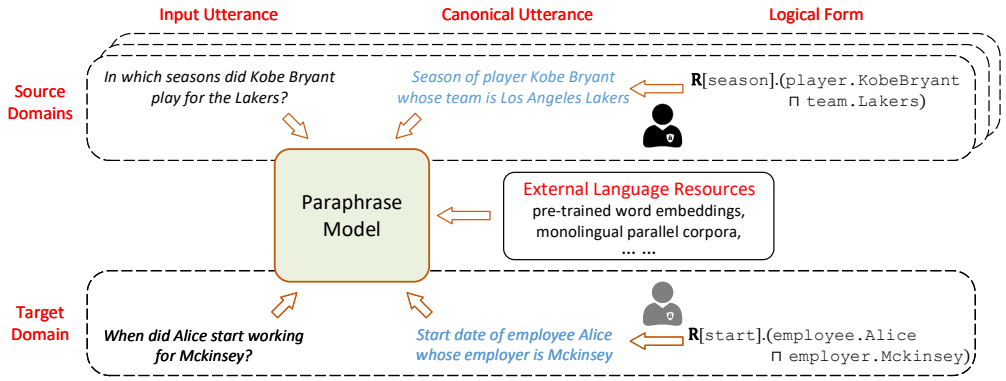


Figure 3.1: Cross-domain semantic parsing via paraphrasing framework. In a deterministic way, logical forms are first converted into canonical utterances in natural language. A paraphrase model then learns from the source domains and adapts to the target domain. External language resources can be incorporated in a consistent way across domains.

on the target domain.

This is a very challenging task. Traditional domain adaptation [56, 57] only concerns natural languages, while semantic parsing concerns both natural and formal languages. Different domains often involve different predicates. In Figure 3.1, from the source BASKETBALL domain a semantic parser can learn the semantic mapping from natural language to predicates like `team` and `season`, but in the target SOCIAL domain it needs to handle predicates like `employer` instead. Worse still, even for the same predicate, it is legitimate to use arbitrarily different predicate symbols, e.g., other symbols like `hired_by` or even `predicate1` can also be used for the `employer` predicate. Therefore, directly transferring the mapping from natural language to predicate symbols learned from source domains to the target domain may not be much beneficial.

Inspired by the recent success of paraphrasing based semantic parsing [25, 58], we propose to use natural language as an intermediate representation for cross-domain semantic parsing. As shown in Figure 3.1, logical forms are converted into canonical utterances in natural language, and semantic parsing is reduced to paraphrasing. It is the knowledge of paraphrasing, at lexical, syntactic, and semantic levels, that will be transferred across domains.

Still, adapting a paraphrase model to a new domain is a challenging and under-addressed problem. To give some idea of the difficulty, for each of the eight domains in the popular OVERNIGHT [58] dataset, 30% to 55% of the words never occur in any of the other domains, a similar problem observed in domain adaptation for machine translation [59]. The paraphrase model therefore can get little knowledge for a substantial portion of the target domain from the source domains. We introduce pre-trained word embeddings such as WORD2VEC [60] to combat the vocabulary variety across domains. Based on recent studies on neural network initialization, we conduct a statistical analysis of pre-trained word embeddings and discover two problems that may hurdle their direct use in neural networks: *small micro variance*, which hurts optimization, and *large macro variance*, which hurts generalization. We propose to *standardize* pre-trained word embeddings, and show its advantages both analytically and experimentally.

On the OVERNIGHT dataset, we show that cross-domain training under the proposed framework can significantly improve model performance. We also show that, compared with directly using pre-trained word embeddings or normalization as in previous work, the proposed standardization technique can lead to about 10% absolute improvement in accuracy.

3.2 Cross-domain Semantic Parsing

3.2.1 Problem Definition

Unless otherwise stated, we will use u to denote input utterance, c for canonical utterance, and z for logical form. We denote $\mathcal{U}$ as the set of all possible utterances. For a domain, suppose $\mathcal{Z}$ is the set of logical forms, a semantic parser is a mapping $f : \mathcal{U} \rightarrow \mathcal{Z}$ that maps every input utterance to a logical form (a `null` logical form can be included in $\mathcal{Z}$ to reject out-of-domain utterances).

In cross-domain semantic parsing, we assume there are a set of K source domains $\{\mathcal{Z}_i\}_{i=1}^{K}$,

each with a set of training examples $\{(u_j^i, z_j^i)\}_{j=1}^{N_i}$. It is in principle advantageous to model the source domains separately [56], which retrains the possibility of separating domain-general information from domain-specific information, and only transferring the former to the target domain. For simplicity, here we merge the source domains into a single domain $\mathcal{Z}_s$ with training data $\{(u_i, z_i)\}_{i=1}^{N_s}$. The task is to learn a semantic parser $f: \mathcal{U} \rightarrow \mathcal{Z}_t$ for a target domain $\mathcal{Z}_t$, for which we have a set of training examples $\{(u_i, z_i)\}_{i=1}^{N_t}$. Some characteristics can be summarized as follows:

- $\mathcal{Z}_t$ and $\mathcal{Z}_s$ can be totally disjoint.
- The input utterance distribution of the source and the target domain can be independent and differ remarkably.
- Typically $N_t \ll N_s$.

In the most general and challenging case, $\mathcal{Z}_t$ and $\mathcal{Z}_s$ can be defined using different formal languages. Because of the lack of relevant datasets, here we restrain ourselves to the case where $\mathcal{Z}_t$ and $\mathcal{Z}_s$ are defined using the same formal language, e.g., λ-DCS [43] as in the OVERNIGHT dataset.

3.2.2 Framework

Our framework follows the research line of semantic parsing via paraphrasing [25, 58]. While previous work focuses on the in-domain setting, we discuss its applicability and advantages in the cross-domain setting, and develop techniques to address the emerging challenges in the new setting.

Canonical utterance. We assume a *one-to-one* mapping $g: \mathcal{Z} \rightarrow \mathcal{C}$, where $\mathcal{C} \subset \mathcal{U}$ is the set of canonical utterances. In other words, every logical form will be converted into a unique

canonical utterance deterministically (Figure 3.1). Previous work [58] has demonstrated how to design such a mapping, where a domain-general grammar and a domain-specific lexicon are constructed to automatically convert every logical form to a canonical utterance. In this work, we assume the mapping is given[1], and focus on the subsequent paraphrasing and domain adaptation problems.

This design choice worths some discussion. The grammar, or at least the lexicon for mapping predicates to natural language, needs to be provided by domain administrators. This indeed brings an additional cost, but we believe it is reasonable and even necessary for three reasons: (1) Only domain administrators know the predicate semantics the best, so it has to be them to reveal that by grounding the predicates to natural language. (2) Otherwise, predicate semantics can only be learned from supervised training data of each domain, bringing a significant cost on data collection. (3) Canonical utterances are understandable by average users, and thus can also be used for training data collection via crowdsourcing [58, 13], which can amortize the cost.

Take comparatives as an example. In logical forms, comparatives can be legitimately defined using arbitrarily different predicates in different domains, e.g., `<`, `smallerInSize`, or even predicates with an ambiguous surface form, like `lt`. When converting logical form to canonical utterance, however, domain administrators have to choose common natural language expressions like "*less than*" and "*smaller*", providing a shared ground for cross-domain semantic parsing.

Paraphrase model. In the previous work based on paraphrasing [25, 58], semantic parsers are implemented as log-linear models with hand-engineered domain-specific features (including paraphrase features). Considering the recent success of representation learning for domain adaptation [61, 62], we propose a paraphrase model based on the sequence-to-sequence (Seq2Seq) model [63], which can be trained end to end without feature engineering. We show

[1] In the experiments we use the provided canonical utterances of the OVERNIGHT dataset.

that it outperforms the previous log-linear models by a large margin in the in-domain setting, and can easily adapt to new domains.

Pre-trained word embeddings. An advantage of reducing semantic parsing to paraphrasing is that external language resources become easier to incorporate. Observing the vocabulary variety across domains, we introduce pre-trained word embeddings to facilitate domain adaptation. For the example in Figure 3.1, the paraphrase model may have learned the mapping from "play for" to "whose team is" in a source domain. By acquiring word similarities ("play"-"work" and "team"-"employer") from pre-trained word embeddings, it can establish the mapping from "work for" to "whose employer is" in the target domain, even without in-domain training data. We analyze statistical characteristics of the pre-trained word embeddings, and propose standardization techniques to remedy some undesired characteristics, which hurdle their direct use in neural networks.

Domain adaptation protocol. We will use the following protocol: (1) train a paraphrase model using the data of the source domain, (2) use the learned parameters to initialize a model in the target domain, and (3) fine-tune the model using the training data of the target domain.

3.2.3 Prior Work

While most studies on semantic parsing so far have focused on the in-domain setting, there are a number of studies of particular relevance to this work. In the recent efforts of scaling semantic parsing to large knowledge bases like Freebase [21], researchers have explored several ways to infer the semantics of knowledge base relations unseen in training, which are often based on at least one (often both) of the following assumptions: (1) *Distant supervision*. Freebase entities can be linked to external text corpora, and serve as anchors for seeking semantics of Freebase relations from text. For example, Cai and Alexander [64], among others [8, 65], use sentences from Wikipedia that contain any entity pair of a Freebase relation

as the support set of the relation. (2) *Self-explaining predicate symbols*. Most Freebase relations are described using a carefully chosen symbol (surface form), e.g., `place_of_birth`, which provides strong cues for their semantics. For example, Yih et al. [9] directly compute the similarity of input utterance and the surface form of Freebase relations via a convolutional neural network. Kwiatkowski et al. [24] also extract lexical features from input utterance and the surface form of entities and relations. They have actually evaluated their model on Freebase sub-domains not covered in training, and have shown impressive results. However, in the more general setting of cross-domain semantic parsing, we may have neither of these luxuries. Distant supervision may not be available (e.g., IoT devices involving no entities but actions), and predicate symbols may not provide enough cues (e.g., `predicate1`). In this case, seeking additional inputs from domain administrators is probably necessary.

In parallel of this work, Herzig and Berant [66] have explored another direction of semantic parsing with multiple domains, where they use all the domains to train a single semantic parser, and attach a domain-specific encoding to the training data of each domain to help the semantic parser differentiate between domains. We pursue a different direction: we train a semantic parser on some source domains and adapt it to the target domain. Another difference is that their work directly maps utterances to logical forms, while ours is based on paraphrasing.

Cross-domain semantic parsing can be seen as a way to reduce the cost of training data collection, which resonates with the recent trend in semantic parsing. Berant et al. [8] propose to learn from utterance-denotation pairs instead of utterance-logical form pairs, while Wang et al. [58] and Su et al. [13] manage to employ crowd workers with no linguistic expertise for data collection. Jia and Liang [67] propose an interesting form of data augmentation. They learn a grammar from existing training data, and generate new examples from the grammar by recombining segments from different examples.

We use natural language as an intermediate representation to transfer knowledge across domains, and assume the mapping from the intermediate representation (canonical utterance)

to logical form can be done deterministically. Several other intermediate representations have also been used, such as Combinatory Categorial Grammars [24, 27], dependency tree [68, 69], and semantic role structure [70]. But their main aim is to better represent input utterances with a richer structure. A separate ontology matching step is needed to map the intermediate representation to logical form, which requires domain-dependent training.

A number of other related studies have also used paraphrasing. For example, Fader et al. [71] leverage question paraphrases to for question answering, while Narayan et al. [72] generate paraphrases as a way of data augmentation.

Cross-domain semantic parsing can greatly benefit from the rich literature of domain adaptation and transfer learning [56, 57, 73, 61]. For example, Chelba and Acero [74] use parameters trained in the source domain as prior to regularize parameters in the target domain. The feature augmentation technique from Daumé III [75] can be very helpful when there are multiple source domains. We expect to see many of these ideas to be applied in the future.

3.3 Paraphrase Model

In this section we propose a paraphrase model based on the Seq2Seq model [63]. Similar models have been used in semantic parsing [67, 10] but for directly mapping utterances to logical forms. We demonstrate that it can also be used as a paraphrase model for semantic parsing. Several other neural models have been proposed for paraphrasing [76, 77, 78], but it is not the focus of this work to compare all the alternatives.

