Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Neural Logic Framework for Digital Assistants

Author:
Nuri Cingillioglu

Supervisor:
Prof. Alessandra Russo

Second Marker:
Dr. Krysia Broda

June 16, 2017

Abstract

Digital assistants are becoming ubiquitous with consumers across mobile platforms helping with everyday tasks. The natural language interface of most assistants are built on machine learning based intent parsing techniques. This design cannot handle higher level abstract reasoning such as defaults while logic programs can incorporate them.

In this project we present Kevin, a digital personal assistant with a logical framework built on top of neural networks to provide a flexible execution environment while harnessing the capabilities of machine learning at a lower level. Kevin demonstrates natural language based logical constructs such as unification and resolution with integrated neural network information retrieval.

Acknowledgements

I would like to thank:

- my parents for their wholehearted support, endless love and perpetual encouragement.
- my supervisor Prof. Alessandra Russo for giving me this opportunity by accepting the project proposal and allowing me to explore the topic as well as her PhD student Mark Law for his feedback.
- my personal tutor Dr. Alistair Donaldson for his support and advice for the years.
- my friends for their companionship and solace throughout my degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Virtual, digital assistants are computer programs that interact with users using natural language to perform tasks for the user. When they are customised for a specific user, they are commonly referred to as personal assistants and are designed to provide user specific tasks such as providing shopping list planning [5]. With the recent advancements in natural language processing such as speech recognition, personal assistants grew in popularity and saw commercial success such as Apple's Siri [6] mainly on mobile platforms.

In this project we present Kevin, a digital personal assistant with a logical framework built on top of neural networks to provide a flexible execution environment while harnessing the capabilities of machine learning at a lower level. This hybrid infrastructure allows Kevin handle natural language constructs within logic-programming using existing and custom neural networks.

1.1 Motivation

As digital personal assistants are becoming widespread across various mediums to provide information on numerous subjects, the need for conversationally apt agents emerged rapidly. Contenders such as Apple's Siri [6] have been released into the consumer market as the primary conversational agent in the mobile operating system iOS. Such personal assistants now play a significant role in reshaping how we interact with computers and push the domain of natural language processing under the spotlight.

The field of natural language processing has seen great success, particularly with advancing machine learning techniques. Syntax analysis with dependency graphs, keyword extraction and named entity recognition have been addressed by academics as well as by industry [7]. Despite the success with machine learning, the true semantics of sentences have not been revealed using similar techniques. Classical logic, on the other hand, has been dealing with manipulating knowledge bases and inferred conclusions. It proves a valuable tool even when faced with the representation of time and knowledge [8]. Therefore, a personal assistant with logic framework lends great potential.

1.2 Objectives

The objective of this project is to provide a proof-of-concept personal assistant with a novel hybrid back end that fuses logic programming with neural networks. We want to understand what benefits would there be for a hybrid system if any in the context of digital assistants. We aim to handle natural language within a logic framework and provide a flexible runtime environment that can integrate with other components such as neural networks. At a high level we aim to:

- Find a suitable representation of natural language that could be used in a logic programming context. We do not restrict ourselves to finding a representation that would be compatible with existing tools.
- Place natural language into a logic framework, either an existing one or one built from scratch. The goal is to have a framework that is comfortable with handling natural language constructs without too much pre-processing.
- Integrate neural networks to boost the robustness of the logic framework when handling noisy input such as user queries. Logic programming is rigidly structured and we aim to ease that with machine learning techniques that are designed to handle unstructured natural language input.
- Explore how certain reasoning elements can benefit a digital assistant using the logic framework. Concepts such as negation by failure and deduction should in theory be possible.

With Kevin, we would observe to what extent having a logic framework might contribute to handling user input in a digital assistant. The interaction between the logic components and neural networks is also an important area of interest as merging logic with neural networks is an active area of research.

1.3 Challenges

The project harbours several challenges as the field of conversational intelligent personal assistants has recently started to proliferate. Creating an assistant from scratch requires the amalgamation of many different concepts from machine learning, logic programming, natural language processing and software engineering. There were, however, four main challenges for building Kevin:

- **Lack of documentation of existing personal assistants** limited our background research around recently created assistants. Since many of the successful personal assistants live in the private industry, it is quite difficult to find relevant documentation or open-source alternatives with similar features. The complexity of building one also hides publicly amiable documentation.
- **Representing natural language in logic** is a difficult task due to the ambiguity and unstructured form of natural language. There are many phrases that may have the same semantics but slight variations can change the meaning to the extent the response becomes irrelevant to the query.
- **Integrating logic with neural networks** is an active area of research and a key concept for this project. With no major examples to tackle the problem, it is challenging both theoretically and technically.
- **Scaling logical knowledge bases** is a possible limitation. Capturing vast amounts of natural language information in a logical representation based on rules might become tedious and create problems such as inconsistent states.

1.4 Contributions

We present Kevin with a novel logic framework that can directly handle natural language with the aid of recent machine learning techniques such as GloVe [3] word vectors and a custom variation of the BiDAF [9] model called the guided BiDAF. The framework functions directly on annotated dependency parse trees and contains no pre-processing between the output of a natural language library and the input to the logic framework; Kevin consumes the output of the natural language library directly without any conversions or translations. Kevin covers different query formats using the framework such as storing facts, intent parsing and deduction (chapter 4).

The multi-stage pipeline provides a modular framework (section 7.3) with options for external components such as neural networks to be integrated in order to extend the functionality of Kevin. As a result, besides logical resolutions Kevin can consume external textual resources to answer free form informational queries.

Chapter 2

Related Work

In this chapter we discuss various projects in the direction of creating intelligent personal assistants as well as understanding natural language. We look at how natural language and computers evolved over time and their role in creating digital assistants. The ideas in this chapter influence what Kevin is and how it fits in the wider spectrum of available work in the field.

2.1 Natural Language Interactions

This section covers how natural language as means of interacting with computers evolved over time. We look at early conversational agents that laid the foundational work for creating user interactions in English and how their key concepts influenced the design of Kevin.

2.1.1 Turing Test

At the very start of conversational agents was the Turing Test. The Turing Test is a test for intelligence in a computer, requiring that a human being should be unable to distinguish the machine from another human being by using the replies to questions put to both [10]. In most occasions the aim was to provide a general conversational agent that can handle a wide range of questions in the most natural way possible. The Loebner Prize competition held annually saw the progress in the field with respect to these agents and the Turing Test on which the competition format is based [11]. At the time providing an human-like *chatterbot* marked a tangible goal in artificial intelligence, capturing the imagination of the public with early chatterbots such as ELIZA.

However, the span of the Turing Test often sprouted philosophical questions and digressed from possible practical applications that could be developed using conversational agents [12]. The test focused at providing a natural language dialogue as human-like as possible. With the lack of available information and limited access to the internet, early agents often mimicked a closed circuit application with meagre capacity for information processing significantly squaring any pragmatism. This situation begged the question, do we actually need an agent to be human-like to yield practical applications?

Despite its early formulation and philosophical debates, the Turing Test provides an important starting point for and fierce considerations on what the purpose of an intelligent personal assistant truly is. In particular, as the information in the internet expanded well beyond what humans could process and search for, the need for computer aided information retrieval tools such as keyword based search engines [13] gained great popularity. To achieve a natural gateway to this pool of information kindled agents with natural language interfaces and paved the way for personal assistants as covered by this project.

2.1.2 Conversational Agents

With the exciting aura of the Turing Test, conversational agents gained ground for several decades to provide different experiences to users. We will analyse some to provide a basis on building personal assistants and how the field transformed from chatterbots to practical assistants.

ELIZA

Created from 1964 to 1966 by Joseph Weizenbaum at the MIT Artificial Intelligence Laboratory, ELIZA was an early chatterbot program aimed to demonstrate the superficiality of communication between man and machine [14]. We need to consider ELIZA due to its simplistic implementation and potential to create the illusion of intelligence. Based on pattern matching pre-determined cues and canned responses, ELIZA was able to provide human-like conversation and regarded as one of the first programs to actually pass the Turing Test in certain cases. In particular, the simulation of Rogerian psychotherapist narrowed down a certain format of perceived intelligence and started the path of conversational agents.

User:	I'm sad.
ELIZA:	How long have you been sad?
User:	2 weeks.
ELIZA:	Tell me more...

Table 2.1: Example conversation with ELIZA.

As seen in the example conversation in table 2.1, the dialogue produced by the program creates the illusion of understanding by asking questions related to the phrases provided by the user. However, such an agent does not provide any practical features besides perhaps therapeutic ones from the perspective of the user. The natural flow of the conversion is crucial in allowing an engaging environment for the user to interact with. This theme unearthed by ELIZA should shed light onto the behaviour of *natural* personal assistants.

ALICE

Using Artificial Intelligence Markup Language (AIML), an XML dialect for creating natural language software agents, Artificial Linguistic Internet Computer Entity (ALICE) is an open-source conversational agent originally created by Richard Wallace [15]. ALICE moves to portray more cognisance in recognising subjects and entities such as cities. Winning the Loebner Prize three times, ALICE provides a base line on what pattern matching techniques can achieve. Despite such success, ALICE wasn't as successful in a Turing Test due to its mechanistic approach and repetitive structure.

Bob:	Are you a poet?
ALICE:	Not really but I enjoy original human thoughts.
Bob:	Where is London?
ALICE:	it is a city in England.
Bob:	What is a car?
ALICE:	A car is what humans use to drive around in.

Table 2.2: Example conversation with ALICE.

The important implications of ALICE for this project can be observed in table 2.2. Providing answers to informational queries start to form a part of a conversational agent. It is essential that such queries are recognised and processed properly to provide the first steps into creating *assistants* which can access a far larger information pool on the internet than a human can. Now, we start to reveal the potential of such conversational agents and their inherent computational power. Even with basic pattern matching techniques, we can work towards for a practical conversational agent.

Jabberwacky - Cleverbot

A step forward from pre-determined answers, initially Jabberwacky, Cleverbot renders itself as a conversational agent that learns from past dialogues with users. Created by Rollo Carpenter, Cleverbot accumulates previous conversations into a search space and then compare fuzzily to find a response [16]. The malleable response structure of Cleverbot puts it beyond static chatterbots; however, falls blindly into the realm of search problems similar to early artificial intelligence techniques used in games [17].

As seen in table 2.3, the shortcomings of fuzzy search based response mechanism are

User:	Where is London?
Cleverbot:	London is in England.
User:	What is a car?
Cleverbot:	Cake is something that a person makes, it is usually eaten at celebrations like birthdays..
User:	What is a car?
Cleverbot:	It is a pet.

Table 2.3: Example conversation with Cleverbot.

revealed. The car is mistaken for a cake and the response for what a cake is provided to the user. Besides the conversational aspects, the two most important features of Cleverbot provide insights into the domain of personal assistants today:

1. **Provide a malleable response mechanism.** The responses are not fixed and there is a sense of learning although in a very crude fashion. Still certain informational queries are replied successfully.

2. **Maintain multiple conversations over a web platform.** With exceeding over 80,000 simultaneous conversations online, Cleverbot hints at a cloud based conversational agent that addresses multiple users with ease.

The fuzzy search through a known set of learnt interactions will emerge as a key concept throughout this project. With the help of semantically aware pre-trained neural networks, we can create more robust and efficient search algorithms to fuzzily search a given knowledge base.

2.2 Domain-specific Assistants

Benefits of creating domain-specific technology to aid certain tasks have been developed over many years. The idea of computers can make certain processes more efficient, reliable, accurate and easier has proved itself invaluable in many cases such as early hand-held devices in the context of restaurants [18] and in e-government applications with the e-citizen assistant [19]. Even without intelligent agents, technology that brought people and information together using the existing mobile platforms at the time proved speedier service, better usability and accuracy.

As the term personal digital assistant moves beyond hand-held devices, we look at some domain-specific assistants with natural language interfaces that exploit the limited scope of their domain. These agents pushed conversational agents into fulfilling a more practical purpose rather than a general purpose assistant.

2.2.1 PANDA: Virtual assistant for in-car child entertainment

PANDA is a parental assistive natural driving assistant developed to be a virtual driving-parent helper with young children in the back seat [20]. We look at PANDA because of its unique setting and niche role it plays. PANDA is designed to address specific issues in the family car to assist the parent in reducing attentional and task load, provide parent-supervised entertainment for children and create social, educational and relational activities.

Given its specific role, PANDA was built to cover different scenarios that are triggered with specific voice commands such as "show movie" and follow a menu-like option. This interaction is scripted providing absolute functionality but little flexibility similar to other pre-determined agents [21]. The set of possible action space is fixed and for good reason as the cases the agent covers are relatively small, yielding an easy-to-use in-car assistant. This approach unearths several important concepts for a limited-domain agent:

- **Small domain interactions can be manually handled** by the programmer anticipating most commands. This situation resurfaces in many other specific voice command natural language interfaces such as smart home applications.
- **The user is restricted to certain commands** and needs to know which ones trigger the correct actions. Slight variations in utterances might not be recognised by the system forcing the user to be more specific.
- **Command based interaction** is maintained between the user and the system. Every utterance has a corresponding action; therefore, most are in imperative form telling the system to do something.

From the perspective of the programmer and the user, the natural language interface acts as a protocol. Instead of forcing the user to the click certain buttons, such command based agents require certain utterances instead. Only the interaction form changes perhaps benefiting ease of user; fundamentally these agents pave the way for more free form natural language interactions between humans and computers.

2.2.2 Radar: Assistant agent for email and task management

Agents do not necessarily need a direct natural language interface with the user. In the case of Radar, a personal assistant that learns to reduce the workload, user emails are the main source of information to process [22]. This situation is in contrast to creating a dialogue with the user and change the terms of a digital assistant.

As seen in figure 2.1, Radar starts by processing user emails. Then it tries to categorise based on content and extract task-specific parameters; for example, how many people will

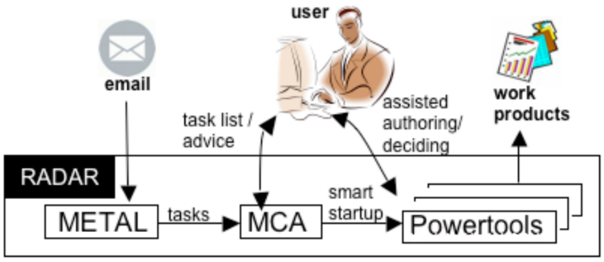


Figure 2.1: The Radar system pipeline overview.

attend a meeting. This information is automatically provided to the user creating an assistive agent that allows the user more efficiently and accurately process their email. The advantage of a limited context again comes into spotlight as "RADAR assumes that the types of typical user tasks are known and treats email task detection as a text classification problem using a regularised logistic regression suite of classifiers (based on body, headers, links) and combines their results." Several extensions such as CMRadar for calendar management have been built upon Radar to cover different task scenarios [23].

Agents with indirect user input play a significant role in processing data from the internet. Streaming data about the user such as their location can provide the extra needed context in order to create a richer conversation thereafter. Throughout this project we follow in certain cases a similar approach and consume external sources to process user queries.

2.2.3 SimCity game assistant

As we move towards more intelligent systems, maintaining a limited context allows agents to perform optimal decision making with a representative model of underlying domain. One intriguing example is the conversational agent to provide assistance in SimCity [24] like games. "The intelligent conversational agent, interacting in natural language with the user, obtains information about the current state of the city and gives, as a consequence, suggestions about the best strategies to apply. The reasoning capabilities of the agent are obtained through the integration of a traditional ontology-based reasoning system with a probabilistic one." [25].

The pipeline of the SimCity assistant provides an insight into domain-specific conversational assistant architectures. As seen in figure 2.2, we obverse an explicit abstraction of the natural language interface under "Linguistic Area" component. The domain-specific knowledge is attached to the decision making components fed by the current domain information. All this architecture allows the assistant to combine the optimal decision making process with a natural language interface.