For an input utterance $u = (u_1, u_2, \ldots, u_m)$ and an output canonical utterance $c = (c_1, c_2, \ldots, c_n)$, the model estimates the conditional probability $p(c|u) = \prod_{j=1}^{n} p(c_j|u, c_{1:j-1})$. The tokens are first converted into vectors via a word embedding layer ϕ. The initialization of the word embedding layer is critical for domain adaptation, which we will further discuss in Section 3.4.

The *encoder*, which is implemented as a bi-directional recurrent neural network (RNN),

first encodes u into a sequence of state vectors $(h_1, h_2, \ldots, h_m)$. The state vectors of the forward RNN and the backward RNN are respectively computed as:

$$\overrightarrow{h}_i = GRU_{fw}(\phi(u_i), \overrightarrow{h}_{i-1})$$
$$\overleftarrow{h}_i = GRU_{bw}(\phi(u_i), \overleftarrow{h}_{i+1})$$

where gated recurrent unit (GRU) as defined in [79] is used as the recurrence. We then concatenate the forward and backward state vectors, $h_i = [\overrightarrow{h}_i, \overleftarrow{h}_i], i = 1, \ldots, m$.

We use an attentive RNN as the *decoder*, which will generate the output tokens one at a time. We denote the state vectors of the decoder RNN as $(d_1, d_2, \ldots, d_n)$. The attention takes a form similar to [80]. For the decoding step j, the decoder is defined as follows:

$$d_0 = \tanh(W_0[\overrightarrow{h}_m, \overleftarrow{h}_1])$$
$$u_{ji} = v^T \tanh(W_1 h_i + W_2 d_j)$$
$$\alpha_{ji} = \frac{u_{ji}}{\sum_{i'=1}^{m} u_{ji'}}$$
$$h'_j = \sum_{i=1}^{m} \alpha_{ji} h_i$$
$$d_{j+1} = GRU([\phi(c_j), h'_j], d_j)$$
$$p(c_j | u, c_{1:j-1}) \propto \exp(U[d_j, h'_j])$$

where W_0, W_1, W_2, v and U are model parameters. The decoder first calculate normalized attention weights α_{ji} over encoder states, and get a summary state h'_j. The summary state is then used to calculate the next decoder state d_{j+1} and the output probability distribution $p(c_j | u, c_{1:j-1})$.

Training. Given a set of training examples $\{(u_i, c_i)\}_{i=1}^N$, we minimize the cross-entropy loss $-\frac{1}{N}\sum_{i=1}^{N} \log p(c_i|u_i)$, which maximizes the log probability of the correct canonical utterances. We apply dropout [81] on both input and output of the GRU cells to prevent overfitting.

Testing. Given a domain $\{\mathcal{Z}, \mathcal{C}\}$, there are two ways to use a trained model. One is to use it to *generate* the most likely output utterance u' given an input utterance u [63],

$$u' = \arg\max_{u' \in \mathcal{U}} p(u'|u).$$

In this case u' can be any utterance permissable by the output vocabulary, and may not necessarily be a legitimate canonical utterance in $\mathcal{C}$. This is more suitable for large domains with a lot of logical forms, like Freebase. An alternative way is to use the model to *rank* the legitimate canonical utterances [82]:

$$c = \arg\max_{c \in \mathcal{C}} p(c|u),$$

which is more suitable for small domains having a limited number of logical forms, like the ones in the OVERNIGHT dataset. We will adopt the second strategy. It is also very challenging; random guessing will give almost zero accuracy. It is also possible to first find a smaller set of candidates to rank via beam search [8, 58].

3.4 Pre-trained Word Embeddings for Domain Adaptation

Pre-trained word embeddings like WORD2VEC have a great potential to combat the vocabulary variety across domains. For example, we can use WORD2VEC embeddings to initialize the word embedding layer of the source domain, with the hope that the other parameters in the model will co-adapt with the word embeddings during training in the source domain, and generalize better to the out-of-vocabulary words (but covered by WORD2VEC) in the target domain.

However, a statistical analysis of the WORD2VEC embeddings shows that it might not be a good idea to directly use them in a deep neural network, and proper *standardization* is needed. Our analysis will be based on the 300-dimensional WORD2VEC embeddings trained

Initialization	L2 norm	Micro Variance	Cosine Sim.
Random	17.3 ± 0.45	1.00 ± 0.05	0.00 ± 0.06
WORD2VEC	2.04 ± 1.08	0.02 ± 0.02	0.13 ± 0.11
WORD2VEC + ES	17.3 ± 0.05	1.00 ± 0.00	0.13 ± 0.11
WORD2VEC + FS	16.0 ± 8.47	1.09 ± 1.31	0.12 ± 0.10
WORD2VEC + EN	1.00 ± 0.00	0.01 ± 0.00	0.13 ± 0.11

Table 3.1: Word embedding initializations. Random: random sampling from $U(-\sqrt{3}, \sqrt{3})$, thus unit variance. WORD2VEC: raw WORD2VEC embeddings. ES: per-example standardization. FS: per-feature standardization. EN: per-example normalization. Cosine similarity is computed on a random (but fixed) set of 1M word pairs.

on the 100B-word Google News corpus[2]. It contains 3 million words, leading to a 3M-by-300 word embedding matrix.

Neural networks are very sensitive to initialization [83]. The "rule of thumb" to randomly initialize word embeddings in neural networks is to sample from a uniform or Gaussian distribution with *unit variance*, which works well for a wide range of neural network models in general. We therefore use it as a reference to compare different word embedding initializations. We report the L2 norm and the per-example (per-row) variance of different initializations in Table 3.1. The statistics show why using the raw WORD2VEC embeddings may hurt model performance. Compared with random initialization, both the L2 norm and the per-example variance (denoted as *micro variance*) of the WORD2VEC embeddings are much smaller, while the variance of the embedding of different words (denoted as *macro variance*) is much larger (the maximum and the minimum L2 norm are 21.1 and 0.015, respectively). Small micro variance can make the variance of neuron activations starts off too small[3], implying a poor starting point in the parameter space. On the other hand, large macro variance may make a model hard to generalize to words unseen in training.

[2] `https://code.google.com/archive/p/word2vec/`

[3] Under some conditions, including using Xavier initialization (also introduced in that paper and now widely used) for weights, Glorot and Bengio [84] have shown that the activation variances in a feedforward neural network will be roughly the same as the input variances (word embeddings here) at the beginning of training.

Based on the above analysis, we propose to do *unit variance standardization* (standardization for short) on the pre-trained word embeddings. There are two possible ways, *per-example standardization*, which standardizes each row of the embedding matrix to unit variance, and *per-feature standardization*, which standardizes each column instead. We do not make the rows or columns zero mean. Per-example standardization enjoys the goodness of both random initialization and pre-trained word embeddings: it fixes the small micro variance problem as well as the large macro variance problem of pre-trained word embeddings, while still preserving cosine similarity, i.e., word similarity. Per-feature standardization does not preserve cosine similarity, nor does it fix the large macro variance problem. However, it enjoys the benefit of *global* statistics, in contrast to the *local* statistics of each word embedding used in per-example standardization. Therefore, in problems where the testing and training vocabularies are similar, per-feature standardization may be more advantageous. Both standardizations lose vector length information. Levy et al. [85] have suggested *per-example normalization*[4] of pre-trained word embeddings for lexical tasks like word similarity and analogy, which do no involve neural networks. Making the word embeddings unit length alleviates the large macro variance problem, but the small micro variance problem remains.

This is indeed a pretty simple trick, and per-feature standardization (with zero mean) is also a standard data preprocessing method. However, it is not self-evident that this kind of standardization shall be applied on pre-trained word embeddings before using them in neural networks, especially with the obvious downside of rendering the word embedding algorithm's loss function sub-optimal.

We expect this to be less of a issue for large-scale problems with a large vocabulary and abundant training examples. For example, Vinyals et al. [80] have found that directly using raw WORD2VEC embeddings for initialization can bring a consistent, though small, improvement in

[4]It can also be found in the implementation of Glove [86]: `https://github.com/stanfordnlp/GloVe`

Metric	CALENDAR	BLOCKS	HOUSING	RESTAURANTS	PUBLICATIONS	RECIPES	SOCIAL	BASKETBALL
# of example (N)	837	1995	941	1657	801	1080	4419	1952
# of logical form ($\|\mathcal{Z}\|, \|\mathcal{C}\|$)	196	469	231	339	149	124	624	252
vocab. size ($\|\mathcal{V}\|$)	228	227	318	342	203	256	533	360
% $\in$ other domains	71.1	61.7	60.7	55.8	65.6	71.9	46.0	45.6
% $\in$ WORD2VEC	91.2	91.6	88.4	88.6	91.1	93.8	86.9	86.9
% $\in$ other domains + WORD2VEC	**93.9**	**93.8**	**90.9**	**90.4**	**95.6**	**97.3**	**89.3**	**89.4**

Table 3.2: Statistics of the domains in the OVERNIGHT dataset. Pre-trained word embeddings cover most of the words in each domain, paving a way for domain adaptation.

Method	CALENDAR	BLOCKS	HOUSING	RESTAURANTS	PUBLICATIONS	RECIPES	SOCIAL	BASKETBALL	Avg.
Previous Methods									
Wang et al. [58]	74.4	41.9	54.0	75.9	59.0	70.8	48.2	46.3	58.8
Xiao et al. [87]	75.0	55.6	61.9	80.1	75.8	–	80.0	80.5	72.7
Jia and Liang [67]	78.0	58.1	71.4	76.2	76.4	79.6	81.4	85.2	75.8
Herzig and Berant [66]	82.1	**62.7**	78.3	82.2	**80.7**	82.9	81.7	86.2	79.6
Our Methods									
Random + I	75.6	60.2	67.2	77.7	77.6	80.1	80.7	86.5	75.7
Random + X	79.2	54.9	74.1	76.2	78.5	82.4	82.5	86.7	76.9
WORD2VEC + I	67.9	59.4	52.4	75.0	64.0	73.2	77.0	87.5	69.5
WORD2VEC + X	78.0	54.4	63.0	81.3	74.5	83.3	81.5	83.1	74.9
WORD2VEC + EN + I	63.1	56.1	60.3	75.3	65.2	69.0	76.4	81.8	68.4
WORD2VEC + EN + X	78.0	52.6	63.5	74.7	65.2	80.6	79.9	80.8	71.2
WORD2VEC + FS + I	78.6	62.2	67.7	78.6	75.8	85.7	81.3	86.7	77.1
WORD2VEC + FS + X	**82.7**	59.4	75.1	80.4	78.9	85.2	81.8	87.2	78.9
WORD2VEC + ES + I	79.8	60.2	71.4	81.6	78.9	84.7	82.9	86.2	78.2
WORD2VEC + ES + X	82.1	62.2	**78.8**	**83.7**	80.1	**86.1**	**83.1**	**88.2**	**80.6**

Table 3.3: Main experiment results. I: in-domain, X: cross-domain, EN: per-example normalization, FS: per-feature standardization, ES: per-example standardization.

neural constituency parsing. However, for smaller-scale problems (e.g., an application domain of semantic parsing can have a vocabulary size of only a few hundreds), this issue becomes more critical. Initialized with the raw pre-trained embeddings, a model may quickly fall into a poor local optimum and may not have enough signal to escape. Because of the large macro variance of the raw word embeddings, standardization is critical for domain adaptation, which needs to generalize to many words unseen in training.

3.5 Evaluation

3.5.1 Data Analysis

The OVERNIGHT dataset [58] contains 8 different domains. Each domain is based on a separate knowledge base, with logical forms written in λ-DCS [43]. Logical forms are converted into canonical utterances via a simple grammar, and the input utterances are collected by asking crowd workers to paraphrase the canonical utterances. Different domains are designed to stress different types of linguistic phenomena. For example, the CALENDAR domain requires a semantic parser to handle temporal language like "*meetings that start after 10 am*", while the BLOCKS domain features spatial language like "*which block is above block 1*".

Vocabularies vary remarkably across domains (Table 3.2). For each domain, only 45% to 70% of the words are covered by any of the other 7 domains. A model has to learn the out-of-vocabulary words from scratch using in-domain training data. Pre-trained word embeddings cover most of the words of each domain, and thus can connect the domains to facilitate domain adaptation. Words that are still missing are mainly stop words and typos, e.g., "*ealiest*".

3.5.2 Experiment Setup

We compare our model with all the previous methods evaluated on the OVERNIGHT dataset. In the original OVERNIGHT paper, Wang et al. [58] use a log-linear model with a rich set of features, including paraphrase features derived from PPDB [88], to rank logical forms. Xiao et al. [87] use a multi-layer perceptron to encode the unigrams and bigrams of the input utterance, and then use a RNN to predict the derivation sequence of a logical form under a grammar. Jia and Liang [67] also use a Seq2Seq model with bi-directional RNN encoder and attentive decoder, but are to predict linearized logical forms. They also propose a data augmentation technique, which further improves the average accuracy to 77.5%. But it is orthogonal to this

work and can be incorporated in any model including ours, therefore not included.

The above methods are all based on the in-domain setting, where a separate parser is trained for each domain. In parallel of this work, Herzig and Berant [66] have explored another direction of cross-domain training: they use all of the domains to train a single parser, with a special domain encoding to help differentiate between domains. We instead model it as a domain adaptation problem, where the source and the target domain training are separate. Their model is the same as Jia and Liang [67]. It is the current best-performing method on the OVERNIGHT dataset.

We use the standard 80%/20% split of training and testing, and randomly hold out 20% of training for validation. Hyper-parameters are selected based on the validation set. State size of both the encoder and the decoder are set to 100, and word embedding size is set to 300. Input dropout rate of the GRU cells is 0.7, and output dropout rate is 0.5. Mini-batch size is 512. We use Adam for optimization, which we find works slightly but consistently better than other popular optimizers like RMSprop and Adadelta. We use the default parameters for Adam as suggested in the paper. We use gradient clipping with a cap for global norm at 5.0 to alleviate the exploding gradients problem of recurrent neural networks. Early stopping based on the validation set is used to decide when to stop training. For each experiment, we do the training for 3 times and then test the model with the best validation performance. In cross-domain experiments, for each target domain, all the other domains are combined as the source domain. The evaluation metric is accuracy, i.e., the proportion of testing examples for which the top prediction yields the correct denotation. Our model is implemented in Tensorflow [89], and the code is available at `https://github.com/ysu1989/CrossSemparse`.

3.5.3 Experiment Results

Comparison with Previous Methods

The main experiment results are shown in Table 3.3. Our base model (Random + I) achieves an accuracy comparable to the previous best in-domain model [67]. With our main novelties, cross-domain training and word embedding standardization, our full model is able to outperform the previous best model, and achieve the best accuracy on 6 out of the 8 domains. Next we examine the novelties separately.

Word Embedding Initialization

The in-domain results clearly show the sensitivity of model performance to word embedding initialization. Directly using the raw WORD2VEC embeddings or with per-example normalization, the performance is significantly worse than random initialization (6.2% and 7.3%, respectively). Based on the previous analyses, however, one should not be too surprised. The small micro variance problem hurts optimization. In sharp contrast, both of the proposed standardization techniques lead to better in-domain performance than random initialization (1.4% and 2.5%, respectively), setting a new best in-domain accuracy on OVERNIGHT.

Cross-domain Training

A consistent improvement from cross-domain training is observed across all word embedding initialization strategies. Even for raw WORD2VEC embeddings or per-example normalization, cross-domain training is able to alleviate the poor initialization. The best results are again obtained with standardization. As we have discussed before, per-feature standardization does not resolve the large macro variance problem, which may make a model harder to generalize to words unseen in training. The results provide an empirical evidence for this hypothesis. The gain of cross-domain training is more significant with per-example standardization (2.4%)

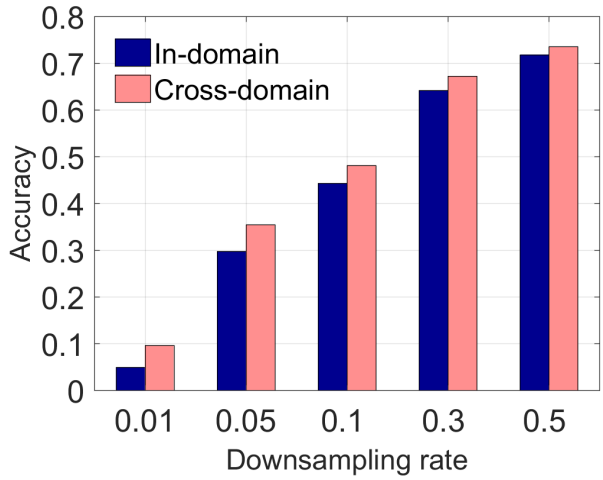


Figure 3.2: Results with downsampled in-domain training data. The experiment with each downsampling rate is repeated for 3 times and average results are reported. For simplicity, we only report the average accuracy over all domains. Pre-trained word embeddings with per-example standardization is used for both methods.

than with per-feature standardization (1.8%). The seemingly radical per-example standardization works best for domain adaptation.

Using Downsampled Training Data

Considering the number of logical forms, the in-domain training data in the OVERNIGHT dataset is indeed abundant. In cross-domain semantic parsing, we are more interested in the scenario where there is insufficient training data for the target domain. To emulate this scenario, we downsample the in-domain training data of each target domain, but still use all training data from the source domain (thus $N_t \ll N_s$). The results are shown in Figure 3.2. The gain of cross-domain training is most significant when in-domain training data is scarce. As we collect more in-domain training data, the gain becomes smaller, which is expected.

3.6 Conclusion

We proposed a paraphrasing based framework for cross-domain semantic parsing. With a sequence-to-sequence paraphrase model, we showed that cross-domain training of semantic parsing can be quite effective. We also studied how to properly standardize pre-trained word embeddings in neural networks, especially for domain adaptation.

This work opens up a number of future directions. As discussed in Section 3.2.3, many conventional domain adaptation and representation learning ideas can find application in cross-domain semantic parsing. In addition to pre-trained word embeddings, other language resources like paraphrase corpora [88] can be incorporated into the paraphrase model to further facilitate domain adaptation. We have restrained ourselves to the case where domains are defined using the same formal language, and we look forward to evaluating the framework on domains of different formal languages when such datasets with canonical utterances become available.

Chapter 4

Interactive Natural Language Interfaces

4.1 Introduction

With the meteoric growth of the digital world and the popularization of computing devices like smartphones and Internet-of-Things (IoT) devices among less technically proficient people, new ways of human-computer interfacing are in great demand. Natural language (NL) is the most common communication method used by humans. Not surprisingly, natural language interfaces (NLIs) have been an aspirational goal in human-computer interaction since the very early days of digital computers [3]. They bear the promise of providing a unified interface for even technically non-proficient users to access a wide range of heterogeneous data, services, and devices.

The core challenge of natural language interfaces is to map natural language utterances (commands) from users to some formal meaning representation, be it SQL for relational databases, SPARQL for knowledge bases, or API (application program interface) for software applications, that is understandable by computers. Recent advances in deep learning make it possible to develop generic natural language interfaces that are free of feature engineering and can more easily generalize to different domains. As a result, we have recently witnessed a growth in

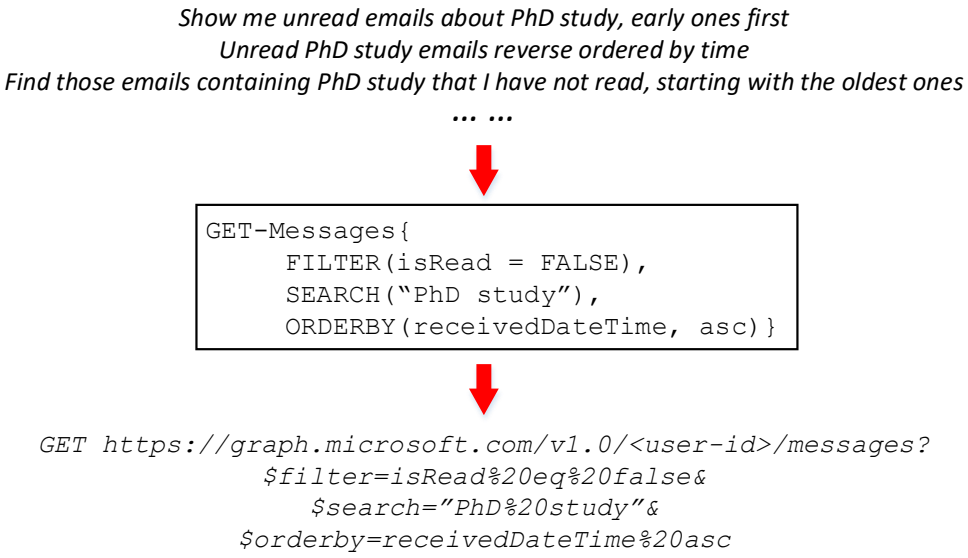


Figure 4.1: Example of natural language interface to web API. *Top*: Natural language utterances (commands). *Middle*: API frame. An abstract representation that can be converted into the real API call deterministically. *Bottom*: Real API call to the Microsoft email search API.

neural network based natural language interfaces to a wide range of data types such as knowledge bases [9, 90], relational database-like tables [91, 17, 92], and APIs to web services and Internet-of-Things devices [93, 18].

One of the main challenges facing natural language interfaces is that natural language is inherently ambiguous. Hence, it is unrealistic to expect a natural language interface to perfectly understand all natural language commands. Additionally, it is difficult for a user to assess the results and decide whether or not the model was able to correctly interpret their commands. Even when they can do that, in case of erroneous results their only resort is to reformulate their command and try again. This is especially true with mainstream neural network models, which provide little insights to help users interpret the predictions made by the model.

In this paper, we study *interactive* natural language interfaces, which allow users to interact with the system and correct possible errors. In particular, we hypothesize that the support of fine-grained user interaction can greatly improve the usability of natural language interfaces. To test this hypothesis, we conduct a case study in the context of natural language interfaces to

web APIs (NL2API). An example of NL2API can be found in Figure 4.1.

The mainstream neural network model for natural language interfaces is the sequence-to-sequence model [63]. However, it is difficult to create interactive natural language interfaces with the vanilla sequence-to-sequence model. To facilitate our case study on interactive natural language interfaces, we propose a novel modular sequence-to-sequence model. The main idea is to decompose the complex prediction process of a typical sequence-to-sequence model into small prediction units called *modules*. Each module is highly specialized at predicting a pre-defined kind of sequence output, and their prediction can therefore be easily explained to the user. The user can then verify the correctness of the prediction of each module, and give feedback to correct possible errors in the module predictions. For every specific command only a few modules will be triggered, and a specifically designed controller will read the input command to decide which modules to trigger. Both the controller and the modules are neural networks. We further propose an interaction mechanism based on the proposed model.

To test the hypothesis on interactive natural language interfaces, we design both simulation and human subject experiments with two deployed Microsoft APIs, which are used for searching emails and calendar events, respectively. In the simulation experiment, we show that the interactive NLI can greatly improve the prediction accuracy via only a small amount of extra user effort: with only one round of user interaction, testing accuracy can be improved from around 0.5 to over 0.9. In the human-subject experiment, we conduct a comparative study. We compare the interactive NLI with its non-interactive counterpart, which is similar to a traditional search engine: If the model prediction is incorrect, a user will reformulate the command and try again. Through log-based analysis and user survey, we find that the interactive NLI outperforms the non-interactive NLI on a variety of measures: The interactive NLI leads to higher task success rate, shorter task completion time (less user effort), and remarkably higher user satisfaction. 85% of the participants indicate that they prefer the interactive NLI over the non-interactive NLI.

In summary, this work makes major contributions in problem formulation, model, and experimentation:

- We conduct a systematic study on fine-grained user interaction in natural language interfaces with a focus on web APIs.
- We propose a novel modular sequence-to-sequence model to facilitate the creation of interactive natural language interfaces.
- We design both simulation and human subject experiments with real-world APIs to demonstrate the benefits of interactive natural language interface along several dimensions including task completion, user effort, and user satisfaction.

4.2 Natural Language Interface to Web API

A web API is a set of operations, associated data definitions, and the semantics of the operations for accessing a Web-based software application. Web APIs provide the foundations for interacting with applications such as email and calendar, customer relation management [94], photo sharing services, social media platforms, online shopping, and the Internet-of-Things [95]. NL2API enables users to access a wide range of applications in a unified, natural way, while staying agnostic to the heterogeneity of data and services that they must handle when using traditional graphical user interfaces (e.g., learn and adapt to different graphical user interfaces to use different applications). As a result, NL2APIs have attracted increased attention in recent times [96, 93, 18].