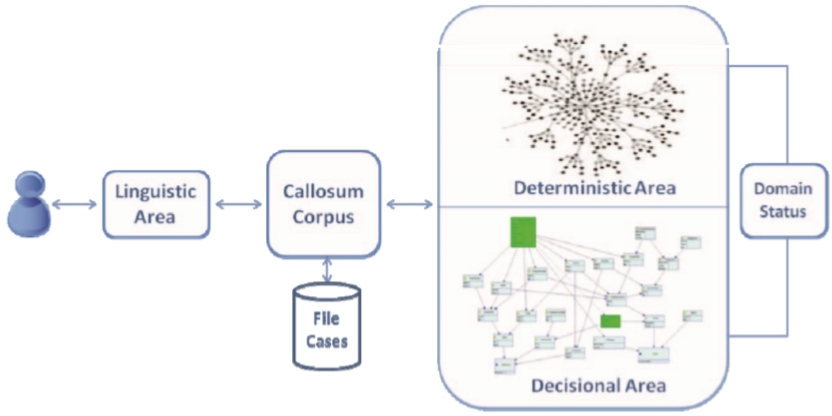


Figure 2.2: The SimCity assistant architecture.

The main point of interest is the linguistic interface the assistant provides. The interface is based on ALICE [15] and built from scratch. The current knowledge of the domain is modelled as expected from an AI in a game, with the agent having full knowledge of the game and its current status [17]. However, this information is communicated via a natural language interface which turns essentially a game AI into an virtual assistant. Therefore, attaching a natural language interface to an existing system can provide a base line for an assistant.

2.2.4 Creating Conversation Flows

As much as the entities provide local context for the phrases given, many actions require a sustained conversation over multiple phrases. A common paradigm emerging in conversational user interfaces is to model the flow of the conversation by hand for a domain specific interaction. For example, conversations follow hard-coded paths similar to checkout procedures triggered by *intents* or key phrases in the case of Converse AI [26]. In the context of such agents, the conversation can be represented by a flow graph with intents as edges.

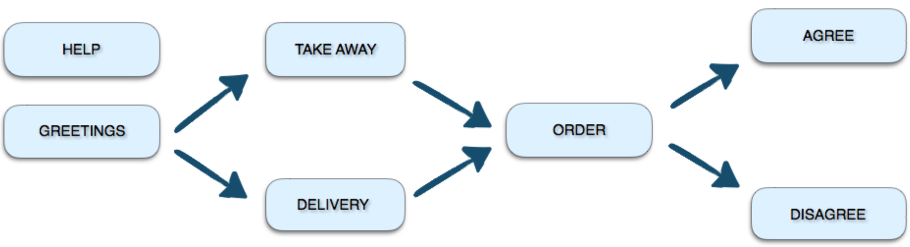


Figure 2.3: Recast AI conversation flow representation.

Figure 2.3 shows a pre-determined conversation flow graph from another conversational user interface library Recast AI [27]. Although this situation might resemble efforts made in early days of artificial intelligence in which actions are coded by the developer, the restriction of the domain allows practical applications to be created. As such, the conversational user interfaces in these cases are *not generic*. However, the advantage of restricting the domain not only provides more control over the interaction but also opens up more integration possibilities such as payment portals. When certain items are to be purchased, classical user interfaces such as forms can be presented to the user with the context information extracted already and filled including the shopping list, address, delivery time as implemented by Chatfuel [28].

Overall, the limited domain approach allows very fine control over the interaction that can be pre-determined manually by the programmers. On the other hand, the interactions are mechanic and restricted pushing towards a strictly practical approach rather than a general purpose assistant. This theme of restricting and relaxing the domain will be touched upon in our implementation as it is still an active area of research.

2.3 Towards Intelligent Assistants

When the domain is fixed, very effective assistants that interact directly or indirectly with the user can be created. This is not only the case with conversational assistants but also with any computerised assistive technology. For example, describing the next generation of driving assistant systems will not only be reactive but also predictive working behind the scenes as the driver requires correction, calculating optimal manoeuvres [29]. However, today intelligent personal assistants demand general understanding of a variety of topics starting with everyday tasks but possibly expanding into anything the user requires.

2.3.1 Domain Problem

Most of the domain-specific assistants achieved successful results thanks to the restricted action space; thus the linguistic elements that might arise in such a context were limited. Scripting, when the user natural language input is matched against expected patterns, were often used as a way of *understanding* the query [21]. This approach provides fine control over the assistant with good effectiveness; however, the assistant's actions are equally limited by the domain. It seems the domain knowledge is a crucial theme in building a personal digital assistant, and for good reason.

Semantic-based conversational frameworks have only been tested in environments when the domain is limited [30]. Generalisation of the domain to allow the agent understand compromises its immediate functionality due to less information about the domain.

The more general the agent is, the less effective it would be in specific tasks. On the contrary, we observed specific agents which provided substantial assistive features in their respective domain. We shall better define the domain problem later in the report for the purposes of and implications to this project.

2.3.2 Adding Knowledge

To achieve a certain level of generalisation of the knowledge the assistant can retain, we can manually create connections and provide information to the assistant. This approach is maintained in the likes of Apple's Siri [6] and Amazon's Alexa [31] in which knowledge about concepts are added to the assistants. In the case of Alexa, these are called "skills" which denote external endpoints the user can interact via the natural language interface the assistant provides. For example, to get weather information the assistant is programmed to recognise phrases which query the weather and then ask an external API for the weather information. Finally, the temperature information is formed part of the response to the user.

This structure of adding knowledge manually limits the cognisance to the ability of the programmers; however, retains great control over the assistants' capabilities. The challenges incorporated is to understand what the user is looking for and search the external resources attached for the answer [32]. For example, beyond the external APIs and search documents, attempts to connect the semantic web into an assistant have been made [33]; "the agent has a modular knowledge organisation composed by four differentiated areas: the rational area, which adds semantic web knowledge, the association area, which simplifies building appropriate responses, the common-sense area, which provides common-sense responses, and the behavioural area, which allows IPA agents to show empathy."

Similarly interfacing with search agents such as Google, agents were able to discover unknown concepts. A 3D-embodied character personifying the fairy tale author Hans Christian Andersen (HCA), for example, could recover the ontology around the word "quake" using Google engine and recognising it is a game [34]. These connections vastly expand the knowledge base of the assistant and increase the areas in which it can act.

2.3.3 Retaining Knowledge

Another important aspect of an assistant is to retain the knowledge it can harness during its operational lifetime. Frequently asked questions, user preferences, assumptions about the world and past conversations can all extend the cognisance of the agent if retained. This malleable nature of the agents knowledge base will allow it to adapt to different users and domains. The biggest advantage of a multi-user assistant is that it can harvest information from millions of users at the same to benefit millions of other users.

Learning based on previous experiences is widely observed in nature. In the context of personal assistants, incorporating case-based memory by exploring models of memory have allowed agents to retain a local memory, MemoPA is an example of such an agent [35, 36]. This approach allowed the system to *remember* certain interactions and use them in future cases. There are multiple memory models on which such agents could be built and they offer various trade-offs [37].

When acquiring information, inconsistencies can arise with the retained knowledge. Especially, when the domain is large enough, the agent can confront contradicting information. Default reasoning [38] can allow the agent to make consistent assumptions with the retained knowledge. In the context of natural language processing, there have been attempts at incorporating default reasoning to acquire and retain a consistent view in the stream of incoming knowledge [39]. However, most approaches have again limited the domain to cases which can be handled and investigated manually. The ability to create large scale knowledge bases automatically from active user interactions is still an active area of research.

2.4 Virtual Assistant Architectures

There are many possible ways to implement a digital assistant depending on its requirements and it is often a matter of software engineering to design the most suitable architecture for one. In this section we look at existing digital assistant's architectures and how their key points influenced the design of Kevin's multi-stage modular pipeline back end architecture.

2.4.1 Attorney 209: Adviser for family-based cases

Attorney 209 is an expert system that mimics a real life lawyer and gives legal advice. It is a conversational chatbot which is designed to handle initial assessment for families who need legal guidance regarding family cases such as child custody and legal separation [40]. The domain of law is fixed to an even finer context of family cases in order to limit the knowledge base from which answers are drawn creating an effective domain-specific assistant.

Figure 2.4 portrays the overall architecture of Attorney 209. We observe sub components which build up the assistant as a whole. However, the flow is dictated from the user query towards answers and repeats in a very structured manner. The natural language processing unit along with the knowledge base sits at the core to process user queries. At-

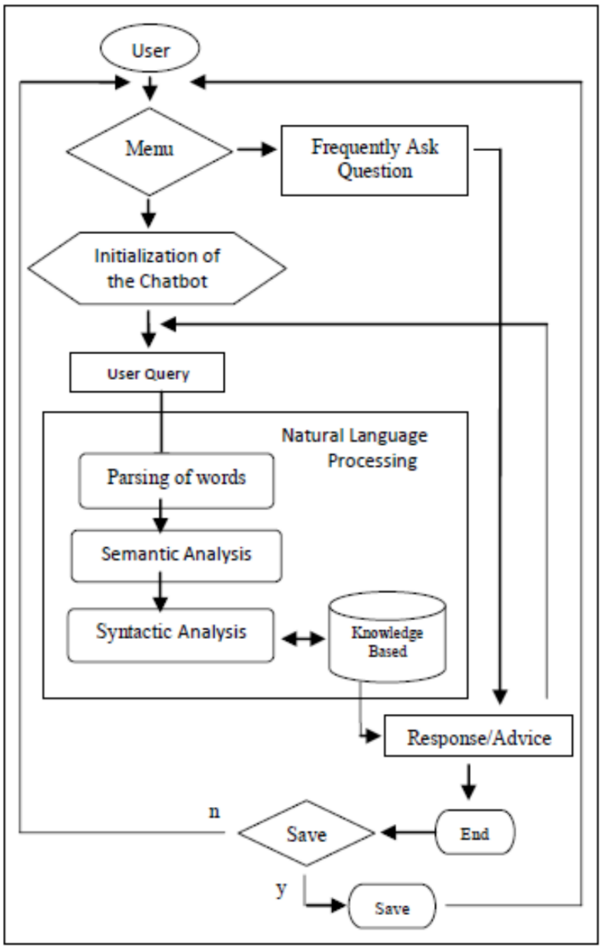


Figure 2.4: Attorney 209 architecture overview.

torney 209 incorporates the Chart Parsing Algorithm [41] in order to process the queries. The structure follows a mostly linear pipeline in which user queries are fed and answers are computed.

The architecture portrays what is called a *separation of concern* approach in which the core components are broken down into smaller more specific tasks moving away from a monolithic architecture. In this case, the parsing of words, semantic and syntactic analysis together with the knowledge base are separated. Kevin takes this idea at its core to separate out the natural language processing, reasoning and the knowledge base.

2.4.2 Project Execution Assistant (PExA)

Now we look at an assistant which features a natural language dialogue interface as well as external resources. Project Execution Assistant, PExA, has been developed to improve the productivity and effectiveness of a knowledge worker by aiding her in organising and performing tasks focusing on two key areas: time management and task management [42]. PExA aids the user in scheduling meetings, setting reminders and organise tasks that may involve around the subject such as expense reimbursement. The domain in this case is the office environment with the context surrounding daily user tasks. PExA brings four several **desiderata of assistance** to render the assistant useful and usable:

- `Directable:` Although the assistant should be capable of operating in an autonomous manner, it must accept explicit directions from the user on what to do (or not) and how to do it.
- `Personalizable:` The assistant should learn a model of the user's preferences and adapt its behaviour accordingly.
- `Teachable:` It should be easy for the user to communicate new or modified problem-solving knowledge to the assistant over time.
- `Transparent:` The assistant should be able to communicate succinctly what it is doing and why, in order to provide the user with insight as to the status and strategy of its actions.

PExA frames ambitiously what the goal of an assistant would be, what are its core traits that build its assistive capabilities. The principles stated above form a basis on which all assistive technology could stem from. PExA as with Radar, the agent email and task management [22], not only can draw information from outside sources but also interact with the user in a natural language dialogue under the directable requirement.

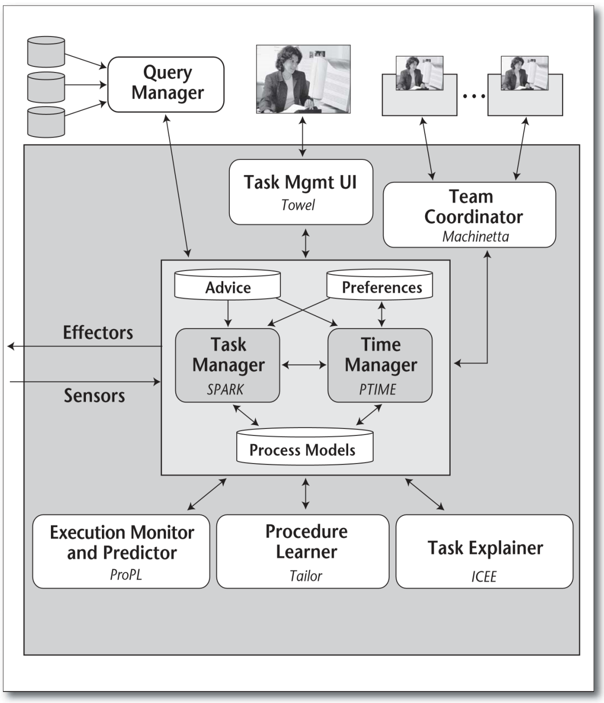


Figure 2.5: Project Execution Assistant (PExA) architecture overview.

With the Project Execution Assistant (PExA), we observe more sub components that build up the assistant. Figure 2.5 shows the different components and how the desiderata form a partitioning on the components. The architecture follows a *divide and conquer* approach to fulfil the requirements of the assistant. The task and time managers are distinguished at the core (SPARK and PTIME), learning capabilities are handled by the procedure learner (Tailor), transparency is achieved by the task explainer (ICEE) and the overall dialogue is maintained by the execution monitor and predictor (ProPL). This structure maintains manageable sub components in order to achieve the complex behaviour observed as a whole. Such an architecture provides insights into scaling towards larger, more complex systems by dividing the assistant into its assistive components.

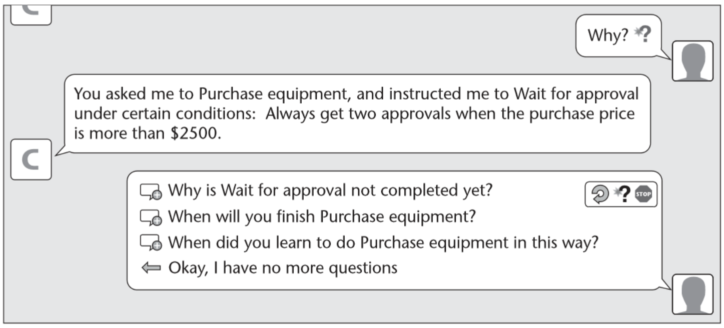


Figure 2.6: Project Execution Assistant (PExA) follow up dialogue interaction.

Finally, with figure 2.6, we observe that PExA can sustain a dialogue with the user incorporating follow up context sensitive questions. The assistant is able to answer the question "Why?" with detail stating the acting conditions explaining its *reasoning* to the user. Although the options to the user are fixed thereafter, the dialogue provides an easy to understand picture of the inner workings of PExA's reasoning. Fixing the options for users response helps the flow of the conversation and avoid ambiguity from the assistants perspective. The domain to create such a conversation is explicitly built upon the process of purchasing equipment.

2.4.3 Cloud Scale Assistant Architectures

There are also significant technical challenges if the personal assistant is to scale and serve several thousand users at the time. Similar to the likes of Apple's Siri, large-scale personal assistant deployment must endure hundreds of queries a second, maintain fast response times and balance on device computation with cloud requests [43]. At such scales we observe more sub components that build up the complexity of the assistant with more focus on performance. Thus, not only the sub components divide what computation happens but also where that computation is performed.

As seen in figure 2.7, one approach is to separate the computational concerns of Au-

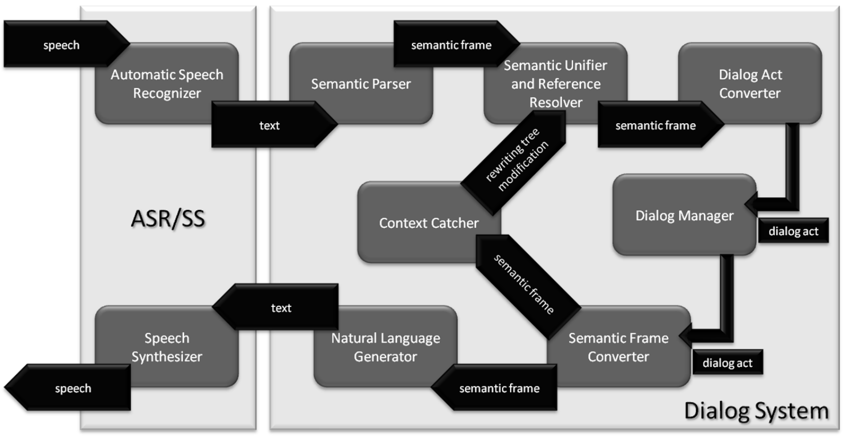


Figure 2.7: A generic cloud-based personal assistant architecture.

tomatic Speech Recognition (ASR/SS) from the Spoken Dialogue System which communicate via text [44]. This architecture separates the concern of heavy computation and allows similar sub components to be partitioned. Looking at an open-source assistant Sirius, an open end-to-end stand-alone speech and vision based intelligent personal assistant (IPA) similar to Apple's Siri [45], we can observe the features function at scale.

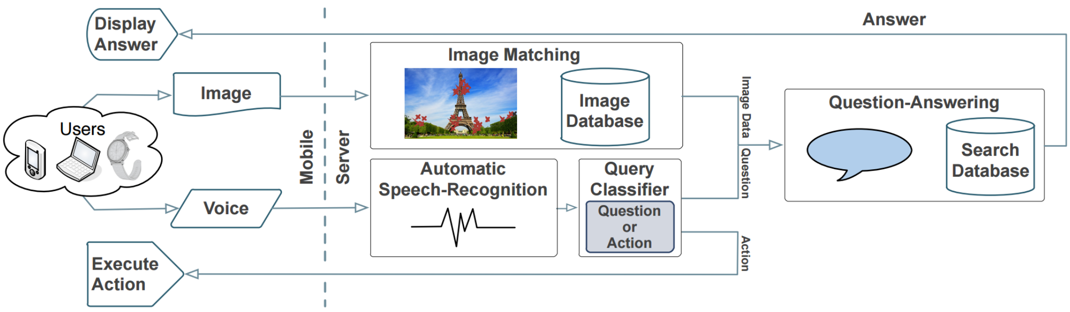


Figure 2.8: Sirius architecture overview with mobile and server components.

In figure 2.8, we clearly see the distinction made between mobile and server components that comprise Sirius. The interaction sub components such as recording the voice is processed on the mobile and the heavy weight computations such as image matching are performed on the server side. This architecture creates a lightweight front-end for the user while maintaining computationally expensive features of the assistant in the cloud. The idea of having lightweight interfaces is emphasised with Kevin as the actual assistant is provided as a service on which user interfaces can be built.

This scaling is crucial in order to expand the user base in a computationally effective manner. Performance aspects of scaling will help us decide over certain algorithms due to runtime costs and how they affect the response time of Kevin. The depth of the pipeline also plays a significant role in reducing the answer calculation time. However, despite user customisation options that the assistant can retain, serving a single assistant to thousands of users renders it *generic*.

2.5 Personalisation & Gender Issues

As assistants start to scale to serve multiple users, they become generic and standard as every user's experience becomes similar. Although the cloud scale deployments require the same assistant serve all the users, it creates a necessary room for personalisation based on user interaction. Ideas presented here influenced heavily the perception of Kevin and how to make it unbiased over such topics.

2.5.1 Interaction design

A key concept in personalisation is *how* the user interacts with the assistant. These interactions could shape the means by which an agent can adapt to the user and change its behaviour accordingly. Interaction design has been considered carefully for many applications. For example, in the hand-held mobile assistant devices era, the Personal Assistant for online Services (PALS) project aimed to develop an intelligent mobile interface for quick and accurate task performance with mobile commerce web sites [46] using case-based design [47]. They outlined 4 main points of interaction with the device and they draw important concepts into the conversational assistants domain:

- `Direct (D)`: The user requires the agent to *do* the requested action. Direct interaction often encompasses voice commands; for example, "turn on the lights" is a direct command to the agent.
- `Solicited (S)`: The agent *helps* the user in a certain task. The interaction follows the agent working with the user to provide guidance. For example, "How do you cook pasta?" will allow the agent to help the user make food by providing steps to achieve their goal but not perform them itself.
- `Non-solicited (NS)`: In this interaction, the agent provides assistance even though the user might not have explicitly requested it. This situation might arise from the fact that the agent might have better domain knowledge on the topic than the user. For example, suggestions on restaurants, routes to take due to traffic can be proposed to the user.
- `Independent (I)`: Finally the agent can act independently from the user. For example, if the agent knows watching a movie requires dimming the lights, it can do so without interacting with the user about lights at all. With such interaction, the agent often performs *obvious* actions such as opening the garage door if the user is driving back home and is about to arrive.

These generic means of interaction apply to all agents; however, there are other means of augmenting the experience. One of which is *embodiment*. Embodiment have been research in cognitive sciences intensively [48]; however, in the context of assistants the concept often yields a tangible face to interact with for the agent. In the aim of creating a natural interaction, moulding a virtual 3D character or a robot puts a tangible face behind the assistant. This allows users to concentrate on the assistant when interacting, looking at the assistant, observing gestures all improve the flow of the interaction significantly. This embodiment has proved crucial in maintaining a positive environment with the assistant. For example, a natural human computer interaction paradigm was proposed for people with cognitive impairments such as Alzheimer's Disease [49]. It consisted of using a realistic virtual character, rendered on a common television set, to play the role of a virtual personal assistant that showed reminders, notifications and performed short dialogues with the users. With simple Yes/No type questions, the test results showed that all users engaged naturally with the avatar.

With the rise of virtual reality, some research even attempted to create complete virtual environments to place the user in the realm of the digital assistant rather than placing the assistant in the real world. One example is the personalised assistant in a virtual shopping environment giving suggestions based on users' past purchases using data-mining techniques [50]. No matter what the substrate, the interaction between the agent and the user proves critical in providing a natural personalised experience.

2.5.2 Social environment and expectations

Placing the assistant in a physical world with human beings is a challenge on its own. In particular, voice activated agents in social environments such as a household living room come along with both technical and social obstacles. A recent study on understanding the implications of voice activated assistants provided insights into how and when users preferred to interact [51]. "Results showed that participants preferred using the VAPA in a private location, such as their home, but even in that environment, they were hesitant about using it to input private or personally identifying information in comparison to more general, non-private information." These results show the difficulties that might arise from the environment the assistant is placed in as well as what information it requires about the user to be active.

Bob:	Are you a pilot?
ALICE:	What kind of female do you think I am?

Table 2.4: Gender portrayal of ALICE.

As seen in table 2.4, the gender of the assistant is expressed explicitly with assumptions made on possible connections between females and pilots. This situation raises very important questions surrounding the human social elements in shaping a virtual assistant:

what are our *social expectations* of one? It has been debated in the press recently due to female characterisation of most commercial agents such as Apple's Siri [6] and Amazon's Alexa [31].

Some research have attempted to unearth some of these biases and social expectations. For example, a study has been made to understand this gender preference via social attraction theory [52]. By asking participants to choose between virtual assistants that differed on tested aspects such as gender, they found female virtual assistants were preferred over male assistants with almost 70% of all participants choosing a female assistant. This study shows the delicate nature of creating agents that will have a role in the social environment.

2.5.3 Becoming human-like

Ultimately, the quest for true personalised virtual assistant might have to portray pungent human-like characteristics, going back to the Turing Test [10]. In most cases, a differentiating factor is the lack of emotion. However, some studies have pushed that boundary even further. By using basic emotions theory which dictate emotions consists of eight prototype basic emotions and constructing a psychology model, virtual assistants capable of empirically obeying the human emotion rules were built [53]. This feat was achieved by varying the parameters of the emotion model used to simulate different human psychologies.

Another study encompasses artificial hearts which is a joint term for the agent's emotion and personality. The study attempted an approach to enable the computational perception required for the automated generation of affective behaviour in multi-agent real-time environments [54]. The results showed that agents can communicate not only knowledge but also affective attitudes about the knowledge at hand. These systems lead the way on truly human-like assistants and provide insights into elusive concepts such as emotion.

However, from the perspective of this project and practical digital assistants, we are not concerned with creating a human-like agent. In most cases, the interaction model assumes the user acknowledging the assistant as another computer program with a natural language interface.

2.6 Language Representations

One of the key concepts of enabling logic based systems to handle natural language was through strict conversion between the two. There have been multiple representations of natural language in order to be processed by computers. In this section we will focus on representations that allow sentences to be de-constructed into constituents which provide the meaning of a phrase.

2.6.1 Combinatory Categorical Grammar (CCG)

Combinatory Categorical Grammar is a form of lexicalized grammar in which the application of syntactic rules is entirely conditioned on the syntactic type, or category, of their inputs [55]. Thus such a grammar can recursively build what a sentence *means* by looking at its constituents. It follows a divide and conquer approach in which complex structures are broken down into more simpler and manageable sub components.

$$
\begin{array}{ccc}
\text{Marcel} & \text{proved} & \text{completeness} \\
\hline
NP_{3sm} : marcel' & (S\backslash NP_{3s})/NP : \lambda x \lambda y.prove' xy & NP : completeness' \\
\cline{2-3}
 & \multicolumn{2}{c}{S\backslash NP_{3s} : \lambda y.prove' completeness' y} \quad > \\
\hline
\multicolumn{3}{c}{S : prove' completeness' marcel'} \quad <
\end{array}
$$

Figure 2.9: CCG proof of "Marcel proved completeness" using lambda calculus.

As seen in figure 2.9, CCG can decompose the sentence and extract arguments to provide a *proof* that allows the constituents to yield a valid sentence using lambda calculus as the base formalism. In this case, the verb `prove` is a function application with two arguments which when given, provide the meaning of something proving something. When such a decomposition works, a computer program has most of the information needed to process the input: the action, subjects and entities. If the construction does not work, for example if the expected arguments are not matched, then the sentence can be considered not to make sense.

From our perspective, we focus on the idea of representing natural language phrases as structured trees and use such concepts throughout the project to build a more flexible representation. We would like to minimise the amount of pre-processing in order to consume natural language but still achieve a flexible representation that does not flounder when faced in unknown constructs.

2.6.2 Abstract Meaning Representation (AMR)

Abstract Meaning Representation [56] captures “who is doing what to whom” in a sentence. Each sentence is represented as a rooted, directed, acyclic graph with labels on edges (relations) and leaves (concepts) [57]. This representation captures many relationships between verbs and nouns in a very similar fashion to a dependency graph. When constructed correctly via a parser, AMR provides upstream natural language processing a clean representation of complex sentences. Similar to CCG, the aim is to deconstruct natural language into consumable structured constructs for a computer program.

```
(w / want-01
  :ARG0 (b / boy)
  :ARG1 (g / go-01
          :ARG0 b))
```

Figure 2.10: PENNMAN representation of AMR on "The boy wants to go."

The representation as seen in figures 2.10 and 2.11, captures the relationship between what the *boy* wants. It is important to observe the tree structure of the sentence, as it enables to be compositional; multiple trees can be combined into a larger to represent a longer sentence for example using conjunction. Therefore more complex sentences can often be represented using divide and conquer relying on the principle of composition.

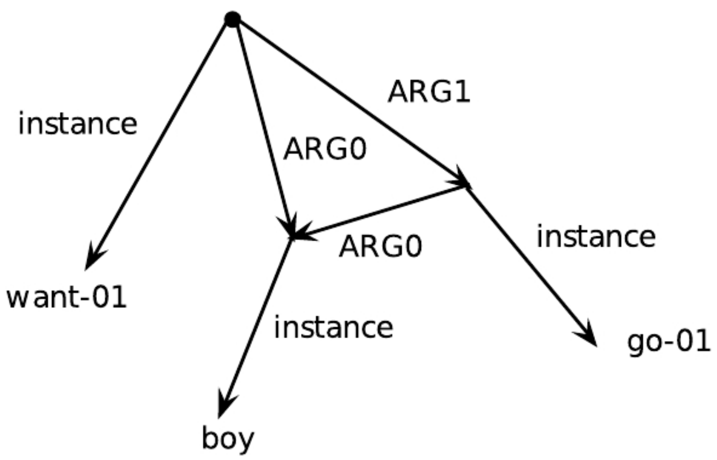


Figure 2.11: Graphical representation of AMR on "The boy wants to go."

With such a strict structural representation, logic based systems can be developed to process and understand natural language. A prominent usage was demonstrated by applying inductive logic programming [58] in order to tackle the toy context-question answering tasks created at Facebook [59]. This work showed that if the representation is well founded and can be parsed into logic, it can outperform neural network based solutions. This result is expected since the computer system in this case has all the information in a clean format. The challenge is not in upstream applications but in creating robust parsers that can match the performance neural networks in unseen examples.

Chapter 3

Background

The background chapter covers concepts that are directly used in this project for creating Kevin. The ideas and projects presented together with the related work (chapter 2) help shape Kevin as a digital assistant and its tangible foundations. We will look at how we can define Kevin as an assistant, the logic programming as the starting point for a new framework and finally some deep neural network natural language processing techniques used in Kevin.

3.1 From Conversations to Assistants

When we look at the definition of an assistant, "a person who helps in particular work", we take note of the keyword *help*. It is this crucial difference that started to reshape conversational agents that can talk into assistants. Now we try to better define what an assistant is in the context of a computer program and the modern perception of a digital assistant.

3.1.1 Personal Digital Assistant, the Definition

One common understanding of a personal assistant is that of a person (or an agent) who is able to provide distinct help at a given time and in a given activity context [44]. The assistant's primary purpose is to aid the user in numerous ways:

- **Answer informational queries:** One of main driving forces behind the popularity of personal digital assistants is to serve as a gateway into the vast information available on the internet [32]. These systems provide a natural language interface in the form of queries such as "What is the weather in London?" to which the answer is the current temperature information. This ability put mainstream digital assistants under the spotlight to tackle the information overload induced by the internet.

- **Perform everyday simple tasks:** Another crucial feature a personal digital assistant provides is convenience in handling simple daily tasks such as setting reminders, organising calendar and email [60]. These tasks are often pre-determined range of tasks that the agent *recognises* and executes the coded task often interfacing with other external APIs such as booking systems for restaurant reservations. These features gave personal assistants their *practical* perception, getting things done easier approach.

- **Advise on optimal planning:** Harnessing the power of computers, the agent should in cases advise the user with optimal planning. For example, asking for directions considers live traffic information [5], setting calendar events while automatically checking for overlaps. In such instances, the assisting agent can provide advise on making the best possible planning.

These features often form a basis for the modern view of a personal digital assistant. Combining multiple capabilities from different categories can create a *general* assistant in which the agent can portray these features in various contexts. If the context is limited, however, then it is considered a *domain-specific* assistant designed to offer functionality on a subset of a certain domain.