The core task of NL2API is to map natural language utterances given by users into API calls. More specifically, we will follow the setting defined by Su et al.[18] and focus on web APIs that follow the REST architectural style [97], i.e., *RESTful APIs*. RESTful APIs are widely used for web services [98], IoT devices [95], as well as smartphone apps [99]. An

Parameter Type	Description
`SEARCH(String)`	Search for resources containing specific keywords
`FILTER(BoolExpr)`	Filter resources by some criteria, e.g., `isRead=False`
`ORDERBY(Property,Order)`	Sort resources on a property in 'asc' or 'desc' order
`SELECT(Property)`	Instead of full resources, only return a certain property
`COUNT()`	Count the number of matched resources
`Top(Integer)`	Only return the first certain number of results

Table 4.1: API parameter types.

example from [18] based on the Microsoft email search API[1] is shown in Figure 4.1. The top portion of the figure shows multiple natural language utterances. The same user intent can be expressed in syntactically-divergent ways in natural language, i.e., paraphrases, which should all be mapped to the same API call. The middle portion shows an API frame; which represents a more compact representation of RESTful API calls defined in [18], and can be mapped to the real API calls in a deterministic way. The bottom portion shows a real API call. It contains many irrelevant constituents such as URL conventions that could be distracting in natural language interfaces. We will use API frame in the following, and will use API frame and API call interchangeably.

A RESTful API (e.g., `GET-Messages`) consists of an HTTP verb (e.g., `GET`, `PUT`, and `POST`) and a set of resources (e.g., a user's emails). In addition, one can call an API with different parameters to specify advanced search requests, for example, filter by some properties of the resource (e.g., `subject`, `isRead`, `receivedDateTime` of an email) or search for some keywords. The full list of parameter types can be found in Table 4.1. An *API call* is an API with a list of parameters. It can be linearized into a sequence (Figure 4.1 middle).

Natural language interface to Web API Given an input utterance $\boldsymbol{x} = \{x_1, x_2, \ldots, x_m\}$, the task of a natural language interface to web API is to map $\boldsymbol{x}$ to the corresponding linearized API

[1] https://developer.microsoft.com/en-us/graph/

call $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$.

4.3 Interactive Natural Language Interface

In this section, we discuss the different levels of user interaction that a natural language interface may support, and propose a modular sequence-to-sequence model which naturally supports user interaction at the fine-grained parameter level.

4.3.1 User Interaction

NL2API maps a command to an API call, which can be executed and return the results to the user. Correspondingly, it is possible to enable interaction and solicit feedback from users at three levels: (1) Result level, by asking users to verify result correctness; (2) API call level, by asking users to verify API call correctness, and; (3) Parameter level, by asking users to interact with each parameter in the predicted API call.

The most straightforward way to interact is to execute the command and ask users to judge the correctness of the returned results. However, this approach has two problems. First, it is not always possible for a user to easily verify result correctness. If a user asked "*how many provinces are there in China?*" and a system said "*23*", how could the user know that the system's understanding is not "*the 9th prime number*" or "*the atomic number of vanadium*?" Second, the information provided by result correctness may be limited. If a user indicates that the provided results are incorrect, how much help does this new information provide to the system to select the correct API call from possibly thousands of candidates?

Alternatively, we can ask users to verify the correctness of the predicted API call. Such information is more definitive than result correctness. Although it may be difficult for general users to directly understand API calls, it is possible to design some rules to automatically convert API calls into natural language utterances (e.g., [18]), which can be easily understood.

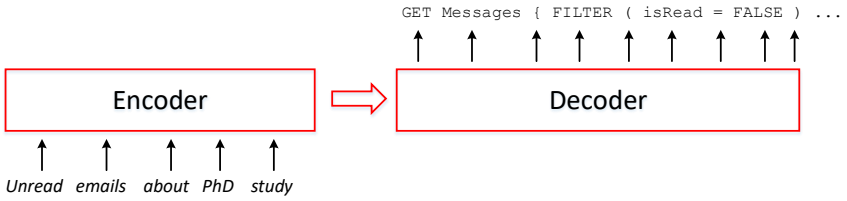


Figure 4.2: Vanilla sequence-to-sequence model for NL2API. In practice, constructs like bi-directional RNN encoder and attention mechanisms (see definitions below) are usually added to the vanilla model for better performance.

However, similar to result-level interaction, there is still the challenge of how to use this new information and how much help it can bring. It is not efficient if a user needs to decline tens of incorrect API calls before obtaining the correct one.

We believe it is more helpful if users can interact with the natural language interface at a finer-grained parameter level. For the example in Figure 4.1, if the natural language interface incorrectly predicts a parameter `FILTER(isRead = TRUE)`, the user may interact with the system and indicate that the parameter value should be changed to `FALSE`. Next, we will first review the mainstream sequence-to-sequence model for natural language interfaces. We then propose a modular sequence-to-sequence model, which naturally supports parameter-level interaction.

4.3.2 Sequence-to-Sequence Model

The core task of natural language interfaces, including NL2APIs, can often be cast into a sequence to sequence prediction problem: utterance sequence as input, and formal meaning representation sequence as output. The sequence-to-sequence (Seq2Seq) neural model [63] is a natural choice for this task, and has been widely used for natural language interfaces to knowledge bases [67, 100], relational databases [92], and web APIs [18]. Since we will use the Seq2Seq model as a building block in the modular Seq2Seq model, we first give its formal definition.

For an input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$, the Seq2Seq model estimates the conditional probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$ for all possible output sequences $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$. The lengths m and n can be different, and both of them can be varied. An illustrative example is shown in Figure 4.2.

The *encoder*, which is implemented as a bi-directional recurrent neural network (RNN), first encodes $\boldsymbol{x}$ into a sequence of state vectors $(h_1, h_2, \ldots, h_m)$. Suppose ϕ is a randomly initialized word embedding layer that embeds every word into a low-dimensional vector, the state vectors of the forward RNN and the backward RNN are respectively computed as:

$$\begin{aligned} \overrightarrow{h}_i &= GRU_{fw}(\phi(x_i), \overrightarrow{h}_{i-1}) \\ \overleftarrow{h}_i &= GRU_{bw}(\phi(x_i), \overleftarrow{h}_{i+1}) \end{aligned} \tag{4.1}$$

where gated recurrent unit (GRU) as defined in [79] is used as the recurrence. We then concatenate the forward and backward state vectors, $h_i = [\overrightarrow{h}_i, \overleftarrow{h}_i], i = 1, \ldots, m$.

We use an attentive RNN as the *decoder*, which will generate the output tokens one at a time. We denote the state vectors of the decoder RNN as $(d_1, d_2, \ldots, d_n)$. The attention takes a form similar to [80] (also known as *additive attention*). For the decoding step j, the decoder is defined as follows:

$$\begin{aligned} d_0 &= \tanh(W_0[\overrightarrow{h}_m, \overleftarrow{h}_1]) \\ u_{ji} &= v^T \tanh(W_1 h_i + W_2 d_j) \\ \alpha_{ji} &= \frac{u_{ji}}{\sum_{i'=1}^{m} u_{ji'}} \\ h'_j &= \sum_{i=1}^{m} \alpha_{ji} h_i \\ d_{j+1} &= GRU([\phi(y_j), h'_j], d_j) \\ p(y_j|\boldsymbol{x}, y_{1:j-1}) &\propto \exp(U[d_j, h'_j]) \end{aligned} \tag{4.2}$$

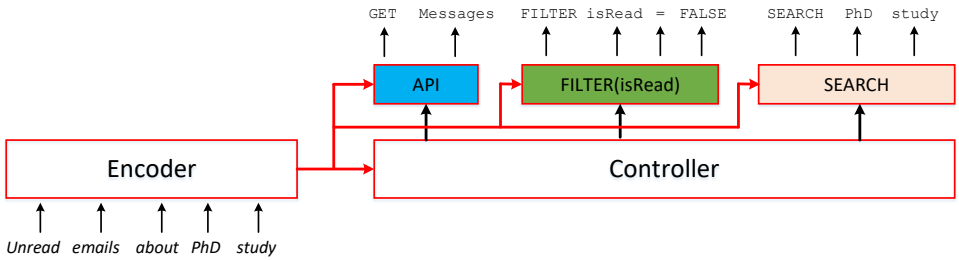


Figure 4.3: Modular sequence-to-sequence model. The controller triggers a few modules, each of which instantiates a parameter.

where W_0, W_1, W_2, v and U are model parameters. The decoder first calculates normalized attention weights α_{ji} over encoder states, and get a summary state h'_j. The summary state is then used to calculate the next decoder state d_{j+1} and the output probability distribution $p(y_j|\boldsymbol{x}, y_{1:j-1})$. During training, the sequence $y_{1:j-1}$ is supplied using the gold output sequence; during testing, it is generated by the decoder.

4.3.3 Modular Sequence-to-Sequence Model

We propose a novel modular sequence-to-sequence model (Figure 4.3) to enable fine-grained interaction of natural language interfaces. To achieve that, we decompose the decoder in the original Seq2Seq model into multiple interpretable components called *modules*. Each module is specialized at predicting a pre-defined kind of output, e.g., instantiating a specific parameter by reading the input utterance in NL2API. After some simple mapping, users can easily understand the prediction of any module, and interact with the system at the module level. It is similar in spirit to modular neural networks [101, 102, 103]. But to the best of our knowledge, this is the first work to study interactive natural language interfaces with modular neural networks. Also, different from previous modular neural networks, each module in our model generates a sequential output instead of a continuous state.

Module. We first define modules. A module is a specialized neural network, which is designed to fulfill a specific sequence prediction task. In NL2API, different modules corre-

spond to different parameters. For example, for the `GET-Messages` API the modules are `FILTER(sender)`, `FILTER(isRead)`, `SELECT(attachments)`, `ORDERBY(receivedDateTime)`, `SEARCH`, etc. The task of a module, if triggered, is to read the input utterance and instantiate a full parameter. To do that, a module needs to determine its parameter values based on the input utterance. For example, given an input utterance "*unread emails about PhD study*", the `SEARCH` module needs to predict that the value of the `SEARCH` parameter is "*PhD study*", and generate the full parameter, "`SEARCH PhD study`", as its output sequence. Similarly, the `FILTER(isRead)` module needs to learn that phrases such as "*unread emails*", "*emails that have not been read*", and "*emails not read yet*" all indicate its parameter value is `False`.

It is natural to implement the modules as attentive decoders, similar to the original Seq2Seq model. However, instead of a single decoder for everything, now we have multiple decoders each of which is specialized in predicting a single parameter. Moreover, as we will show in Section 4.4, because each module has clearly defined semantics, it becomes straightforward to enable user interaction at the module level. Formally, a module M_k is an attentive decoder as defined in Eq (4.2), with the goal to estimate the conditional probability distribution $p_k(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{y}$ is from the set of API frame symbols.

Controller. For any input utterance, only a few modules will be triggered. It is the job of the controller to determine which modules to trigger. Specifically, the controller is also implemented as an attentive decoder. Using the encoding of the utterance as input, it generates a sequence of modules, called the *layout*. The modules then generate their respective parameters, and finally the parameters are composed to form the final API call. Formally, the controller is an attentive decoder as defined in Eq (4.2), with the goal to estimate the conditional probability distribution $p_c(\boldsymbol{l}|\boldsymbol{x})$, where the layout $\boldsymbol{l}$ is from the set of modules.

Example. Take Figure 4.3 as example. The controller first reads the input utterance and

generates a sequence of modules, `API`, `FILTER(isRead)`, and `SEARCH`. Each module then reads the input utterance again to generate their respective parameter, where the main work is to determine the correct parameter values based on the utterance.

Training Objective. Given a set of training examples $\{(\boldsymbol{x}_i, \boldsymbol{l}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, the loss function of the whole modular Seq2Seq model consists of three kinds of losses:

$$\Theta = \frac{1}{N}\sum_{i=1}^{N}(\Theta_{c,i} + \Theta_{m,i}) + \lambda\Theta_{L2}. \tag{4.3}$$

For the i-th example, the controller loss is a cross-entropy loss on the layout prediction:

$$\Theta_{c,i} = -\log p_c(\boldsymbol{l}_i|\boldsymbol{x}_i). \tag{4.4}$$

Suppose the gold layout of the i-th example $\boldsymbol{l}_i = \{M_1, M_2, \ldots, M_t\}$ with respective gold parameters $\{\boldsymbol{y}_{i,1}, \boldsymbol{y}_{i,2}, \ldots, \boldsymbol{y}_{i,t}\}$, the module loss is the average cross-entropy loss on the module predictions:

$$\Theta_{m,i} = -\frac{1}{t}\sum_{j=1}^{t}\log p_j(\boldsymbol{y}_{i,j}|\boldsymbol{x}_i). \tag{4.5}$$

Finally, we add an L2 regularization term Θ_{L2} with balance parameter λ to alleviate overfitting. We also apply dropout [81] on both the input and the output of GRU cells to alleviate overfitting.