When building Kevin, we try to allow these capabilities to be implemented. In other words, our framework design should not be the cause for failing these tasks. As a result, the framework was designed to be flexible in order to accommodate external functionality and cover extra features with the help of external components.

3.1.2 Pragmatism in Agents

Besides the capabilities that a personal digital assistant can maintain, its attitude also forms part of the definition. Assistants, as discussed in this paper, sustain a practical view in aiding the user. The focus is on providing functionality in a natural interaction, not creating a human-like natural agent. To clarify further, the agent does not have to portray human attributes. There have been active research in that area such as creating systems that can empirically obey human emotion rules by simulating different human psychologies [53]; however, from a pragmatic perspective functionality is the primary concern.

Often the agent is designed to complete certain processes in a convenient manner for the user. The very first of such basic assistants were *command and control* agents in which pre-determined voice commands corresponded to certain tasks that the agent can coordinate such as the actions of a robot [61]. The style of command and control agents still find way into many assistants as a reliable way to cover many basic tasks to the extent those tasks can be pre-programmed. One prominent usage case in modern assistants is to place Easter eggs that trigger when certain utterances are recognised.

Therefore, Kevin's interactions lean towards achieving a practical goal such as providing answers to information queries rather than natural dialogues. We assume the user recognises Kevin as a computer program and do not attempt to mimic human conversational models.

3.2 Extracting Structured Data

A common theme in using natural language as an interface is to convert phrases into categorised entities or chunks. This transformation often encapsulates many techniques available in a natural language processing libraries such as named entity recognition. Although the field of Named Entity Recognition and Classification (NERC) started with hand-written rules, developments in machine learning techniques have yielded robust results [62]. Combined with the domain knowledge, many conversational user interfaces convert these entities into structured data to be consumed by other computer programs.

`Schedule lunch with Mary Johnson at 12 pm tomorrow.`

action:	meeting.create
name:	Lunch with Mary Johnson
invitees:	Mary Johnson
time:	2014-08-06T12:00:00-07:00

Table 3.1: Extracting structured data using API AI.

As seen in table 3.1, the API.AI [63] library can extract entities from a given phrase and convert into categorised actions with the appropriate context. This structured information is consumed by upstream software creating an natural language interface for existing software. Depending on the action category, different entities are extracted to create the *context* of the phrase. For example, creating meetings have different entities than navigation tasks.

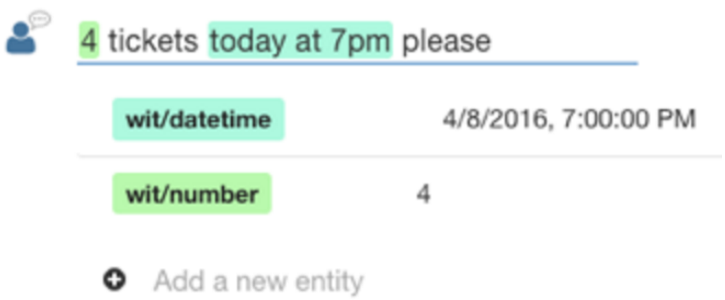


Figure 3.1: Entity recognition and variable assignment in Wit AI.

One common approach to imposing structure on natural language phrases is to assign variables into pre-determined phrases. Figure 3.1 captures the implementation from Wit AI [64] showing how the users can assign variables with certain types such as date and time or numbers. These types eventually form the basis of the extracted structure from the phrase. Rest of the phrase chunks such as "please" are discarded in this scheme. The process requires a lot of human input in order to *tell* the system what to look for and such a process can still exist at a larger scale as seen in the insurance chat-bot powered by IBM Watson [65]. Despite its shortcomings, extracting structured data from natural language is at the centre of intent parsing and as a result the sentence variable approach will present in Kevin as well.

3.3 Logic Programming

Logic programming is a programming paradigm based on formal logic. It is most commonly used in its declarative form in which rules and facts are presented in predicate logic and a control mechanism to prove certain queries [66]. Although there are many variants of logic programming with different implementation and semantics such as Prolog [67], from our project's perspective we will focus on a generic logic program and understand the basic constructs of one.

$$head \leftarrow body \tag{3.1}$$

All logic programs built on the following basic construct of a rule 3.1. The rule often expresses a material implication; however, material implication semantics are not necessary. Depending on the formalism used, the head and body can contain expressions of varying degree of complexity. Often the meaning suggests, if one can prove the *premises* in the `body` then one can conclude the `head`. The head or the body of a rule can be omitted to give constraints and facts respectively. The empty section can be thought of being `false`, thus facts become vacuously true and constraints imply false when all premises (constraints) are satisfied.

$$\begin{aligned} animal(X) &\leftarrow cat(X) && \text{(rule)} \\ cat(sally) &\leftarrow && \text{(fact)} \\ &\leftarrow cat(X), dog(X) && \text{(constraint)} \end{aligned}$$

In the example logic program rule, we state that all cats are animals; this meaning is captured by the *variable* `X` which can denote an entity in this case. Thus, whatever is a cat also is an animal. In the fact, we explicitly say that `sally` is a cat and that case is vacuously always true since the body is empty. This result is an effect of material implication in that `false` implies anything is considered `true`. Finally, we can specify a constraint to say that nothing can be a `cat` and `dog` at the same instant. Following this setup, we can deduce that `animal(sally)` using the rule without violating the constraint.

We will not consider *time* in this project and take every rule to represent the same instant. Although there are methods such as event calculus to represent discrete time within logic, in the context of a digital assistant we decided to not consider it due to its complexity. A common approach is to use predicates similar to `happensAt(X, T)` that specifies an event X occurred at time T. It is very difficult to time-stamp events in natural language and therefore Kevin only considers the present.

3.3.1 Negation by Failure

Negation by failure can be viewed as another inference rule but one that is *non-monotonic* often represented as `not` A. Though the semantics of it depends on the formalism used, we will follow a simple approach and detail it as *all attempts to prove fail* [68]. Therefore, the semantics can be considered as: if it is not known to be not the case then it might well be the case.

$$\begin{aligned} slow(X) &\leftarrow car(X),\ \text{not}\, fast(X) \\ car(bmw) &\leftarrow \end{aligned} \tag{3.2}$$

When we declare a setup as seen in program 3.2, we stated that every car is slow unless it is known to be fast. Thus when required to show whether `bmw` is slow, the program succeeds as it is not known to be that `bmw` is fast in the given context.

Negation by failure construct allows us to provide a way of specifying what to do in the absence of provable information; it can be used to specify defaults such as every car is normally slow. Such reasoning is often associated with *common sense* which are integral to our reasoning in our daily lives. For example, we might assume that an apple is by default red unless it is known be green for a shopping assistant until the user specifies they want green apples. Rules with negation by failure are called normal clauses and logic programs that use them are referred to as normal logic programs [68].

3.3.2 SLDNF Resolution

Selective Linear Definite (SLD) clause resolution and with added negation by failure, the SLDNF resolution provide an operational semantics to normal logic programs with negation by failure that follows a top-down (goal-oriented) approach. The computation of a goal query G follow a series of derivation steps that unify the goal with rule heads and try to prove the body as seen in figure 3.2. There are other semantics such as Answer Set Programs (ASP) [68]; however, we will follow Prolog style interpretations for their operational semantics.

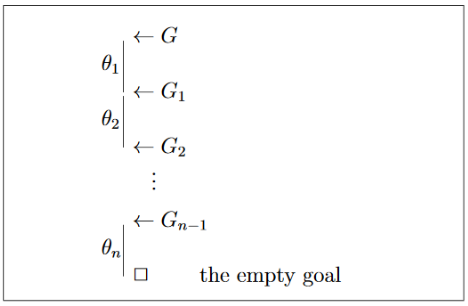


Figure 3.2: SLDNF operational semantics of goal oriented proof.

There are two kinds of derivation steps which are selected based on whether it is a negation by failure literal `not` L or positive literal L. In the case of L, the body of a rule $L \leftarrow B$ is replaced with L unifying if necessary using θ_i; in the case of `not` L the premise is removed if all attempts to prove L fail finitely. These derivation steps in return transform the initial goal G into sub goals G_i that need to be proved forming a proof tree.

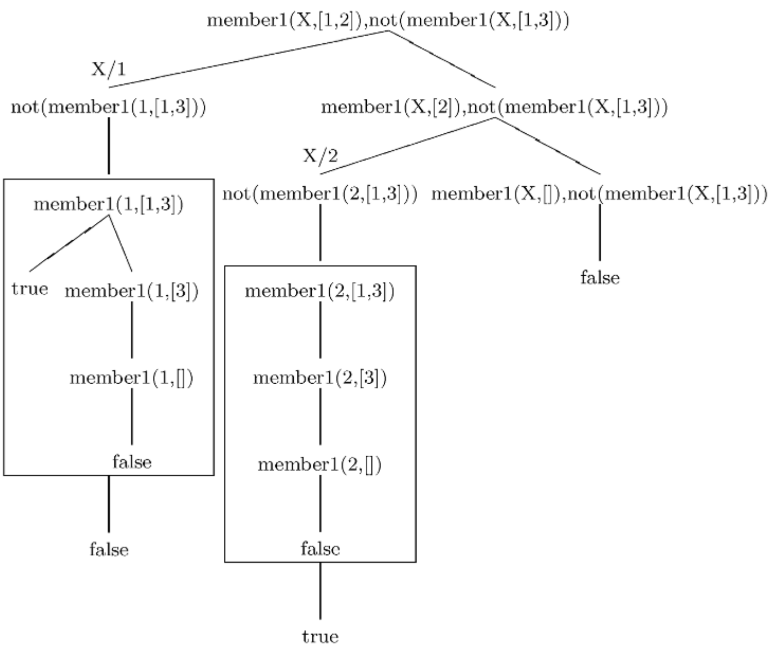


Figure 3.3: Example search space of SLDNF on list membership.

Following the computation in the figure 3.3, we observe that the proof of a given goal G constructs a *search space* in the form of a tree that the program explores until it reaches `true` or the empty clause $\square$. In the most simplest case a depth first search algorithm could resolve the goal. We will re-encounter tree like search spaces in Kevin, particular when trying to handle negation by failure constructs in natural language.

3.4 Deep NLP

Deep neural networks provide a method to learn complex non-linear functions by layering multiple connected neural units such as densely connected perceptrons. In the case natural language processing, neural networks yield incredible results that mainly work on vector representations of words of a given corpus. In this section we discuss the techniques that allow Kevin to handle natural language input.

Most models rely on the distributional hypothesis popularised by Firth: "a word is characterised by the company it keeps" [69]; words that occur in similar contexts often exhibit similar meanings. This idea lies at the foundation of vector space representations of words and how context aware dense vector representations can be learnt using algorithms based on distributional hypothesis.

3.4.1 word2vec

Word2vec is a collection of algorithms that are used to produce dense vector embeddings of words based on the distributional hypothesis [1]. Given a large corpus, word2vec maps every word W in the corpus into a d dimensional vector $f : W \to V^d$.

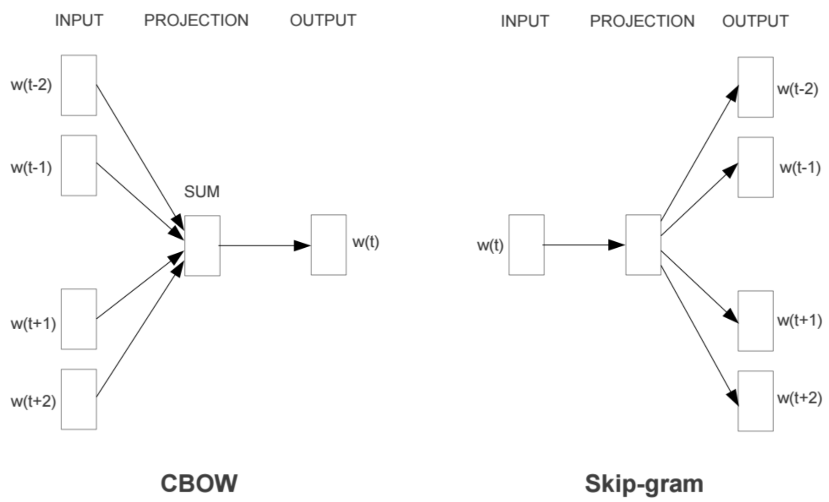


Figure 3.4: Continuous bag-of-words and skip-gram models presented in word2vec [1].

There are two main approaches: the continuous bag-of-words model and the skip-gram model in which a centre word is predicted given context or vice versa respectively. Given context words w_{t+j} and a centre word w_t the models try to predict either the context or the centre word and try to maximise the probability of success. As an example, we will consider the skip-gram model.

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m<j<m} \log p(w_{t+j}|w_t) \tag{3.3}$$

The skip-gram model tries to predict the context words of a given centre word w_t. The objective function 3.3 tries to maximise the log probability of a context word $p(w_{t+j})$ across a window radius m given the centre word w_t. This is done for every token t in the corpus T and then normalised across. The derivative with respect to the model parameters $\nabla_\theta J_t(\theta)$ given centre token t, can then be used to train the network using stochastic gradient descent (SGD).

$$p(outer|centre) = \frac{\exp u_o^T v_c}{\sum_{w=1}^{V} \exp u_w^T v_c} \tag{3.4}$$

One way of representing probabilities of words is to use the `softmax` of the dot product of word vector embeddings as portrayed in equation 3.4. In this case, u and v are two vector embeddings for words, i.e. the associated vectors of words that the model is trying to refine. Thus, the **similarity** of words are captured by the dot product of their corresponding vectors. However, since we consider $2m+1$ words at a time using a sliding window, the derivative $\nabla_\theta J_t(\theta)$ is sparse, thus cause sparse updates that are not very efficient.

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{j \sim P(w)} \log \sigma(-u_j^T v_c) \tag{3.5}$$

The loss function of the skip-gram model with negative sampling, equation 3.5 adds noise to reduce the probability of some random words $j \sim P(w)$ and train the network accordingly. Often the final vector representation of a word is the sum of its corresponding embeddings $u_w + v_w$.

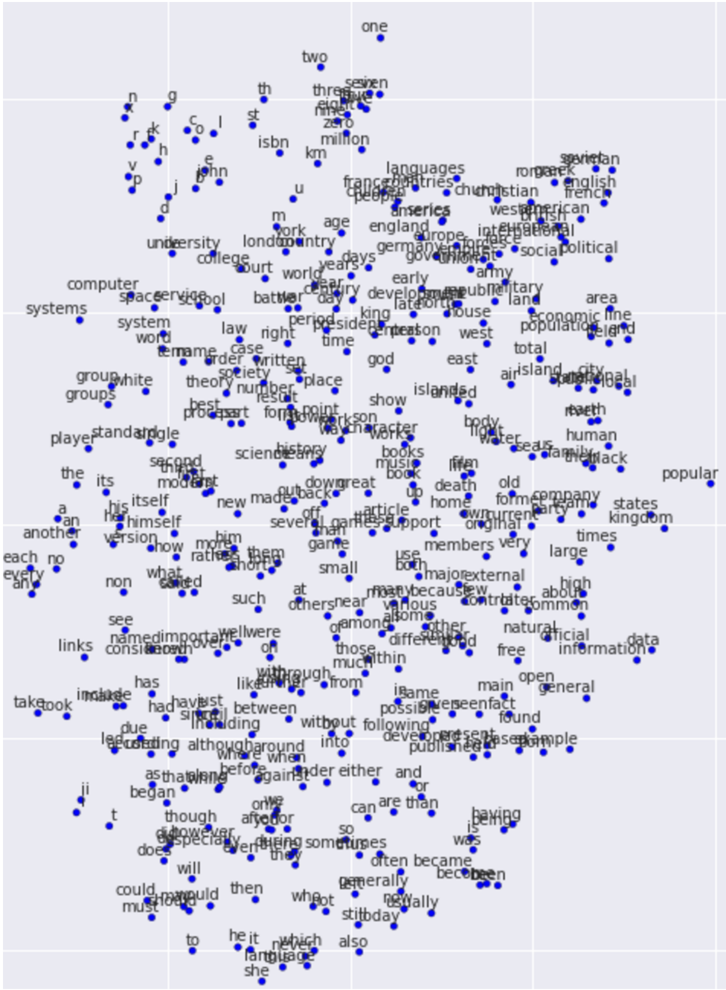


Figure 3.5: Word2vec results from TensorFlow [2] example implementation.