4.4 Interaction Mechanism

In this section we present our interaction mechanism based on the proposed modular Seq2Seq model.

Interpretable module output. The output of each module can be easily explained to the user.

Parameter Syntax	Natural Language Explanation
`FILTER isRead = BOOLEAN`	is (not) read
`SEARCH KEYWORDS`	contains keyword KEYWORDS
`SELECT receivedDateTime`	return the receive time

Table 4.2: Example mapping of module output to natural language explanation. A few rules suffice for the mapping.

Because each module is highly specialized at predicting one pre-defined parameter, its output highly conforms to the syntax of that parameter. For example, for the `FILTER(isRead)` module, the parameter syntax is "`FILTER isRead = BOOLEAN`", where `BOOLEAN` is either `TRUE` or `FALSE`. Similarly, for the `SEARCH` module, the parameter syntax is "`SEARCH KEYWORDS`", where `KEYWORDS` is a sequence of keywords. Therefore, it is easy to use a simple rule to map the output of a module to a natural language phrase that is understandable by general users. Several examples are shown in Table 4.2.

Parameter value suggestion. Since the modules are neural decoders, each of them can generate a ranked list of outputs. For example, for the input utterance "*unread emails about PhD study*", the `SEARCH` module may generate the following list:

1. `SEARCH PhD`
2. `SEARCH PhD Study`
3. `SEARCH PhD study emails`

Therefore, in addition to the top-ranked output, we can present to the user several plausible suggestions (mapped to natural language explanations as in Table 4.2). If the top-ranked output is incorrect, the user may find the correct one in the suggestion list[2].

[2]The output space of a module is much smaller than the whole API call space, which makes the suggestion task easier.

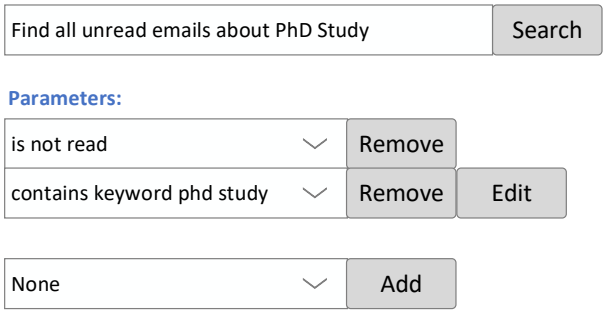


Figure 4.4: Interactive natural language interface. Once the user types in the command and clicks "Search," the system will generate the most probable API call from the modular Seq2Seq model, convert the output of each module into natural language, and show the results to the user. The user can then interact with the system using a number of operations such as adding or removing modules, selecting alternative parameter values from drop-down menus, or editing parameter values.

Module suggestion. Sometimes the controller makes a mistake when predicting the layout and misses some module. We also provide a list of module suggestions and allow the user to add modules from the list. Currently we run all the modules of an API and include the top-ranked output in the suggestion list. One can also only keep a few most probable ones to reduce the number of suggestions.

Module removal. Similarly, the controller may make a mistake when predicting the layout and adds an unnecessary module. To address this, we allow the user to remove modules from the list. Currently, we allow the user to remove any module from the list returned by the model.

We design a graphical user interface (Figure 4.4) to accommodate all the above interaction components. The user is initially shown a query box where she can type her query and click search. Given an utterance, our model will come up with the most likely interpretation of the utterance and show it to the user. Additionally, a drop-down menu is shown corresponding to each module in the interpretation. For example, the utterance *"find all unread emails about PhD study"* shown in Figure 4.4 will result in the following API call: `GET-Messages{FILTER(isRead = FALSE), SEARCH(''PhD study'')}`. Hence, the interface will show the two modules for filtering based on `isRead` and searching. If any

of the module output is incorrect, the user can click on the module output to select from a list of suggestions in a drop-down menu. In rare cases, the user can also click the "edit" button to input the desired parameter value. Finally, the user can also remove a module completely, or add a module from a drop-down list if some desired modules are missing.

It is worth noting that the interaction mechanism can also be implemented based on natural language communication instead of display and click in a graphical user interface. We have opted for a graphical user interface mainly because it naturally leads to a compact interface to accommodate all interaction components as in Figure 4.4, and allows for more efficient user interaction.

4.5 Evaluation

In this section we experimentally evaluate the proposed modular Seq2Seq model and the interaction mechanism. The main goal is to test the hypothesis that *fine-grained user interaction can greatly improve the usability of natural language interfaces*. We carry out the study in two experimental settings: (1) Using a *simulated* user on a standard NL2API dataset, we show that the interaction mechanism can significantly improve the accuracy of NL2API, with only a small number of interactions. (2) Through a *human* user experiment, we show that an interactive natural language interface, compared with its non-interactive counterpart, leads to higher success rate, less user effort, and higher user satisfaction.

While the main goal is to study fine-grained user interaction, We also compare several models in a non-interactive experiment that performs a traditional evaluation over held-out test data. The goal is to show that modular Seq2Seq model can achieve competitive performance in comparison with other models, to support its use as the base model for the subsequent study on interactive natural language interfaces.

API	Training	Validation	Testing
`GET-Messages`	3670	917	157
`GET-Events`	5036	1259	190

Table 4.3: Dataset statistics.

4.5.1 Experimental Setup

Dataset. We use the NL2API dataset released in [18] to train our model. It contains utterance-API call pairs for two deployed Microsoft APIs respectively for searching a user's emails (`GET-Messages`) and calendar events (`GET-Events`). The dataset was collected via crowdsourcing, and is split into a training set and a testing set. The training set contains some noise from crowdsourcing, while the testing set is smaller but each example is manually checked for quality. For model selection purpose we further hold out 20% of the training data to form a validation set, and use the rest for training. The statistics can be found in Table 4.3. For the modular Seq2Seq model, there are 19 modules for each API.

This is a challenging dataset. A good portion of the testing set (close to 40%) involves API calls that are more complex than those covered by the training set (larger number of parameters than ever seen in the training set). It is designed to test model generalizability on more complex and unseen API calls. Also, because of the flexibility of natural language, the same API call can be represented using different natural language utterances, i.e., paraphrases. So even if an API call is covered by the training set with several utterances, the utterances in the testing set are still unseen in training. A good natural language interface therefore needs to be able to generalize to both unseen API calls and unseen utterances for covered API calls.

Measures. For the non-interactive experiment (Section 4.5.2) and the simulation experiment (Section 4.5.3), following the literature [18, 67], we use accuracy as the evaluation measure. It is the proportion of testing examples for which the top API call generated by the model

Model/API	GET-Messages	GET-Events
Su et al. [18]	0.573	0.453
Seq2Seq	0.586	0.453
Modular Seq2Seq	0.599	0.453

Table 4.4: Model accuracy in the non-interactive experiment. Su et al. [18] use a vanilla Seq2Seq model for ranking API calls. The Seq2Seq model (second row) is the one with bi-directional RNN encoder and attentive decoder as defined in Section 4.3.2. Modular Seq2Seq model is the proposed model as defined in Section 4.3.3. Both of these models directly generate an API call as output. For GET-Events, the three models happen to make the same number of errors on the second test set, but on different examples.

exactly matches the correct API call. For the human subject experiment (Section 4.5.4), we use a variety of measure such as task success rate, completion time, and user satisfaction (more details later).

Implementation details. We implement the proposed modular Seq2Seq model in Tensorflow [89]. The Tensorflow Fold [104] library is employed to dynamically build the computation graph according to the layout prediction from the controller. We use Adam [105] as the optimizer. Hyper-parameters of the model are selected based on the validation set. State size of the encoder is 100, and state size of all the decoders, including the controller and the modules, are 200. The word embedding size is 300 for the encoder, and 50 for the decoders since their vocabulary is smaller. Input and output dropout rate of the GRU cells are 0.3 and 0.5, respectively. The balance parameter for L2 regularization is 0.001. We use a large mini-batch size. 2048, to fully take advantage of the dynamic batching [104], which significantly improves training speed. Early stopping based on the validation set is used.

4.5.2 Non-interactive Experiment

We first evaluate the modular Seq2Seq model in a non-interactive setting, where there is no user interaction involved. The goals are two-fold. First, through error analysis we can get

additional insights into the challenge of NL2API. Second, we show that the modular Seq2Seq can achieve competitive performance compared with other alternatives, which supports its use as the basis for an interactive natural language interface.

The testing accuracies on the NL2API dataset are shown in Table 4.4. Each model is trained on the training set and evaluated on the testing set. As can be seen, the modular Seq2Seq model achieves comparable performance with other models.

We present an error analysis of the modular Seq2Seq model. The prediction of the model can have three types of errors, two from the controller, i.e., having extra modules or missing required modules in the predicted layout, and one from the modules, i.e., having incorrect prediction of parameter values (e.g., return read emails while the user wants to find unread emails). For `GET-Messages`, 87.3% of the error cases have missing modules, 25.4% have extra modules, and 9.5% have erroneous parameter values. For `GET-Events`, 77.9% of the error cases have missing modules, 23.1% have extra modules, and 8.6% have erroneous parameter value. Note that some error cases involve more than one type of errors. Therefore, most of the errors come from the controller. A promising future direction is to develop more advanced models for the controller. One possible way is to allow the controller to access the module states in addition to the input utterance, so that it knows which parts in the input utterance have been processed by which modules, and which parts are left unprocessed that may need some additional modules.

The current best accuracy is not sufficient for a practical natural language interface in real use: it will fail on roughly one half of the user commands. However, it should be noted that accuracy is a strict binary measure: A model is correct on a testing example only if the predicted API call exactly matches the correct one; otherwise, it gets zero score. But most of the time the predicted API calls are very close to the correct API calls, only missing one module or getting a parameter value slightly wrong. If users can interact with the model to correct such errors, the model accuracy can be greatly improved. With the original Seq2Seq model, it is

difficult for users to correct possible errors. The modular Seq2Seq model makes it easier for users to understand model prediction, and interact with the model at the fine-grained module level to correct errors. In the next two experiments, we show the effectiveness of the interaction mechanism with both simulated users and real human subjects.

4.5.3 Simulation Experiment

Because the dataset contains the correct API call for each testing example, we can use it to simulate a human user to interact with the UI in Figure 4.4. Given a testing example, it first issues the utterance as input to the model. After obtaining the model prediction, the simulated user will use the interaction actions introduced in Section 4.4 to correct possible errors until the prediction matches the correct API call. We record the number of actions taken in this procedure. More specifically,

Behavior. At the beginning of a task, the simulated user has an utterance and the correct API call. It issues the utterance to the search box in Figure 4.4. After getting the initial model prediction, it will try to match the prediction with the correct API call, and if there are mismatches, it will carry out necessary actions to correct the mismatches in the following order: (1) If there are modules missing from the correct API call, add from the module list. (2) If there are extra modules not in the correct API call, remove the modules. (3) If there are modules with erroneous parameter value, first try to select from the drop-down suggestion list. If the correct parameter value is absent from the suggestion list, click the "edit" button and type in the correct parameter value.

Example. Suppose the utterance is "*unread emails about PhD study*" and the correct API call consists of two parameters, "`FILTER isRead = FALSE`" and "`SEARCH PhD study`", and the initial model prediction has three parameters, "`FILTER isRead = FALSE`", "`SEARCH PhD`", and "`SELECT attachments`". The simulated user will first remove the `SELECT` param-

eter because it knows this one is not in the correct API call. Then the simulated user will change the value of the `SEARCH` parameter from "`PhD`" to "`PhD study`" by selecting from the drop-down suggestion list. In total it takes two actions to convert the initial model prediction to the correct API call.

The experiment results are shown in Figure 4.5. When no interaction is involved (# of actions = 0), the model achieves the same accuracy as in the non-interactive experiment (Table 4.4). A small amount of user interaction can greatly improve the accuracy. Most remarkably, *with only one action from the simulated user, the accuracy can be improved to around 0.92 for both APIs*. This shows that most of the time the initial model prediction is quite reasonable, only one step away from the correct API call. However, this does not necessarily mean that one can easily develop a better model to do this last step without user interaction. One difficulty is that some utterances are inherently ambiguous and the correct interpretation depends on the specific user or context. For example, with the same utterance "*find the first unread email*", some users may mean the earliest one, while some other users may mean the last one. User interaction may be necessary to resolve such ambiguities and improve personalization and context awareness. In summary, the simulation experiment results show that the designed interactive NLI can lead to remarkably better accuracy with only a small amount of user interaction.