Figure 3.5 shows the results from the example implementation[1] in TensorFlow demonstrating how similar words, i.e. words that appear in similar contexts cluster together. For example, numbers such as one, two, three clustered at the top centre top of the figure as well as other words that represent numerical value such as million.

3.4.2 Global Vectors (GloVe)

A common problem with models such as word2vec is that lots of time spent scanning context windows to learn a distribution for frequently appearing words such as `the`. This not only adds a bias towards frequently appearing words but also lacks training for rare words. Global Vectors (GloVe) [3] aim to address this issue by introducing weighting into the objective function. As such, the GloVe model is used as the de facto standard of

[1] https://www.tensorflow.org/tutorials/word2vec accessed 2017-05-23

vector representations for upstream natural language networks.

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{V} f(P_{ij}) \left(u_i^T v_j - \log P_{ij}\right)^2 \tag{3.6}$$

The objection function 3.6 tries to get the dot product of vector embeddings closer to the probability of word i co-occuring with word j, P_{ij} calculated from the co-occurrence matrix of the corpus. Often, the weighing function f tries to minimise the negative effect of frequently occurring words in the corpus. The function in the paper is given as $f(x) = min(g(x), 1.0)$ in which $g(x)$ is a function that increases with x, thus the weight is capped at 1. This approach scales to very huge corpora and uses statistics such as the co-occurrence of words very well.

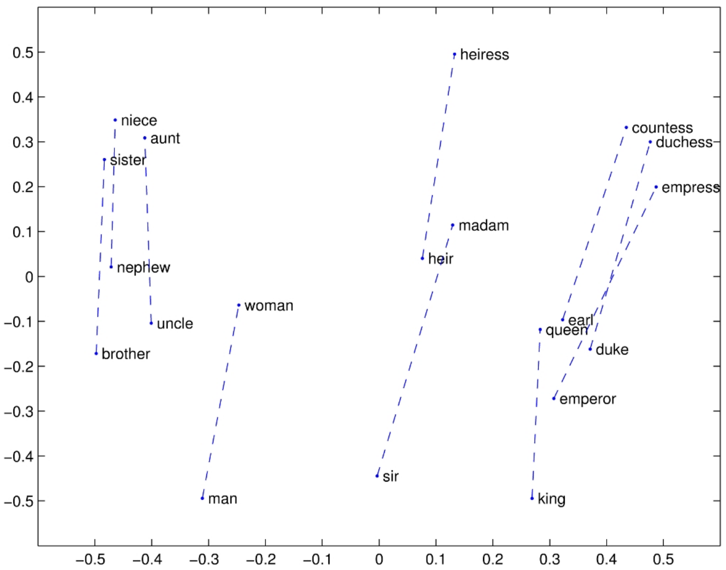


Figure 3.6: Superlative linear structures in learnt vectors from GloVe.

As seen in figure 3.6, the vector representations can not only capture similar semantics but also linear relationships between words. These vector embeddings provide the basis for many upstream natural language networks and play a crucial role in Kevin allowing it *understand* similar phrases without needing extra hand written rules.

3.4.3 Long Short-Term Memory (LSTM)

A big problem with neural networks even with Recurrent Neural Networks (RNNs) is to capture long term dependencies. Long short-term memory networks [70] are a special kind of recurrent network designed to handle long-term relationships, patterns in data. These recurrent network units are useful in understanding co-relations of words in sentences and are frequently used in upstream natural language processing neural networks to capture context within a given phrase or paragraph.

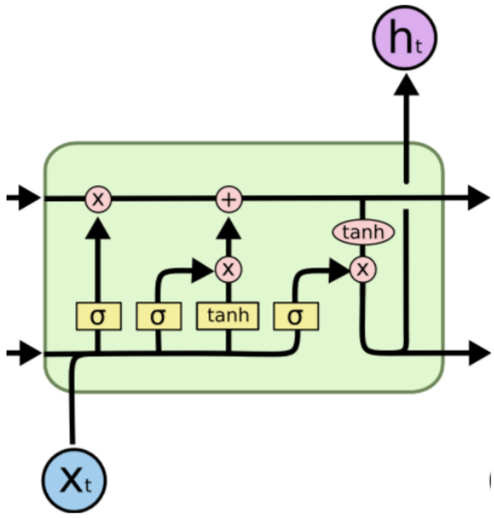


Figure 3.7: A representation of an LSTM unit.

Figure 3.7 shows a representation of an LSTM unit[2]. The upper horizontal line represents the internal state C_t which the unit updates depending on the previous output of a unit h_{t-1} and the current input X_t. The two operations to the internal state, multiplication and addition represent what information needs to be forgot and added respectively. This update of internal state allows long term dependencies to be remembered and passed along the chained units within an LSTM layer.

3.4.4 Bi-Directional Attention Flow (BiDAF) Model

The BiDAF model is a hierarchical multi-stage architecture for modelling the representations of the context paragraph at different levels of granularity [9]. It builds on vector embeddings as well as character embeddings to answer questions trained on the SQuAD dataset [71]. The model achieved an Exact Match (EM) score of **68.0** and a F1-score of **77.3** on the SQuAD dataset. We use a shallower, narrower version of this model in order to extract answers from given context sentences.

As seen in figure 3.8, the architecture comprises of 6 layers. The input to the network is a context sequence $\{x_1, x_2, ..., x_T\}$ (ex. a paragraph or a sentence) and a query $\{q_1, q_2, ..., q_J\}$. The output of the network is the start and end indices of the answer within the context.

1. **Embed layers** transform the context and the query into their respective word and character embeddings. The word embedding used is the GloVe model (sec-

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ accessed 2017-05-17

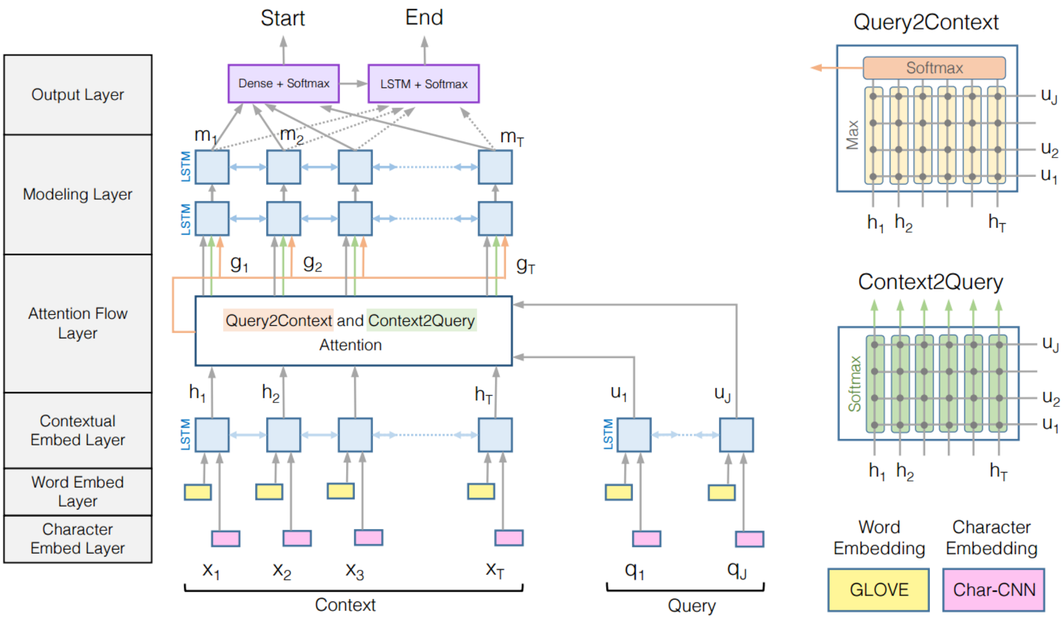


Figure 3.8: BiDAF model architecture overview.

tion 3.4.2). The character embeddings allow a finer granularity of representation and often handle mixed punctuation, misspellings better than word embeddings.

2. **Contextual layer** recalculates embeddings of words and characters with respect to the given context or query. This layers enables words to acquire meaning, a new vector representation based on what was before and after them in the paragraph. Hence, a bidirectional LSTM is used to capture these relationships.

3. **Attention flow layer** calculates weights of context and query representations with respect to each other; $S_{tj} = \alpha(H_{:t}, U_{:j})$ matrix captures the *similarity* of context item t with query item j. The α can be any similarity metric, in the paper a dense layer with the concatenation of inputs is used. The shared similarity matrix is then used to compute `Query2Context` and `Context2Query` attentions that capture relationships between the query and the context and vice versa.

4. **Modelling layer** feeds on the original context embeddings and the attention representations to model the query answering problem and find the most likely words in the context that might constitute the answer. A bidirectional LSTM layer is used to capture relationships between inputs both forwards and backwards.

5. **Output layer** then creates two probability distributions using `softmax` on the output of a dense and a LSTM layer for the start and end indices respectively. The index at which the probabilities are highest give the *slice* of context which constitutes the answer.

It is important to note that the context and query sizes are fixed. Therefore, the network is not able to answer queries in large contexts such as complete Wikipedia articles. Since increasing the context length increases the size of each layer, the network training time magnifies accordingly. We will focus on how to overcome these issues with a guided model that has a smaller context size but can scan complete articles.

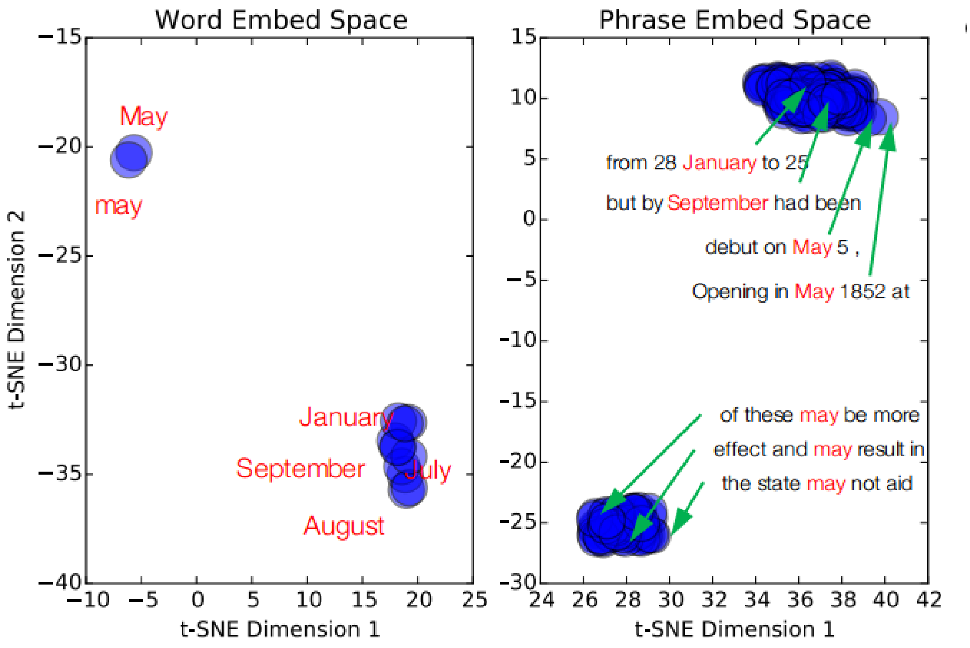


Figure 3.9: BiDAF model contextual embedding visualisation.

The contextual embedding layer provides the context sensitive embeddings for words which represent different meanings depending on their surroundings. In figure 3.9, we can see the distinction made in the contextual level of words `May` and `may` which are grouped in the word embedding space. We take note of the contextual embed layer at this point as our model lacks one due to resource and time constraints and subsequently performs worse.

Chapter 4

Logic Framework

In this chapter we will focus on *how Kevin figures out what to do* when presented with user input. We present a novel logic framework that builds on natural language constructs rather than other primitives such as predicates. We use concepts such as sentence similarity to create a logical framework that allows Kevin to process user input and compute a response. The logic framework resembles logic programming (section 3.3) but does not necessarily follow the semantics of one. We will construct `Rules` (section 7.2.3) in certain ways to perform different actions based on user input.

Expression	Description
Hi there.	`ExprPTree` without any variables
My name is X:Alice.	`ExprPTree` with Alice sub tree variable
:weather:X	`ExprCall` for `weather` function with argument variable X
:weather:X(London)	`ExprCall` for `weather` function with default argument

Table 4.1: Expression notation summary for logic framework.

We summarise the basic expressions that will be used throughout this chapter in table 4.1. Unless otherwise stated, we will be using the constructions described in the implementation section 7.2. All rules are composed of natural language parse trees with wrapping expressions to create a natural language framework that runs Kevin.

4.1 Scripting

The first and most basic construct we have captures *if this then that* style interactions. We manually script, write rules that determine the question answer model. Basic scripting rules can be used in many different scenarios and are predetermined by the programmer. These query-to-answer rules cover many possible interactions and serve as the building blocks for more complicated queries. Since the interaction is fixed, they are reliable in creating a desired conversation with the user. An important emerging property of Kevin is that all answers computed in such a manner require them to be a *supported model* [68] of its knowledge base. If all answers were purely drawn from a closed knowledge base, no external interference, then every answer that Kevin gives would be a supported model.

$$\text{Hi.} \lor \text{Hello.} \lor \text{Hi.} \leftarrow \text{Hi.} \lor \text{Hello.} \tag{4.1}$$

$$\text{Heads!} \lor \text{Tails!} \leftarrow \text{Kevin, heads or tails?} \tag{4.2}$$

A very basic example used by Kevin is how to answer a greeting captured by rule 4.1. The query part contains a greeting from the user capture by the disjunction of similar greetings Kevin will recognise. The head of the rule then contains the answer to that greeting which could be one of three parse trees. The display semantics (summarised in table 7.4) of an `ExprOr` picks a random greeting as a reply. A more interesting example that uses the display semantics of such an expression is captured by rule 4.2. Kevin randomly picks heads or tails whenever the user asks the corresponding query.

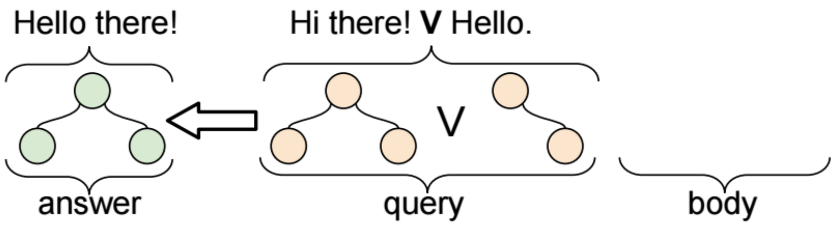


Figure 4.1: Graphical representation of a basic rule in logic framework.

Since the body of these rules are empty, they become *vacuously* true as represented graphically in figure 4.1. Once they are triggered by a matching query, the head is proved trivially thus displaying it as the answer. We also note that the expressions in the logic framework are built upon parse trees as the base construct in this situation. It is crucial to understand that rules are selected similarity measures during the query evaluation stage (section 7.3.4). Therefore, the user can vary the query and still trigger the correct rule up to a certain similarity threshold.