4.5.4 Human Subject Experiment

Study Methodology: To better understand the impact of the interactive and the standard approaches for NL2API on the user experience, we conducted a lab study using the web-based interface described earlier for both the standard and interactive modes. Both modes are based on the same trained modular Seq2Seq model. The only difference is that the standard mode does not allow user interaction. The study used within-subject design with the interaction mode

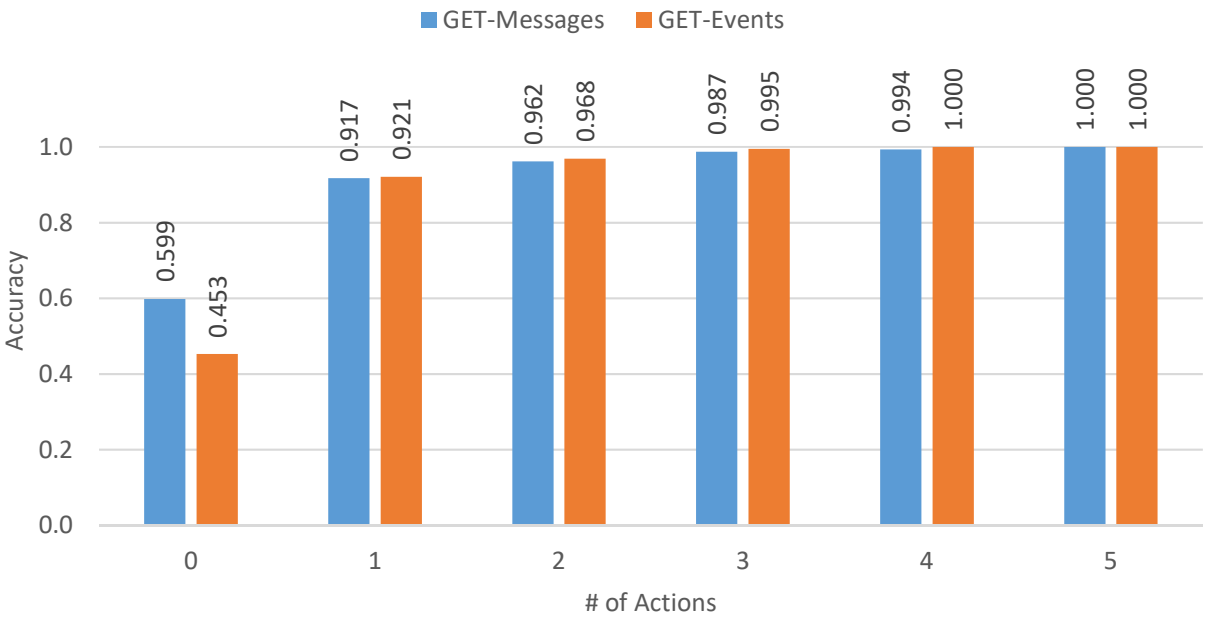


Figure 4.5: Simulation experiment results.

as the factor.

For the standard interaction mode, the user issues a query and gets the results back. The user examines the results and then decides if they satisfy her need or not. If they do, she stops. Otherwise, she may decide to try again by reformulating the query or give up. For the interactive mode, the user gets to interact with the results using the UX controls shown in Figure 4.4. For example, if the user decides that the keyword in the keyword filter should be changed, she may simply edit the filter. Similarly if she decides that the results should be ordered by the received time, she may select to add such a filter. The suggestions for adding, removing or editing the filters are provided by the model using the hypothesis space it builds as it interprets the natural language command.

Participants: Twenty people participated in the study. Participants were recruited via email advertising to a group of people affiliated with a large university in the US. Most participants were students from various backgrounds with ages ranging between 20 and 30 years old. All participants were experienced with using search engines and intelligent assistants such as Siri,

Task Description	Difficulty
List unread messages	Easy
Find emails with high priority about 'PhD Study'	Medium
Find unread emails from John Smith with early ones first	Hard
Find the attachment of the most recent email in the Red category	Very Hard

Table 4.5: Task examples.

Cortana or Alexa.

Protocol: Upon starting, participants were given an overview of the study. To familiarize themselves with the system, they were given 6 experimental trail tasks (3 for each interaction mode). Data from the trial tasks were not used for the results of this study. After completing the experimental trails, participants were given 10 tasks (5 for each mode), resulting a total of 200 tasks. The order of the tasks and which interaction mode they belong to was randomized for each participant. Examples of the task are shown in Table 4.5. Each task was assigned a difficulty level (based on the number of parameters in the target API call). Tasks across the two interaction modes had balanced difficulty level. To encourage participants to come up with their own formulation of the query text, we showed them the task description in another language (we used Chinese and recruited participants that are fluent in both English and Chinese). Previous work has used similar techniques such as giving participants task descriptions in a different language [106] or in a recorded voice message [107].

After completing all the tasks, participants were asked to complete a questionnaire about the system they preferred and they were also asked to provide general feedback about the system. Participants also answered questions about their background and familiarity with the use of search and intelligent assistants.

Measures: Our overarching research question is: *what are the costs and benefits of the interactive NL2API compared to the standard search engine-like approach?* To answer this question, we used a combination of log-based analysis and survey questions. We implemented a rich

instrumentation that records all interactions between the participants and the system. For example, all queries, clicks, query reformulation, etc. were logged using an event-based schema that also recorded the time stamp of the event, a task id and an anonymized study id. We also collected answers to survey questions after the experiment and linked it to the same study id. We describe more details of our measures as follows:

Task Completion: To study the effect of the interaction mode on the task completion rate, we measured the outcome of the completion of each task. Since the target result was known a priori, participants get feedback about whether the system was able to retrieve the correct answer or not. A task is considered successfully completed, only when the system is able to generate the interpretation that would retrieve the correct answer. Note that the participants were given feedback about whether the model got a task correct or not. In a real scenario, the users would be retrieving their own emails, appointments, etc. and they can decide whether the current answer satisfied their need or not. If the user gives up without getting the correct result, the task is considered as not successfully completed.

Effort: We also wanted to study the effort needed to achieve success in each interaction mode. We do that by measuring the total number of actions (e.g. queries, clicks, etc.) and the time to completion from the start to the end of each task.

User Satisfaction: Finally, we assessed the overall user satisfaction with the two interaction modes. We asked users to assess their satisfaction with both systems and to assess their relative preference between the two modes using a 5-point Likert scale.

Results:

Task Completion: The top portion of Figure 4.6 compares the success rate for the standard and interactive modes. Interactive mode helped participants complete tasks successfully at a higher rate than the standard mode. It was particularly helpful with harder tasks where the model is more likely to make mistakes in translating the natural language command to the correct API call.

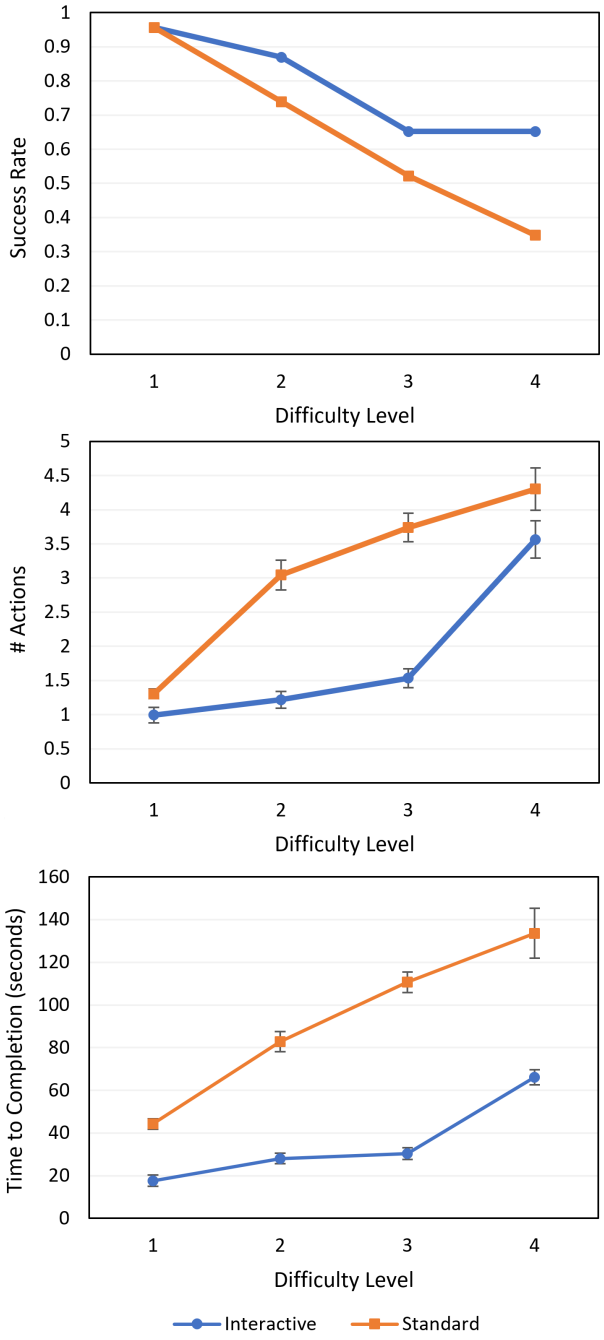


Figure 4.6: Success Rate (Top), Number of Actions (Middle) and Time to Completion (Bottom) for the Standard and Interactive Modes.

Mode	Successful	#Action	Time to Completion
Standard	No	6.39	119.08
Standard	Yes	4.67	84.08
Standard	All	5.10	92.83
Interactive	No	4.30	47.40
Interactive	Yes	3.45	29.81
Interactive	All	3.73	35.54

Table 4.6: Average Number of Actions and Time to Completion for successful and abandoned tasks for the Standard and Interactive Modes.

Effort: A 2 (interaction modes) by 4 (difficulty levels) ANOVA was performed for the the number of actions and time to completion for the standard and the interactive modes across different task difficulty levels. The result is also shown at middle and bottom portions of Figure 4.6. The interactive mode resulted in a smaller number of actions for all task difficulty levels ($p \ll 0.001$). The difference is smaller though for very hard tasks. This suggests that with harder tasks, participants had to either reformulate with the query or interact with the results to get to complete their tasks. Note that the actions are not equal though in terms of cost to the user. For example, reformulating the query is likely more expensive than editing the parameter of a filter module. To capture this, we use time to completion as a proxy to effort and compare the two modes as shown in the bottom portion of Figure 4.6. We see here that the interactive model resulted in faster task completion than the standard one ($p \ll 0.0001$), but unlike the number of actions, the gap was consistently large even for harder tasks. Since not all tasks have been completed successfully, we break down the different measure of effort (number of actions and time to completion) by whether the task was successfully completed or not in Table 4.6. A 2 (interaction modes) by 2 (successful or not) ANOVA was performed. As expected, we see that the participants had to perform a higher number of actions ($p \ll 0.001$) and longer time to completion ($p \ll 0.0001$) when the task was not completed successfully and

the interactive mode resulted in less effort across the board.

User Satisfaction: Overall the interactive mode was overwhelmingly preferred over the standard mode for the scenario we studied, with 17 participants preferring the interactive mode to the standard mode. Participants also reported higher overall satisfaction level with the interactive mode (60% were satisfied or strongly satisfied with the mode) compared to only 35% reporting they were satisfied or strongly satisfied with the standard mode. Participants also indicated that they had to put in extra effort to complete tasks with the standard system, with only 25% of them reporting that they only needed little effort to complete the tasks. This number increases to 70% for the interactive system.

In summary, the user study showed that interactive mode provides several benefits over the standard mode and results in higher task completion rate, lower effort and higher overall user satisfaction. This can be more evident if we examine the utterances submitted by a user using the standard mode (see Table 4.7). In this example, the standard model interpreted the utterance mostly correctly except for missing the "is not read" filter. The user reformulated the query and this time the model got the missing filter right but missed the order by received time operator. After a third reformulation, the model was able to get the correct interpretation. Alternatively, if the user had used the interactive mode, she could have simply added the "is not read" filter which was ranked among the top 3 in the module suggestions. This would have resulted in much faster task completion and hence higher user satisfaction.