4.1.1 Tracing an Answer

To clarify how the logic framework comes about answering a query, we will consider rule 4.2 and trace the functions performed by the framework in order to give back either heads or tails as the answer. The generic overview of this functionality is given in the query pipeline section 7.3; however, we will focus on a specific instance. We start tracing once the query hits the evaluation stage:

1. A query *similar* to `heads or tails` as a parse tree comes into the evaluation stage

and Kevin searches for the *most* similar expression in the *query* part every rule. The search covers all rules which have a query part that are within the applicable context across all knowledge bases.

2. Evaluation step discovers the rule 4.2 which contains the most similar parse tree for the given query. Now the evaluation step passes the *head* of the rule to the proof engine and tell it to start the proof from that particular rule.

3. The proof engine takes the rule and looks at the body to check if there are any premises that needs to be satisfied in order to the prove the head. In this case, there aren't any; the proof engine vacuously succeeds and passes the head of the rule to the display module to be returned as the answer.

4. The display module receives the disjunction (Heads! $\vee$ Tails!) as the answer expression. By invoking the display semantics of the disjunction expression, we get back a random expression containing a parse tree. Invoking the display semantics of an expression wrapping a parse tree gives back the string value of the wrapped parse tree. Finally, that string is passed to the output module to be shown to the user.

The crucial part of the logic framework and how rules work lies in the distinction of queries and the body of a rule with the answers in the head. This construction is a common theme to compute answers using the framework. A similar sequence of computations happen for more complicated rules as the initial query triggering rule has the response in the head.

Query	$\alpha_{\text{fuzzy}}(Q, r_q)$
Heads or tails?	1.0
heads or tails	0.950
what about heads or tails?	0.926
tell me heads or tails!	0.898
tails?	0.864
heads?	0.855
give me lucky tails	`fail`
heads	`fail`
tails	`fail`

Table 4.2: Example user query triggers for rule query "Heads or tails?"

Table 4.2 demonstrate the flexibility of the similarity based evaluation mechanism within Kevin for triggering the heads or tails rule 4.2. In this example, almost any medium length sentence containing heads or tails will trigger the rule as there aren't any other rules within the test knowledge base containing information about heads or tails. Therefore, if the user query *hints* at heads or tails the similarity measure picks up on it up to a certain threshold so that queries which might mean something different do not easily trigger it. If we lower the threshold then some of the failing queries can succeed but might trigger rules other than the intended one.

4.2 Facts

When we need to store unconditional information within the logic framework, often facts, we can omit both the query and the body to capture that piece of information. Since facts do not have a query part, they cannot be returned as answers to user queries but can be used in other proofs. In most cases, facts store permanent information such as user location and external corpus input. Therefore, anything *given* without an argument to Kevin can be considered a fact which is precisely what this construct tries to capture.

$$\begin{aligned} \text{Martin lives in Berlin.} &\leftarrow \\ \text{Par is the capital city of France.} &\leftarrow \\ \text{You left your keys in the car.} &\leftarrow \\ \text{My name is Brian.} &\leftarrow \end{aligned} \tag{4.3}$$

Examples of facts can be seen in rules 4.3 capturing various use cases. The first fact is often a user directive giving a piece of information that the logic framework would take unconditionally as true. The second case covers information that is specific to a user, in general instances user preferences. The location of the user's keys are accepted to be in the car regardless of the state the knowledge base is in. Finally, facts can act as customisations such as changing Kevin's name to Brian in the third case.

All external parsed text such as Wikipedia articles are consumed as facts into the logic framework. Every sentence in the text becomes the *head* of an unconditional rule that is added to some knowledge base, possibly temporary. The consumed information then can be used to help with other proofs by using the proof semantics of parse trees (section 7.2.2). The information does not need to be consistent logically and it is not enforced to be so. This approach creates a lot of flexibility to intake vast amounts of information from noisy inputs; however, it can cause false proofs to succeed if not isolated. Therefore, especially when Kevin consumes external text outside of user input, the information is placed in their respective *context* such that those facts only trigger when the appropriate context is set. For example, it could be the Wikipedia application consuming summary pages for a search query, then all the facts generated will be under some Wikipedia context. This design allows to almost quarantine noisy and possibly inconsistent information from generating wrong proofs in the face of other valid user input.

A concern about facts is that they are very powerful since they cannot be refuted.

They only provide a piece of unconditional information to the active context they belong to. However, they are considered equal to every other rule and are exposed to external functions to update and remove them. For example, if the user updates the location of their keys, the process which generated the fact that describes the location might consider updating the existing fact to create its own view of consistency. Since the framework doesn't force a consistent state, the meaning of being consistent can vary on the situation and how components, particularly external functions, would like to handle it. This design creates a lot of space for creating unexpected situations such as forcing Kevin to think Apple Inc. is a type of fruit rather than a company and subsequent proofs such as searching information on Wikipedia will yield information about the apple tree instead of the company.

4.3 Constraints

Facts are very powerful in changing the outcome of the proof mechanism due to empty queries and bodies. Therefore, inconsistent facts can force Kevin to output wrong results defeating the purpose of having a logic framework proving answers as supported models its knowledge base. To counter that situation we can leverage a different rule construction that allows us to express constraints. These constraints are at the logical framework level and do not have a specific meaning unless actively invoked. In other words, they are not enforced by the framework and it is the responsibility of a component that is concerned about inconsistent states such as an external function worried about the user being at two places at the same time. Similar to facts they do not have a query part meaning that they cannot be triggered by a user input to give an empty answer.

$$\begin{gathered}\text{Your keys are at the kitchen.} \leftarrow \\ \leftarrow \text{Your keys are at the X:office. , Your keys are at Y:home. , } X \neq Y\end{gathered} \tag{4.4}$$

Equations 4.4 describe a setup for constrains in which the user's keys cannot be at two places from Kevin's perspective. The example is currently consistent since there is only one fact detailing the location of the keys. However, when the user provides an updated location the constraint will be violated. For example, the user can say "I left my keys at my friends house." adding another fact clashing with the existing one. As far as the framework is concerned this input is perfectly fine and unknowingly will create an inconsistent state unless the component that processed the fact checks for any violated constraints. In other words, Kevin is entirely oblivious to the semantics imposed on the contents of its knowledge base; the components and external functions which interact with the knowledge base can interpret them in a certain way.

Since the meaning of inconsistency is variable within the framework there is no default way of resolving conflicts when a constraint is violated. It is perfectly fine to have an inconsistent knowledge base for Kevin to run on. However, as a basic method of resolution, the pipeline provides a mechanism that randomly removes a rule which caused the constraint to be violated. In the case of equation 4.4, if the user provides a new location, the fact describing the last the known location will be removed from the knowledge base as it is the only existing rule that helped violate the constraint.

There is no default mechanism to check for constraints that is enforced to all fact based inputs. Therefore, addition of constraints does not guarantee a consistent knowledge base by any means. They can be regarded as any other rule which contain some information that pieces together parse trees within the knowledge base. How constraints are used is not necessarily the concern of the logic framework, it just provides them for the interacting components. This design deviates from traditional logic semantics on purpose to accommodate more flexibility within the digital assistant environment.

4.4 Intent parsing

An important feature of any digital assistant is to convert natural language queries into structural intents that external functions can process in order to produce a response. Our logical framework accommodates such a requirement by a small change to basic rule structure. Instead of returning a fixed sentence for a query, we can place an external function in the head of rule to be invoked as the response. By using variables within the query, we obtain a intent parsing mechanism built on existing tools in the framework without any additional changes.

$$\text{:weather:X} \leftarrow \text{What is the weather in X:London?} \tag{4.5}$$

A very common intent, asking the weather at a location, is captured by rule 4.5. In this case, any user query that is *similar* to asking the weather at a location will trigger this intent. Fuzzy unification (section 5.2.2) will extract the correct location argument for the intent, converting in a sense the natural language input into structured information for the external function `weather` to consume. Since there are no premises to prove, the head becomes vacuously true and the display semantics of the external function is triggered as the response. Thus, the user receives the what the external function returns, in this case the weather at a given location.

Table 4.3 captures what kind of user queries trigger the intent described by equation 4.5. Thanks to the similarity based fuzzy unification, Kevin is able to capture most of the queries that *hint* at weather at a give a location. As a result, the user utterance does

Query	$\alpha_{\text{fuzzy}}(Q, r_q)$
What is the weather in Berlin?	1.0
what is the weather in Paris	0.983
how is the weather in Paris	0.975
how is the weather like in Rome	0.972
is there a storm in Dublin	0.928
is it warm in Birmingham	0.904
I wonder if it is cold in New York	0.897
weather in Liverpool	0.847
hot or cold in Madrid	`fail`
Is Paris cold these days?	`fail`
Paris weather?	`fail`

Table 4.3: User queries that trigger the weather intent.

not even have to contain exact matching tokens in the query part of the rule, just similar words suffice creating an extremely flexible intent parsing mechanism. The failed cases are due to the similarity threshold which tells Kevin to stop trying if it below $k = 0.82$ in this case, obtained from evaluating the fuzzy similarity metric (section 8.1). We can lower the threshold to succeed the failed unifications; however, lowering the threshold creates an increasingly unstable query matching by making the evaluation more and more aggressive. As a result, even if the user asks something Kevin doesn't know, evaluation can find something *similar* enough which might completely be the wrong intent.

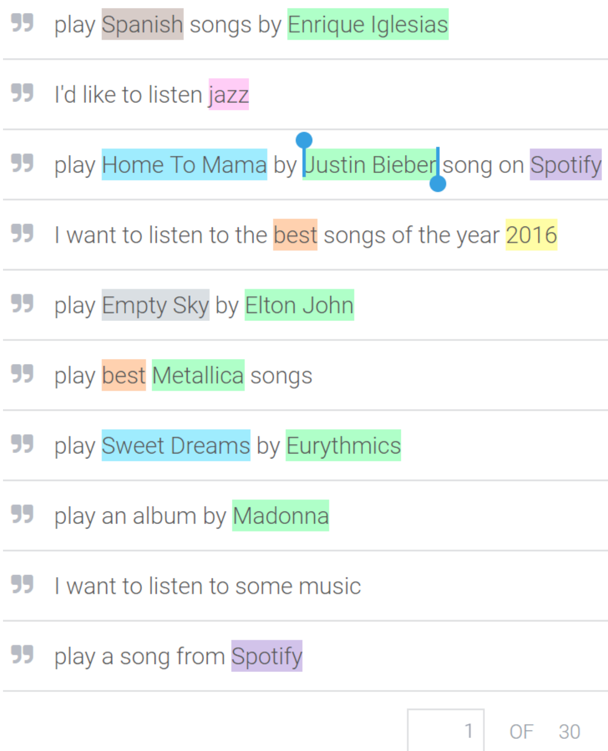


Figure 4.2: Query rules for music playing intent in API AI.

As with any intent parsing mechanism the intents have to be manually constructed by the programmers. This situation is present in industry leading companies such as API.AI [63] and shown in figure 4.2. The intent for playing music has 300 triggers with different variable sub components that it tries to match similar to the constructs available in Kevin. The number of rules is usually directly related to the similarity threshold, the higher threshold the more rules required. When the threshold is set to 1.0 then Kevin will be looking for exact matches and require a rule for every possible user input for particular intent. On the other extreme, if the threshold is 0 then any rule is considered similar enough.

Since the intent parsing mechanism relies on fuzzy unification, any parsing errors that affect the unification process are also reflected in intent parsing. Intent parsing is most severely affected by natural language processing library parsing discrepancies described in section 7.1.1. Therefore, most of the capacity of this mechanism is limited to the downstream operations that process the natural language input.

Writing a few intent rules and invoking external functions can be regarded as a natural language interface for any software. For example, we could write rules such as "Open the internet browser." and "Switch to the next application." to provide an interface for a generic desktop environment. Kevin would recognise the intents and trigger corresponding functions working as natural language service that sits between user utterances and desktop environment functionality.

4.4.1 Interactive Rule Constructing

Since we can attach any external functions to intents, we can also provide means of constructing new rules using intent parsing. In other words, Kevin can interactively build new rules using few intents that capture those actions. This capability adds a natural language interface to a knowledge base. The interaction needs to be carefully crafted as with any other intents using several nested contexts to understand which part of the rule the user is trying to modify.

$$\begin{aligned} \text{:newrule} &\leftarrow \text{That is the wrong answer.} \\ \text{:addpremise} &\leftarrow \text{Add a new premise.} \\ \text{:setanswer} &\leftarrow \text{Here is how the answer looks like.} \\ \ldots &\leftarrow \ldots \end{aligned} \tag{4.6}$$

Rules in equation 4.6 provide a non-exhaustive list of intents that integrate into Kevin's knowledge base and allow users to create new rules if an answer given is wrong. This interaction allows a user to add new information on the fly still using natural language that could be a personal preference such as their favourite colour or a general method that could be shared with other users such as getting the traffic information. Although potentially useful for basic user input based rules, any action that requires external functions such as API calls would still require a programmer's touch if they are not already implemented.

4.5 Template IR

We can continue building on top of fuzzy unification in order to unify sentences to extract out arguments as we have done in intent parsing. Template based information retrieval uses fuzzy unification to match similar sentences to extract a piece of information out. This approach allows a fast method for constructing rules that provide a uniform representation of noisy inputs. Template matching does not use another neural network to extract an answer, just matches sub tree templates using only what is available through fuzzy unification (section 5.2.2).

$$\begin{aligned}\text{The capital of X:Slovakia is Y:Bratislava.} \leftarrow &\text{What is the capital of X:Slovakia?}\ , \\ &\text{The capital city is Y:Bratislava.}\end{aligned} \tag{4.7}$$

We start by constructing a rule that contains the query for the user to trigger and an answer format; additionally we add a premise that will act as a *template* as shown in equation 4.7. We the can override the default semantics of a parse tree for that particular premise to perform a fuzzy search over facts related to the given country; for example it can search for facts in the context of Wikipedia. As a result we obtain a template matcher which extracts the capital city of a given country. Then the result is returned in a sanitised hand-crafted format. When the fuzzy unification succeeds on the most similar sentence for the consumed facts about the country, it unifies the capital city completing the proof for the head which now has all its variables set.

It is important to note that we override the default semantics of the wrapping expression that acts as the template. Instead of searching through the knowledge base, in this example, we complete the proof by searching for an answer on Wikipedia. As a result, we altered how that premise is satisfied; when the template is matched on a Wikipedia page it is considered true thereafter. The proof engine is again oblivious as to *how* an expression is proved and we exploit the flexibility of the framework to consume external information without having to modify the framework.

For the rule constructed in equation 4.7, we look at examples extracted from Wikipedia summary pages of the corresponding countries in table 4.4. So long as the sentences are similar and contain the same sub tree structure as the variable for the city name, fuzzy unification can extract the correct piece of information since it is based on dependency structure matching. For cases it fails, either the similarity threshold $k = 0.7$ has to be

S	$\alpha_{\text{fuzzy}}(S, r_p)$	$\theta_{\text{fuzzy}}(S, r_p)$
Poland's capital and largest city is Warsaw.	0.816	`true`
Germany's capital and largest metropolis is Berlin, while its largest conurbation is the Ruhr.	0.757	`true`
New Zealand's capital city is Wellington, while its most populous city is Auckland.	0.732	`true`
Australia's capital is Canberra, and its largest urban area is Sydney.	0.691	`false`
France is a unitary semi-presidential republic with the capital in Paris.	0.533	`false`

Table 4.4: Matching noisy facts to template The capital city is X at $k = 0.7$.

lowered or another template has to be created capture that particular way of expressing the capital city of a country. Since the template rule is considered equal to any other rule, the proof engine can now use this rule to find out if a city is a capital of a country; in other words, this piece of information captured by the template can be chained in proofs.