4.6 Related Work

Natural language interface (also called semantic parsing in the computational linguistics community) research has spanned several decades [3]. Early NLIs are mostly rule-based. A set of rules are carefully designed to map natural language utterances in a domain to the corresponding meaning representation [3, 4]. Rule-based systems are characterized by a high

Standard Mode	
Action Type	Task Description
Query	show me unseen emails about PhD study
Query	show me emails about PhD study that I did not read
Query	show me the latest emails about PhD study that I did not read
Interactive Mode	
Action Type	Task Description
Query	show me unseen emails about PhD study
Add Module	New filter:"is not read"

Table 4.7: Examples of tasks using the standard and the interactive mode. Each example is the sequence of actions taken by a user to solve a task. The examples are representative of user behaviors with different modes.

precision on their admissible inputs. However, also salient is the brittleness of the systems when facing inputs not covered by the pre-defined rules. Over the past decade, statistical learning-based methods have gained momentum as they can naturally handle the uncertainty and ambiguity of natural language in a well-established statistical framework. Early learning-based methods were based on manually-defined features [6, 8]. With recent advances in deep learning, neural network based methods have become the mainstream for natural language interfaces [9, 10, 17, 92, 90], which are free of feature engineering and can more easily generalize to different domains. Our work follows this trend toward neural-network-based methods.

With the growth of web services, IoT devices, and mobile apps, natural language interfaces to API (NL2API) have attracted significant attention [96, 18, 93]. For example, Quirk et al. [96] study how to enable users to issue If-This-Then-That commands over a rich set of APIs including social media, mobile apps, and web services. Campagna et al. [93] present a virtual assistant system, at the core of which is a natural language interface to map user commands into APIs to IoT devices and mobile apps. Su et al. [18] study how to train an NL2API model by collecting training data from crowdsourcing, and propose a sequence-to-sequence model

for NL2API. While the main goal of this paper is to study user interaction in natural language interfaces, we conduct our study in the context of NL2API, and benefit from the insights from previous studies.

Natural language interfaces that can seek feedback from users to improve prediction accuracy have also received significant recent attention [106, 108, 14, 109]. For example, Li and Jagadish [106] develop an interactive natural language interface to relational databases (NLIDB). The mapping from natural language commands to SQL queries is mainly done using a rule-based mechanism, and user feedback is solicited to resolve ambiguities in the rule-based mapping. In contrast, we focus on neural-network-based natural language interfaces targeting web APIs. Iyer et al. [109] and Su et al. [14] study NLIDB and knowledge base search, respectively, and ask users to verify the correctness of the final results generated by the systems, and employ user feedback to improve system accuracy. However, none of the previous studies allows for fine-grained (e.g., module level) user interaction with neural network models.

Also related is a line of research on crowd-powered dialog systems [110, 111]. Different from out approach of semantic parsing with user feedback, these approaches leverage crowd workers to address user commands, which reduces workload on users possibly at the expense of response latency. Our work also resembles mixed-initiative approaches [112], leveraging human-machine collaboration.

The idea of modular neural networks are also explored in related problems such as visual question answering [102, 101] and program synthesis [103]. For example, Rabinovich et al. [103] propose a novel abstract syntax network to generate the abstract syntax tree of programs. In abstract syntax network, different modules are composed together to generate a full abstract syntax tree. Each module usually only fulfills a simple task, like choosing a value from a predefined list. In our model, each module is itself an attentive decoder, and needs to generate a full parameter sequence by reading the input utterance. Moreover, the main goal of the proposed modular Seq2Seq model is to help create interactive natural language interfaces, which has not

been explored previously.

4.7 Conclusion

We conducted a systematic study on fine-grained user interaction in natural language interfaces, focused on web APIs. To facilitate the creation of interactive natural language interfaces, we proposed a novel modular sequence-to-sequence model. By decomposing the prediction of a neural network into small, interpretable units called modules, the proposed model allows users to easily interpret predictions and correct possible errors. Through extensive simulation and human subject experiments with real-world APIs, we demonstrated that fine-grained user interaction can greatly improve the usability of natural language interfaces. Specifically, in the human subject experiment, we found that with the interactive natural language interface, users achieve a higher task success rate and a lower task completion time, greatly improving user satisfaction.

In this work, we focused on soliciting user feedback to improve prediction accuracy in a single session. Going forward, we are interested in the following question: *Given a new API, can we first cold-start an NL2API model with a reasonable prediction accuracy, and then improve it through user interaction?* In this vision, the interactivity of the NL2API helps form a closed data loop: It improves usability and thus attracts more users to use the system, which in turn accumulates more training data to improve the system.

Chapter 5

Conclusion

In this dissertation, we discussed the growing gap between human users and data, a pressing issue faced by the entire society. We then argued that natural language interfaces, which can bridge human users to data by parsing natural language commands from users into computer languages, are a promising way to bridge this gap. The main technical contribution of this dissertation is a systematic framework to bootstrap natural language interfaces in new domains, which leverages crowdsourcing, transfer learning, and user interaction to form a closed data loop for developing natural language interfaces.

Moving forward, the ultimate goal of research in this area is to create *virtual assistants for data analytics* that serve as a unified natural language interface to query, manipulate, and analyze massive and heterogeneous data. With such virtual assistants, users can stay focused on high-level thinking and decision making, instead of overwhelmed by low-level programming and software-specific implementation details. Several important research frontiers are as follows:

1. **Natural Language Interface to Analytics Functions**. Many successful data analytics software and platforms have been developed to meet the pervasive need of data science, which provide analytics functions to view, edit, visualize, and even create advanced predictions on

data. However, understanding the large amount of analytics functions and choosing proper ones for the task at hand is still a challenge for non-expert users, especially for complex tasks that require assembling a program of multiple functions. A typical example is as follows. In order to use nocturnal luminosity observed by satellites to study development in China, MIT economist Matt Lowe attempted to use a mainstream geographic information system to analyze geospatial data. The specific analysis he needed is to "*calculate the average luminosity along roads in 1994*." However, he found himself repeatedly puzzling over understanding software-specific geospatial data formats, choosing proper functions among hundreds, and chaining their inputs and outputs. Eventually a complex program with 9 steps was assembled for this seemingly simple and one-off analysis. An NLI that can automatically transduce user needs into proper analytics function calls will largely bridge this gap. Investigation on this problem can be divided into two steps: (1) *Transducing simple user needs into individual function calls*. (2) *Transducing complex user needs into a program of function calls*. With the popularity of machine learning toolkits like Tensorflow and scikit-learn, such NLI may even enable on-demand creation of machine learning models to support user needs like "*tell me the sentiment of these tweets*."

2. **Theoretical Foundations of Natural Language Interface**. (1) *What is the inherent structure of the NLI problem space?* The inherent structure in a problem can provide strong priors for learning. For example, the manifold assumption, that probability mass mostly lies in a low-dimensional subspace of the original high-dimensional feature space, has been widely used in many computer vision and speech recognition problems to make the learning more efficient. Both natural and formal languages are characterized by *compositionality*, the algebraic capacity to understand and produce a potentially infinite number of novel combinations from known components. If a person understands the meaning of integer "three" and "four" and operator "plus", she can immediately understand and compose phrases like

"three plus four plus four plus three". The lack of compositionality is one main reason for the data inefficiency of neural networks. (2) *How to integrate symbolic and neural computation*. While early NLI models are purely symbolic, now we are shifting towards the other extreme of the spectrum, and study the NLI problem almost exclusively in the realm of neural computing, largely ignoring the symbolic nature of languages. For integrating symbolic and neural computation in NLI, one promising direction is to borrow ideas from cognitive science, e.g., Tensor Product Representation from Paul Smolensky that can encode symbolic structures losslessly with numerical tensors.

3. **Human-Computer Conversational Interaction**. Decision making is a dynamic process involving multi-round interactions with data for hypothesis generation and verification. Such a process can be naturally accommodated in a conversation between human and computer. Traditional research on dialog and conversational interface, e.g., on tasks like restaurant or flight booking, often assumes fixed "slots" for user needs. For example, the task of restaurant booking has slots like time, party size, etc. The main goal of dialog management is then to interact with the user and fill these slots. However, for the virtual assistant for data analytics, user needs come in unrestricted forms and have no fixed slots.

Bibliography

[1] N. Yang and E. Hing, *Table of electronic health record adoption and use among office-based physicians in the U.S.*, *2015 National Electronic Health Records Survey* (2017).

[2] "Dante Labs Offers EUR 850 Whole Genome Sequencing and Interpretation for the First Time in the World." `https://www.prnewswire.com/news-releases/dante-labs-offers-eur-850-whole-genome-sequencing-and-interpretation-for-the-first-time-in-the-world-300443895.html`. Accessed: 2018-12-32.

[3] W. A. Woods, *Progress in natural language understanding: an application to lunar geology*, in *Proceedings of the American Federation of Information Processing Societies Conference*, 1973.

[4] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, *Natural language interfaces to databases–an introduction*, *Natural language engineering* **1** (1995), no. 1 29–81.

[5] A.-M. Popescu, O. Etzioni, and H. Kautz, *Towards a theory of natural language interfaces to databases*, in *Proceedings of the 8th international conference on Intelligent user interfaces*, pp. 149–157, ACM, 2003.

[6] L. S. Zettlemoyer and M. Collins, *Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars*, in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[7] R. J. Kate, Y. W. Wong, and R. J. Mooney, *Learning to transform natural to formal languages*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2005.

[8] J. Berant, A. Chou, R. Frostig, and P. Liang, *Semantic parsing on Freebase from question-answer pairs*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2013.

[9] S. W.-t. Yih, M.-W. Chang, X. He, and J. Gao, *Semantic parsing via staged query graph generation: Question answering with knowledge base*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[10] L. Dong and M. Lapata, *Language to logical form with neural attention*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[11] I. Gur, S. Yavuz, Y. Su, and X. Yan, *Dialsql: Dialogue based structured query generation*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1339–1349, 2018.

[12] S. Yavuz, I. Gur, Y. Su, and X. Yan, *What it takes to achieve 100 percent condition accuracy on wikisql*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pp. 1702–1711, 2018.

[13] Y. Su, H. Sun, B. Sadler, M. Srivatsa, I. Gür, Z. Yan, and X. Yan, *On generating characteristic-rich question sets for QA evaluation*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016.

[14] Y. Su, S. Yang, H. Sun, M. Srivatsa, S. Kase, M. Vanni, and X. Yan, *Exploiting relevance feedback in knowledge graph search*, in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2015.

[15] S. Yavuz, I. Gur, Y. Su, M. Srivatsa, and X. Yan, *Improving semantic parsing via answer type inference*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pp. 149–159, 2016.

[16] S. Yavuz, I. Gur, Y. Su, and X. Yan, *Recovering question answering errors via query revision*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pp. 903–909, 2017.

[17] H. Sun, H. Ma, X. He, W.-t. Yih, Y. Su, and X. Yan, *Table cell search for question answering*, in *Proceedings of the International Conference on World Wide Web*, 2016.

[18] Y. Su, A. H. Awadallah, M. Khabsa, P. Pantel, M. Gamon, and M. Encarnacion, *Building natural language interfaces to web apis*, in *Proceedings of the International Conference on Information and Knowledge Management*, 2017.

[19] Y. Su, A. H. Awadallah, M. Wang, and R. W. White, *Natural language interfaces with fine-grained user interaction: A case study on web apis*, in *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018.

[20] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, *et. al.*, *DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia*, *Semantic Web Journal* **6** (2014), no. 2 167–195.

[21] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, *Freebase: a collaboratively created graph database for structuring human knowledge*, in *Proceedings of the ACM SIGMOD International conference on Management of data*, 2008.

[22] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum, *Deep answers for naturally asked questions on the web of data*, in *Proceedings of WWW*, 2012.

[23] Q. Cai and A. Yates, *Large-scale semantic parsing via schema matching and lexicon extension.*, in *Proceedings of ACL*, 2013.

[24] T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer, *Scaling semantic parsers with on-the-fly ontology matching*, in *Proceedings of EMNLP*, 2013.

[25] J. Berant and P. Liang, *Semantic parsing via paraphrasing.*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[26] A. Fader, L. Zettlemoyer, and O. Etzioni, *Open question answering over curated and extracted knowledge bases*, in *Proceedings of KDD*, 2014.