Template matching is very limited in consuming noisy input; however, it serves as a good starting point for acquiring and processing external information and accommodating them within the logic framework. It is worth noting that template matching is quite efficient at processing large corpus of text since it relies on vector cosine similarities used by fuzzy unification process. Similar to intent parsing, though, it suffers severely from any downstream natural language processing errors such as dependency structure changes or other parsing errors.

4.6 Template deduction

Often in logic, rules are chained to derive conclusions based on existing information. Template based deduction allows Kevin to derive those conclusions using the different rule constructs described so far: facts, constraints, intents and templates. When proving an expression, the proof engine does not know how each individual expression acts as they all have different proof semantics (section 7.2.2). This design allows an arbitrary rule to draw information from various sources without requiring changes to the underlying proof system. As far as the framework is concerned, the expressions know how to prove themselves allowing for various functionality to be incorporated by overriding the existing proof semantics of expressions.

$$\begin{aligned} \text{Yes.} \leftarrow &\text{Does X:Mary live in the capital of Y:Germany? ,} \\ &\text{X:Mary lives in Z:Hamburg. ,} \\ &\text{The capital of Y:Germany is Z:Hamburg.} \end{aligned} \tag{4.8}$$

Equation 4.8 describes a single rule that determines whether someone lives in the capital city of a given country. In this case we build on the template information retrieval rule from equation 4.7 that provided a way of proving whether a city was the capital city. It is important to remember that the proof engine strictly runs from left to right following these steps:

1. The user asks whether someone lives in the capital city of a country. As an example let us consider "Does Alice live in the capital of England?" for the user query.

2. As usual, the evaluation module matches the query part of the rule described by equation 4.8 as the best match and fuzzily unifies the user query with the query part of the rule. Thus, the current state of the variables becomes ($X \to$ Alice, $Y \to$ England) yielding the head expression to the proof engine.

3. Since the rule contains two premises, the proof engine starts from the left most expression that tries to figure out where Alice lives. If there is a fact of the given expression form such as Alice lives in London. ←, the proof engine will unify the premise with the head of that fact.

4. The proof for where Alice lives in this case will succeed since it was a *fact*, thus unifying variable $Z \to$ London. If the proof engine wasn't able to find out where Alice lived, the entire proof would have failed since it is the first premise and there aren't any other expressions to backtrack; the query is not considered part of proof.

5. The final premise has all its variables grounded and the proof engine searches for a head of a rule that matches the parse tree. It is important to clarify that since these premises are searching internally, they are unified using strict unification (section 5.2.1). The rule we created for template information retrieval successfully unifies and the proof engine tries to prove whether London is the capital city of England or not.

6. At this point, Kevin has no idea it is accessing external corpus such as Wikipedia to show that London is indeed the capital of England. Therefore, the proof engine succeeds following in proving the capital city of England is London as the final premise.

7. All the premises are satisfied and since this rule was top level query trigger rule, the head of the rule with the unified variables is passed on the display module and the proof is complete. Kevin can show as a consequence of its current knowledge base, Alice does indeed live in the capital city of England.

All rules within the knowledge base can be chained using the oblivious proof engine allowing Kevin to derive consequences of its knowledge base regardless of where the information is retrieved. Hence, it can build new relationships and form conclusions using existing ones.

$$\begin{aligned} \text{Yes.} \leftarrow & \text{Does X:Mary live in the capital of Y:Germany? ,} \\ & \text{The capital city of Y:Germany is Z:Hamburg. ,} \\ & \text{X:Mary lives in Z:Hamburg.} \end{aligned} \tag{4.9}$$

The efficiency of this process depends on the ordering of the premises. Since they are proved strictly from left to right, both the semantics and the proof runtime can change if they are re-ordered. For example, a re-ordered version of rule 4.8 is presented in rule 4.9. In this case, the rule is looking for *a capital city of the given country* and then checking whether *Alice lives in that capital city*. In the original version, Kevin was looking for where Alice lived first then checking if the city was a capital. If the query was just asking if Alice lives in a capital not just for a given country, then this construction would force the proof engine to iterate through countries and their capitals and check if Alice lives there. Therefore, the ordering affects the performance of the proof process but also the subtle semantic differences caused by at which stage the variables are unified. Often, the quicker the variables are unified the better the performance as it translates to less backtracking in a smaller search space of grounded values.

4.7 Negation by failure

An interesting construct for Kevin to handle in natural language is negation by failure (section 3.3.1). Negation by failure allows Kevin to reason in the absence of information since the semantics of such a construct rely on *all attempts to prove it failing*. It can be rephrased as if something is not known to be the case then this expression is true because the proof engine couldn't prove otherwise. By adding negation by failure, we often consider defaults that describe the general case such as birds can fly; but, adding specific facts such as penguins don't fly, allow the proof to overrule the negation providing an intricate reasoning process.

$$\text{My name is Kevin.} \leftarrow \text{nbf My name is X:Bob.} \tag{4.10}$$

One of the most interesting uses of negation by failure is how Kevin determines its name, described in rule 4.10. The rule states that in the absence of any information regarding Kevin's name, its name can be Kevin. Thus, if the user hasn't provided any information on what the name should be, Kevin will conclude that its name for now is Kevin. If there was any information, then the proof engine would succeed in finding *a name* thus causing the rule to fail. In this case, the variable inside the negation is not grounded at the time of execution to find any name that might Kevin have, including Kevin if it is explicitly stated. The reason this construct doesn't go into an infinite loop by trying re-apply the same rule since the premise is unifiable with the head is because the proof engine remembers the rule chain. As described in section 7.3.5, a rule cannot be reused in proving its own premise thus stopping Kevin from entering an infinite loop. In other words, the premise of a rule has to find information from other rules that did not contribute to the proof so far.

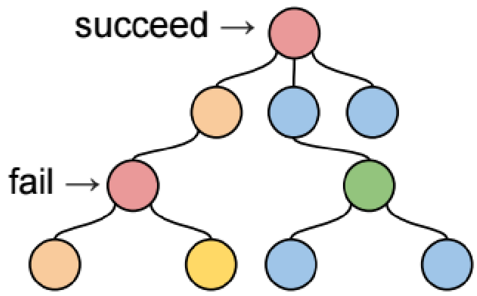


Figure 4.3: Graphical representation of negation by failure.

Figure 4.3 portrays a graphical representation of a negation by failure proof. The red nodes denote rules that the negation rule cannot access, including itself. Blue nodes denote the expressions that haven't been touched and green nodes represent the rules the proof from negation can access. This structure captures the design choice we made in order not to flounder when faced with ungrounded variables within negation by failure expressions. As a result, negation by failure is defined purely by its operational semantics in the proof engine.

$$\begin{aligned} \text{Most likely.} \leftarrow & \text{Does X:John live in the capital of Y:Sweden? ,} \\ & \text{X:John is from Y:Sweden. ,} \\ & \text{nbf X:John lives in Z:London.} \end{aligned} \tag{4.11}$$

Another example that describes if we don't know where someone lives but do know where they are from, we can deduce that most likely they live in the capital city of the country they are from, equation 4.11. When this rule is triggered, Kevin tries to prove that John is from the given country and if *it is not known to be the case* that he lives somewhere, Kevin deduces that it is likely that he lives in the capital. There are many more cases that follow a similar pattern particularly with user preferences in which initially Kevin wouldn't know what the user likes and can make general *assumptions* based on an average user. As more and more information is built up in the knowledge base, these default cases would stop to provide answers and at some point can be removed all together from a user's knowledge base.

4.8 Free Form Integrations

In most rules, we imposed a certain structure to the sentence to be searched. However, we can leverage free from question answering using neural networks to create a more flexible information processing pipeline within the logic framework. For example, similar sentence extraction and deep neural network methods such as guided BiDAF (section 6.2) can be incorporated into the framework. Kevin can then search through noisy external corpus to *prove* or find an answer to a query not only the user is interested in but also another internal rule. The proof engine is as usual oblivious to the underlying method of proof; in this case it will be delegated to neural networks and other methods of question answering. There are three main methods for integrating external components into Kevin:

1. **External functions** provide a very simple way running external procedures with side affects. However, their semantics within logic framework are very limited. For example, they are not allowed to backtrack. They are mostly used for basic actions that do not require too many interactions with the user and little information to run. They are wrapped inside `ExprCall` within the logic framework (section 7.2.2).

2. **Query redirecting** allows larger applications to consume Kevin's raw user input using the command module (section 7.3). These applications often require more complicated user interactions such as follow-up questions in a takeaway ordering application. As a result, the logic framework acts as an application context manager, starting and closing them while the application itself is responsible of handling user input.

3. **Expression overriding** allows new `Expr` objects to be created within the framework and their proof mechanisms to be overridden. For example, an expression can *prove* itself by checking an external source of text for a particular information without polluting or interacting with the existing knowledge. The proof function can be any procedure that returns a boolean and optionally unifies variables. Since the proof engine is oblivious to this process, it allows the new expressions to backtrack and be part of the full proof procedure.

Using different integration methods of the framework, we can thus incorporate a multitude of functionality within Kevin to expand its capabilities. This design is important to accommodate more and more functionality, intents, preferences and custom components without modifying the core framework.

4.8.1 Query Redirecting

The simplest way of integrating different question answering schemes is by binding to the command module (section 7.3) and redirecting all of Kevin's input into another component such as an external function or neural network. The component that receives the user input is then responsible of generating a response that piped to the display module directly and therefore can bypass the entire proof process.

$$\begin{aligned}
\text{Opening Wikipedia app.} \land \text{:setcontext:X(wiki)} &\leftarrow \text{Start the Wikipedia app.} \\
\text{:searchwiki:Q} &\leftarrow \text{Q:} \\
\text{Closing current application.} \land \text{:setcontext} &\leftarrow \text{Close the current app.}
\end{aligned} \tag{4.12}$$

Rules under equation 4.12 demonstrate a Wikipedia application within Kevin. The context is set when the user requests to open the application and subsequent rules become available to the evaluation module. The second rule is a *match all* clause that returns a fixed similarity measure $\alpha(S, \_) = 0.94$ for any given query; thus, it allows the input to be redirected to the external function `searchwiki` as variable Q. The similarity measure for a match all clause is not set to 1.0 so that queries that have a higher measure can trigger other rules and change the context. For example, if the user asks to close the current application, it will have a higher similarity measure with the third rule than 0.94 to trigger the correct rule rather than redirecting that query to the external function.

$$\begin{aligned}
\text{Opening X:Bus app.} \land \text{:setcontext:X} &\leftarrow \text{Start the X:Bus app.} \\
\text{:currentapp} &\leftarrow \text{Which application is running?} \\
\text{Closing current application.} \land \text{:setcontext} &\leftarrow \text{Close the current app.} \\
\ldots &\leftarrow \ldots \\
\text{:appredirect:Q} &\leftarrow \text{Q:}
\end{aligned} \tag{4.13}$$

A more general version of context based applications is captured by rules in equation 4.13. The interaction starts with the user requesting an application to be opened and Kevin sets the appropriate context. The logic framework can still provide basic triggers such as which application is running, what the current state is as well as closing it. However, other queries which Kevin does not recognise then get routed to the active context application. This setup creates an API for full applications that could be integrated into Kevin.

In this approach, the logic framework only wraps external query answering components but does not contribute to them. This design is a good starting point for accommodating applications within Kevin that require a certain context to be set and then the input to be redirected. As such, Kevin's functionality is not restricted by the capabilities of the logic framework alone and can be expanded with various integrated applications that work alongside the framework to process user input.

4.8.2 Neural Network Extraction

Besides redirecting queries to different external components, we can leverage the logic framework and create new expressions with different proof semantics. Since the proof engine is oblivious on how expressions are proven, we can integrate various sub components such as neural networks to *prove* an expression and bind it to variables as the framework would expect. Neural networks could be used as a stand-alone application with query redirecting, but in this section we will discuss how to integrate one into the logic framework. We will use the guided BiDAF model described in section 6.2 as our underlying neural network query answering mechanism by fusing them into new expressions.

$$\begin{aligned}\text{The capital of X:Slovakia is A:Bratislava.} \leftarrow &\text{What is the capital of X:Slovakia?} \ , \\ &\text{What is the capital of X:Slovakia?}\end{aligned} \tag{4.14}$$

Building on the previous capital city template matching rule in equation 4.7, we construct a very similar rule as seen in equation 4.14 that provides the same outcome. However, in this case it seems like we duplicated the query and indeed we have; but, the difference is captured by the semantics of those expressions. The first question is the user query that triggers this rule and as such it is separate from the body which needs to be proved in order to conclude the head. Kevin follows these steps in this case:

1. The user query asking what is the capital city of a country triggers the rule as usual. The rule looks simple and behaves the same from the evaluation module's perspective.

2. The proof engine sees there is a premise and invokes its proof procedure. Again the proof engine is oblivious to *how* the premise will be proved.

3. The premise expression containing the identical query has its proof function overridden to redirect to a neural network to search for the given country X on Wikipedia. Then, Kevin literally asks the neural network to give an answer from the Wikipedia

summary and bind the answer to variable A. If an answer is found, the *proof* succeeds and the proof engine passes the head of the rule to the display module.

4. The head of the rule is grounded with the country given by the user X and the answer obtained by asking the query to the neural network A. Together these variables complete the response and is returned as the answer to the user.

By incorporating different components as expressions within the logic framework, we can exploit the knowledge base in order to build more and more complicated rules. For example, if Kevin wants to figure out if a famous person lives in the capital city of the country, the query can be deconstructed to find where the person lives and whether it is the capital city. If there are already rules for determining where famous people live such as an external function and using another rule to check if a given city is the capital, Kevin can conclude if that person lives in the capital or not. This knowledge expansion with the flexibility of external components creates vast potential for flexibility.

$$\begin{aligned} \text{AA:Bob is a teammate of X:Alice.} \leftarrow & \text{Who is a teammate of X:Alice? ,} \\ & \text{What team X:Alice play for? ,} \\ & \text{Who plays for A:Arsenal? ,} \\ & AA \neq X \end{aligned} \tag{4.15}$$

Rules in equation 4.15 captures the power of creating higher abstractions using underlying questions answering components such as neural networks. In this example, we describe what a *teammate* is in natural language using the logic framework. As before, we bind the two premises to neural networks that extract the information we are looking for: the team Alice plays for and someone who plays for the same team. Finally, we make sure they are not the same person to deduce that Bob is the teammate of Alice. The first neural network inquiry binds the answer to variable A and the second one to AA with the proof proceeding as normal. *Instead of training a new neural network on thousands of examples to understand what a teammate is, we could provide that description in one rule in natural language within our framework* while still using the existing trained neural network.

Figure 4.4 shows a graphical representation of the expression proof overriding mechanism. As seen in the graph, the proof engine is not aware that the proof is *delegated* to other components such as neural networks. As a result, the existing the framework harnesses information from external resources which it wouldn't have been able to do before.