[27] S. Reddy, M. Lapata, and M. Steedman, *Large-scale semantic parsing without question-answer pairs*, *Transactions of the Association for Computational Linguistics* **2** (2014) 377–392.

[28] J. Bao, N. Duan, M. Zhou, and T. Zhao, *Knowledge-based question answering as machine translation*, in *Proceedings of ACL*, 2014.

[29] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao, *Natural language question answering over rdf: a graph data driven approach*, in *Proceedings of SIGMOD*, 2014.

[30] X. Yao and B. Van Durme, *Information extraction over structured data: Question answering with Freebase.*, in *Proceedings of ACL*, 2014.

[31] H. Sun, H. Ma, W.-t. Yih, C.-T. Tsai, J. Liu, and M.-W. Chang, *Open domain question answering via semantic enrichment*, in *Proceedings of WWW*, 2015.

[32] L. Dong, F. Wei, M. Zhou, and K. Xu, *Question answering over Freebase with multi-column convolutional neural networks*, in *Proceedings of ACL*, 2015.

[33] X. Yao, *Lean question answering over Freebase from scratch*, in *Proceedings of NAACL*, 2015.

[34] J. Berant and P. Liang, *Imitation learning of agenda-based semantic parsers*, *Transactions of the Association for Computational Linguistics* **3** (2015) 545–558.

[35] E. M. Voorhees and D. M. Tice, *Building a question answering test collection*, in *Proceedings of SIGIR*, 2000.

[36] V. Lopez, C. Unger, P. Cimiano, and E. Motta, *Evaluating question answering over linked data*, *Web Semantics: Science, Services and Agents on the World Wide Web* **21** (2013) 3–13.

[37] A. Bordes, N. Usunier, S. Chopra, and J. Weston, *Large-scale simple question answering with memory networks*, *arXiv preprint arXiv:1506.02075* (2015).

[38] I. V. Serban, A. García-Durán, C. Gulcehre, S. Ahn, S. Chandar, A. Courville, and Y. Bengio, *Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus*, in *Proceedings of ACL*, 2016.

[39] X. Yao, J. Berant, and B. Van Durme, *Freebase QA: Information extraction or semantic parsing?*, in *Proceedings of ACL*, 2014.

[40] J. Burger, C. Cardie, V. Chaudhri, R. Gaizauskas, S. Harabagiu, D. Israel, C. Jacquemin, C.-Y. Lin, S. Maiorano, G. Miller, *et. al.*, *Issues, tasks and program structures to roadmap research in question & answering (q&a)*, in *Document Understanding Conferences Roadmapping Documents*, pp. 1–35, 2001.

[41] L. Hirschman and R. Gaizauskas, *Natural language question answering: the view from here*, *natural language engineering* **7** (2001), no. 4 275–300.

[42] Y. Wang, J. Berant, and P. Liang, *Building a semantic parser overnight*, in *Proceedings of ACL*, 2015.

[43] P. Liang, *Lambda dependency-based compositional semantics*, *arXiv preprint arXiv:1309.4408* (2013).

[44] M. Bendersky and W. B. Croft, *Analysis of long queries in a large scale search log*, in *Proceedings of the 2009 workshop on Web Search Click Data*, 2009.

[45] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh, *The value of semantic parse labeling for knowledge base question answering*, in *Proceedings of ACL*, 2016.

[46] A. Aula, R. M. Khan, and Z. Guan, *How does search behavior change as search becomes more difficult?*, in *Proceedings of CHI*, 2010.

[47] E. Gabrilovich, M. Ringgaard, and A. Subramanya, "FACC1: Freebase annotation of ClueWeb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0)." `http://lemurproject.org/clueweb09/` and `http://lemurproject.org/clueweb12/`, 2013.

[48] T.-H. Wen, M. Gašić, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young, *Semantically conditioned LSTM-based natural language generation for spoken dialogue systems*, in *Proceedings of EMNLP*, 2015.

[49] O. Dušek and F. Jurčíček, *Training a natural language generator from unaligned data*, in *Proceedings of ACL*, 2015.

[50] J. M. Zelle and R. J. Mooney, *Learning to parse database queries using inductive logic programming*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1996.

[51] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam, *Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant*, in *Proceedings of the International Conference on World Wide Web*, pp. 341–350, International World Wide Web Conferences Steering Committee, 2017.

[52] D. L. Chen and R. J. Mooney, *Learning to interpret natural language navigation instructions from observations.*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2011.

[53] S. A. Tellex, T. F. Kollar, S. R. Dickerson, M. R. Walter, A. Banerjee, S. Teller, and N. Roy, *Understanding natural language commands for robotic navigation and mobile manipulation*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2011.

[54] Y. Artzi and L. Zettlemoyer, *Weakly supervised learning of semantic parsers for mapping instructions to actions*, *Transactions of the Association for Computational Linguistics* **1** (2013) 49–62.

[55] Y. Bisk, D. Yuret, and D. Marcu, *Natural language communication with robots*, in *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

[56] H. Daumé III and D. Marcu, *Domain adaptation for statistical classifiers*, *Journal of Artificial Intelligence Research* **26** (2006) 101–126.

[57] J. Blitzer, R. McDonald, and F. Pereira, *Domain adaptation with structural correspondence learning*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2006.

[58] Y. Wang, J. Berant, and P. Liang, *Building a semantic parser overnight*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[59] H. Daumé III and J. Jagarlamudi, *Domain adaptation for machine translation by mining unseen words*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2011.

[60] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2013.

[61] X. Glorot, A. Bordes, and Y. Bengio, *Domain adaptation for large-scale sentiment classification: A deep learning approach*, in *Proceedings of the International Conference on Machine Learning*, 2011.

[62] M. Chen, Z. Xu, K. Weinberger, and F. Sha, *Marginalized denoising autoencoders for domain adaptation*, in *Proceedings of the International Conference on Machine Learning*, 2012.

[63] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, in *Proceedings of the Annual Conference on Neural Information Processing Systems*, pp. 3104–3112, 2014.

[64] Q. Cai and A. Yates, *Semantic parsing freebase: Towards open-domain semantic parsing*, in *Second Joint Conference on Lexical and Computational Semantics (* SEM)*, 2013.

[65] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao, *Question answering on freebase via relation extraction and textual evidence*, *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (2016).

[66] J. Herzig and J. Berant, *Neural semantic parsing over multiple knowledge-bases*, *arXiv:1702.01569 [cs.CL]* (2017).

[67] R. Jia and P. Liang, *Data recombination for neural semantic parsing*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[68] S. Reddy, O. Täckström, M. Collins, T. Kwiatkowski, D. Das, M. Steedman, and M. Lapata, *Transforming dependency structures to logical forms for semantic parsing*, *Transactions of the Association for Computational Linguistics* **4** (2016) 127–140.

[69] S. Reddy, O. Täckström, S. Petrov, M. Steedman, and M. Lapata, *Universal semantic parsing*, *arXiv:1702.03196 [cs.CL]* (2017).

[70] D. Goldwasser and D. Roth, *Leveraging domain-independent information in semantic parsing.*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[71] A. Fader, L. S. Zettlemoyer, and O. Etzioni, *Paraphrase-driven learning for open question answering.*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[72] S. Narayan, S. Reddy, and S. B. Cohen, *Paraphrase generation from latent-variable PCFGs for semantic parsing*, *arXiv:1601.06068 [cs.CL]* (2016).

[73] S. J. Pan and Q. Yang, *A survey on transfer learning*, *IEEE Transactions on knowledge and data engineering* **22** (2010), no. 10 1345–1359.

[74] C. Chelba and A. Acero, *Adaptation of maximum entropy capitalizer: Little data can help a lot*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2004.

[75] H. Daumé III, *Frustratingly easy domain adaptation*, *arXiv:0907.1815 [cs.LG]* (2009).

[76] R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning, *Dynamic pooling and unfolding recursive autoencoders for paraphrase detection.*, in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2011.

[77] B. Hu, Z. Lu, H. Li, and Q. Chen, *Convolutional neural network architectures for matching natural language sentences*, in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2014.

[78] W. Yin and H. Schütze, *MultiGranCNN: An architecture for general matching of text chunks on multiple levels of granularity.*, in *ACL*, 2015.

[79] K. Cho, B. van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using RNN encoder–decoder for statistical machine translation*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734, 2014.

[80] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, *Grammar as a foreign language*, in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2015.

[81] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, *arXiv:1207.0580 [cs.NE]* (2012).

[82] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukács, M. Ganea, P. Young, *et. al.*, *Smart reply: Automated response suggestion for email*, in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016.

[83] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, *Why does unsupervised pre-training help deep learning?*, *Journal of Machine Learning Research* **11** (2010), no. Feb 625–660.

[84] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks.*, in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010.

[85] O. Levy, Y. Goldberg, and I. Dagan, *Improving distributional similarity with lessons learned from word embeddings*, *Transactions of the Association for Computational Linguistics* **3** (2015) 211–225.

[86] J. Pennington, R. Socher, and C. D. Manning, *Glove: Global vectors for word representation*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014.

[87] C. Xiao, M. Dymetman, and C. Gardent, *Sequence-based structured prediction for semantic parsing*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[88] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch, *PPDB: The paraphrase database.*, in *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2013.

[89] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et. al.*, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, *arXiv:1603.04467 [cs.DC]* (2016).

[90] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao, *Neural symbolic machines: Learning semantic parsers on freebase with weak supervision*, *arXiv:1611.00020 [cs.CL]* (2016).

[91] P. Pasupat and P. Liang, *Compositional semantic parsing on semi-structured tables*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[92] V. Zhong, C. Xiong, and R. Socher, *Seq2sql: Generating structured queries from natural language using reinforcement learning*, *arXiv:1709.00103 [cs.CL]* (2017).

[93] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam, *Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant*, in *Proceedings of the International Conference on World Wide Web*, pp. 341–350, 2017.

[94] E. W. Ngai, L. Xiu, and D. C. Chau, *Application of data mining techniques in customer relationship management: A literature review and classification*, *Expert systems with applications* **36** (2009), no. 2 2592–2602.

[95] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, *Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services*, *IEEE transactions on Services Computing* **3** (2010), no. 3 223–235.

[96] C. Quirk, R. Mooney, and M. Galley, *Language to code: Learning semantic parsers for if-this-then-that recipes*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[97] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[98] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services*, in *Web Services*, pp. 123–149. Springer, 2004.

[99] L. Zhang, C. Stover, A. Lins, C. Buckley, and P. Mohapatra, *Characterizing mobile open apis in smartphone apps*, in *Networking Conference, 2014 IFIP*, pp. 1–9, IEEE, 2014.

[100] Y. Su and X. Yan, *Cross-domain semantic parsing via paraphrasing*, in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2017.

[101] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, *Neural module networks*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[102] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, *Learning to compose neural networks for question answering*, in *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

[103] M. Rabinovich, M. Stern, and D. Klein, *Abstract syntax networks for code generation and semantic parsing*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[104] M. Looks, M. Herreshoff, D. Hutchins, and P. Norvig, *Deep learning with dynamic computation graphs*, in *Proceedings of the International Conference on Learning Representations*, 2017.

[105] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv:1412.6980 [cs.LG]* (2014).

[106] F. Li and H. V. Jagadish, *Constructing an interactive natural language interface for relational databases*, *Proceedings of VLDB Endowment* **8** (Sept., 2014) 73–84.

[107] J. Kiseleva, K. Williams, J. Jiang, A. Hassan Awadallah, A. C. Crook, I. Zitouni, and T. Anastasakos, *Understanding user satisfaction with intelligent assistants*, in *Proceedings of the ACM SIGIR Conference on Human Information Interaction and Retrieval*, pp. 121–130, 2016.

[108] S. I. Wang, P. Liang, and C. D. Manning, *Learning language games through interaction*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[109] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, *Learning a neural semantic parser from user feedback*, in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[110] T.-H. K. Huang, W. S. Lasecki, and J. P. Bigham, *Guardian: A crowd-powered spoken dialog system for web apis*, in *Third AAAI conference on human computation and crowdsourcing*, 2015.

[111] T.-H. K. Huang, J. C. Chang, and J. P. Bigham, *Evorus: A crowd-powered conversational assistant built to automate itself over time*, in *Proceedings of Conference on Human Factors in Computing Systems*, 2018.

[112] Y. Huang and T. M. Mitchell, *Exploring hierarchical user feedback in email clustering*, in *AAAI Enhanced Messaging Workshop*, 2008.