A concern regarding expressions wrapping external proof mechanisms is stability. As more and more unrelated external components, services become integrated into rules, the

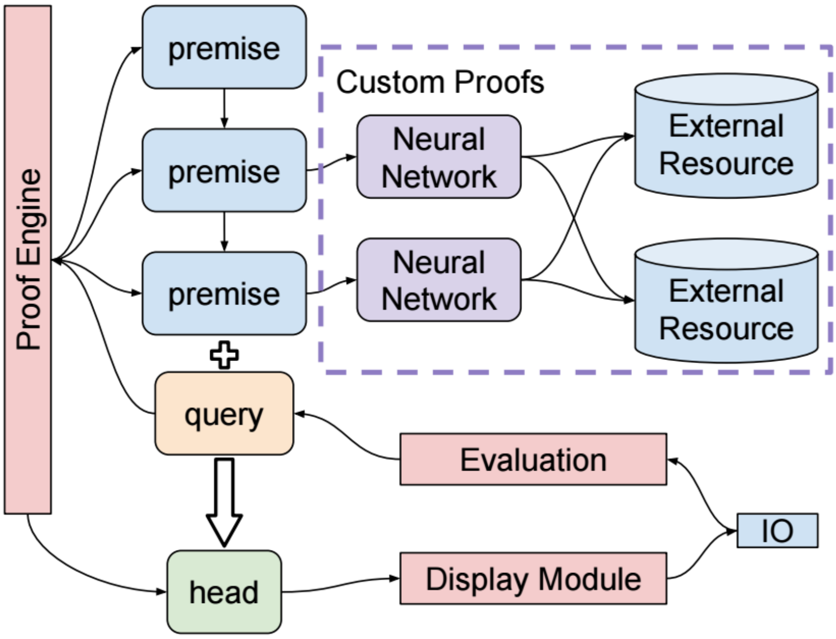


Figure 4.4: Graphical representation of overriding expression proofs with neural networks.

logic framework becomes more unstable in the sense that the information flow is not regulated. Any expression can interact with any source returning some value and it is assumed to behave correctly. Thus, the framework harbours more points of failure not only in the proof engine but also from an operational point of view. The external functions can indeed crash Kevin creating unwanted behaviour and responses. The whole idea of answers being supported models slowly fade away as more proofs are delegated to external components.

Chapter 5

Similarity & Unification

In this chapter we focus on how to select correct rules and expressions using different sentence similarity metrics as well as the unification procedures that chain expression proofs together. Kevin is heavily built on these similarity concepts that underpin its capability to handle natural language both in its logic framework and when consuming external textual sources.

5.1 Types of Similarity

A recurring topic through this report is the *similarity* of two given sentences. We do not consider multiple sentences, document similarity and restrict our definition to a binary function. All similarity measures are implemented as part of `PTree`s (section 7.2.1) which represent the corresponding sentences. Sentence similarity is used throughout Kevin's infrastructure to handle natural language input. For example, it is used to select rules that match a user query (section 7.3.4). We first define what sentence similarity is and consider different forms of it that are present in Kevin.

$$0.0 \leq \alpha(s_1, s_2) \leq 1.0 \tag{5.1}$$

The similarity between two arbitrary sentences s_1 and s_2 is a function α that takes those sentences and outputs a value strictly ranging from 0.0 to 1.0, equation 5.1. A value of 0.0 indicates that the sentences are completely different and a value of 1.0 signifies identical sentences with respect to some property. The similarity function α can represent different properties such as semantic or syntactic similarity. It can be a combination of other similarity functions so long it is *well-founded.*

Definition 5.1.1 *A similarity measure is considered well founded if the following three properties hold:*

1. *returns a value ranging from 0 to 1,* $0 \leq \alpha() \leq 1$.
2. *identical sentences always have a value of 1,* $\alpha(s, s) = 1$.
3. *it is commutative,* $\alpha(s, t) = \alpha(t, s)$.

A well-founded similarity measure is ideal but not necessary as we will discover most useful ones are not well-founded. The important feature that this property provides is stability when measuring sentence similarities. We can have some sense of reassurance that the similarity metric behaves in an intuitive manner such as recognising duplicate sentences by returning 1.

$$\alpha_{\cos}(w, k) = \cos(\theta) = \frac{\mathbf{u}_w \cdot \mathbf{v}_k}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} \tag{5.2}$$

In most cases, we build the similarity metrics out of vector embeddings presented in the background section 3.4.2. The GloVe [3] vector embeddings provide a similarity measure using the cosine angle between the two word vectors. Equation 5.2 represents the formulation of the *cosine similarity*[1] between two words w and k and their corresponding vectors u and v. It is a well-formed similarity measure since $\cos(\theta)$ follows the conditions of definition 5.1.1.

w	k	$\alpha_{\cos}(w, k)$
apple	apple	1.0
apple	orange	0.56
apple	chair	0.17
London	Paris	0.61

Table 5.1: Examples of cosine similarity using GloVe [3] vectors.

Table 5.1 contains some examples of cosine similarity. We observe that `apple` is more *similar* to `orange` then to `chair` as expected. The similarity measure in this setup depends on the training method used; as a result words that appear in similar contexts are more similar. These vector similarities provide us with a metric at the word level which we can use to build sentence level similarity metrics.

5.1.1 Document Average Similarity

The simplest way to combine word vector similarities is to average them to get a single vector for the corresponding sentence. The document average similarity averages all vectors of its tokens to get a single vector representing the sentence itself. This is the default

[1]Cosine similarity might refer to different measures depending on the context. In our case it is always defined on vectors of same dimensionality.

similarity behaviour in the natural language processing library Spacy (section 7.1.1) used by Kevin.

$$\mathbf{v}_s = \frac{1}{|S|}\sum_{i=1}^{|S|} S_i \tag{5.3}$$

$$\alpha_{\text{doc}}(S,T) = \alpha_{\cos}(\mathbf{v}_s, \mathbf{v}_t) \tag{5.4}$$

The vector representing the sentence $\mathbf{v}_s$ in equation 5.3 is the same dimension as its word vectors; S_i represents the vector of the ith token in sentence S. This method is very efficient as the sentence vectors can be computed when the sentences are parsed. Since the result is another single vector, any similarity method applied on single words can be used on sentences such as cosine similarity, equation 5.4. In fact, it can handle large documents by simply averaging all token vectors again.

However, as the sentences get longer, more and more information is collapsed into a single vector. As a result, commonly occurring tokens such as `the` pull the average towards their vector embedding leading to a biased similarity measure. An approach to avoid this problem is to ignore frequent tokens as well as certain tags that do not necessarily encode useful information such as determinants (DT)(appendix A.1).

5.1.2 Bag-of-words Similarity

The bag-of-words similarity performs an identical operation to document average similarity with one exception. We normalise the entity vectors that are present in the parse tree; in other words we leverage the named entity recogniser to have a single vector for location, time and people. If there are no named entities, the bag-of-words approach yields exactly the same result as the document average approach.

$$f(x) = \frac{1}{|C_x|+1}\left(g(x) + \sum_{i\in C_x} g(i)\right) \tag{5.5}$$

$$g(x) = \begin{cases} \mathbf{v}_{ent}, & \text{if } x \in E \\ \mathbf{v}_x, & \text{otherwise} \end{cases} \tag{5.6}$$

$$\alpha_{\text{bow}}(S,T) = \alpha_{\cos}(f(S_r),\ f(T_r)) \tag{5.7}$$

The named entity vector substitution is represented as a function $g(x)$ which returns a pre-determined vector $\mathbf{v}_{ent}$ if token x is a named entity, just the token vector otherwise, defined in equation 5.6. This construction is recursively applied to normalise all named entity vectors in the parse tree as in equation 5.5 in which C_x represent all the children of the current node not just 1 level deep. If a node has no children $C_x = \varnothing$ then the corresponding word vector is returned; otherwise, the average of the node's word vector with its children's bag-of-words vectors is calculated as the corresponding vector. Finally, the vector returned on the root node can be used to calculate the cosine similarity between two sentences as before, equation 5.7.

S	T	doc(S, T)	bow(S, T)
What is the weather in London?	What is the weather in Liverpool?	0.973	1.0
How is the weather like in Berlin?	What is the weather in London?	0.958	0.987
How are you?	Are you doing well?	0.954	0.954
My name is Alice.	My name is Bob.	0.945	0.999

Table 5.2: Comparison of document average and normalised bag-of-words similarity.

Table 5.2 captures the reason why we normalise the vectors of named entities. Despite the vectors of named entities being *similar*, normalising gives consistently higher similarity of sentences that contain them. This step is useful if there are a lot of rules with named entities in them and we want to pick the most similar rule matching the query. If we do not normalise, rules which have exactly the same named entities might be more similar leading to mistakes such as wrong intents being recognised. For example, if we want to capture a weather information intent, we do not want the vector of the named location to add bias to the similarity measure.

The vector substitution function can also be used to trim out most commonly used tokens and weight specific tokens such as nouns. However, in most cases it is hard to determine which tokens should contribute or not. For example, we might want to trim the determinant (DT) in sentence "Bob is the builder." but if we have another sentence "Bob is a builder." we won't be able to distinguish them. Although such cases are rare, a greedy trimming function can cause discrepancies in the similarity measure.

5.1.3 Maximal Token Similarity

Instead of compressing the entire sentence into a single vector to calculate its similarity, we can consider each token separately. The maximal token similarity looks for the most similar token in the other sentence and then averages every tokens' findings. In other words, it calculates how much of the given sentence is found in the other sentence. Thus, the similarity represents how much of the current sentence in some sense is present in the other.

$$\alpha_{\text{token}}(S, T) = \frac{1}{|S|} \sum_{i=1}^{|S|} \max_j \alpha_{\cos}(S_i, T_j) \tag{5.8}$$

Equation 5.8 demonstrates that for every token in the original sentence S, we find the most similar token T_j in the other sentence and then take the average over all tokens in the original sentence. When used with context based similarity measure such as cosine similarity, the maximal token similarity reads how much meaning of the original sentence is embedded inside the other sentence. Thus, this similarity construction is impervious to permutations of tokens in either sentence. It does not consider the ordering of tokens or structure and can be considered too *greedy*.

The maximal token similarity is not well-founded as it is not commutative. If a sentence is completely embedded inside another, then the similarity will be 1.0 but not the other way, demonstrated in table 5.3. However, when common tokens such as determinants (DT) and prepositions (PRE) are not considered, maximal token similarity can be used to find similar sentences for a given sentence in documents. This usage is discussed in detail in the evaluation section 8.2.

5.1.4 Weighted Maximal Token (Fuzzy) Similarity

The fuzzy similarity measure tries to address some of the issues of the maximal token similarity by combining some structural information into the sum. Instead of every token having the same contribution to the overall similarity, we follow the dependency structure to average at every node. Thus, the sentence similarity is defined from a node of the given sentence in a recursive fashion. To get the final sentence similarity, the root node of the first sentence can be used as the starting node.

$$\alpha_{\text{fuzzy}}(S_x, T) = \frac{1}{|C_x| + 1} \left(\max_j \alpha_{\cos}(S_x, T_j) + \sum_{c \in C_x} \alpha_{\text{fuzzy}}(c, T) \right) \tag{5.9}$$

As defined in equation 5.9, if the node has no children C_x then fuzzy similarity yields the maximal token similarity for that particular token. Otherwise, the children's maximal token similarity are averaged following the dependency structure. This approach avoids common tokens to significantly influence the final similarity and gives root nodes such as verbs higher precedence.

$$\alpha(\text{how is the weather}, T) = \frac{\alpha(\text{how}, T) + \alpha(\text{is}, T) + \frac{\alpha(\text{the}, T) + \alpha(\text{weather}, T)}{2}}{3} \tag{5.10}$$

An example is shown in equation 5.10 demonstrating the averaging over the dependency structure. By collapsing similarities further down in the dependency tree, we try to accumulate information about the structure. In the example, the token `the` determines `weather` and is averaged together and not weighed equally with other root level tokens.

In table 5.3, the fuzzy similarity still returns 1 if the sentence is embedded inside

S	T	mts(S, T)	fuzzy(S, T)
How is the weather?	How is the weather today?	1.0	1.0
How is the weather today?	How is the weather?	0.893	0.893
Alice picked up the chair.	The chair was picked up by Alice.	0.999	0.999
The chair was picked up by Alice.	Alice picked up the chair.	0.926	0.868

Table 5.3: Comparison of maximal token and fuzzy similarity.

another; however, it is slightly more careful about structure changes. Overall, it could be considered a greedy not well-founded similarity measure but slightly more stable then maximal token similarity. As a result, fuzzy similarity is the main measure used by Kevin for matching user inputs with rules and extracting similar sentences from a given corpus.

5.1.5 Tree Edit Similarity

Since the dependency parsing returns a tree structure, we can define a similarity measure based on the tree edit distance of the two sentences. The tree edit similarity is a metric based on the tree edit distance between sentences which considers carefully the structure. We implemented the tree edit distance algorithm described by Zhang and Shasha [72] using dynamic programming and consider the following insert, update and remove costs:

$$insert(x) = remove(x) = 1.0 \tag{5.11}$$

$$update(x, y) = 1 - \alpha_{\cos}(\mathbf{v}_x, \mathbf{v}_y) \tag{5.12}$$

The cosine similarity between words are embedded within the tree edit distance calculation as the cost of updating a node with another is 1 minus the similarity, equation 5.12. That design gives a cost of 0 if the words are identical and 1 if they are completely different. Most of the penalties are, therefore, incurred when the structure is different due to high insert and remove costs, equation 5.11.

$$\alpha_{\text{edit}}(S, T) = 1 - \frac{d(S, T)}{\max(|S|, |T|)} \tag{5.13}$$

Since the maximum edit distance can be the addition or removal of the longest sentence, we can normalise the edit distance to give us a similarity metric base on it, represented in equation 5.13. This construction is well-founded since identical sentences have an edit distance of 0, thus a similarity of 1 and the tree edit distance is commutative. However, the tree edit distance is very susceptible to structural changes by design.

S	T	d(S,T)	edit(S,T)
Hi there.	Hello there!	0.747	0.751
How is the weather?	The weather, how is it?	4.0	0.428
Is this OK?	This is OK.	0.484	0.878

Table 5.4: Examples of tree edit similarity.

Table 5.4 captures examples of tree edit similarity. Compared to greedy similarity measures, the tree edit distance is much more conservative. The second example drops in similarity due to significant structure change reflected by a distance of 4. Therefore, the tree edit similarity has very limited scope on its own when matching relative sentences and it is not used in Kevin even though implemented.

Another significant problem with the tree edit distance is the performance cost. Unlike taking dot products of vectors, the edit distance requires dynamic programming and run much slower than other metrics, summarised in table 8.2 in the evaluation section 8.2. If we are to match hundreds of rules or scan through a larger corpus, the performance hit degrades user experience and causes *lag* between the user input and Kevin's response.

5.1.6 Hybrid Similarity

To at least address the susceptibility of tree edit distance, we can combine it with a greedy similarity measure to get a hybrid similarity measure. In this case, we combined the tree edit distance (section 5.1.5) with the fuzzy similarity (section 5.1.4) using a weight w to balance them. The hybrid similarity does suffer from poor runtime performance since it not only needs to compute the expensive edit distance between sentences but also find maximal token similarity for fuzzy similarity.

$$\alpha_{\text{hybrid}}(S,T) = w\alpha_{\text{edit}}(S,T) + (1-w)\alpha_{\text{fuzzy}}(S,T) \tag{5.14}$$

Equation 5.14 describes the simple construction of the hybrid similarity. The weight w can be a function of the sentences $w = f(S,T)$ to find the optimum balance between the two similarity measures. However, for our implementation we use a constant of $w = 0.2$. Since the performance cost of hybrid is so high like tree edit similarity, it is not used within Kevin although implemented.

5.2 Sentence Unification

Since `PTrees` (section 7.2.1) can contain variables, we require a mechanism to unify sentences. This unification process allows Kevin to extract segments, arguments from sentences that it is interested in such as location names. By using variables in natural language sentences, we can generalise rules and allow them to be chained for more complex proofs. Throughout this report, we will only consider unifying two sentences represented as parse trees which might contain variables. Equation 5.15 describes the function that takes two sentences as input and returns true if the sentences are *unifiable*, false otherwise. As a side effect, the unification process can assign variable values depending on the type and implementation of the unifier function.

$$\theta : S^2 \rightarrow (\text{True},\ \text{False}) \tag{5.15}$$

We will present two unification mechanisms: strict and fuzzy unification. Both are used within Kevin at different stages and internal components. Both work on the dependency structures obtained from dependency parsing. Therefore, any discrepancies in the dependency parsing could affect unification directly.

5.2.1 Strict Unification

Strict unification is based on basic tree structure matching with extra checks for nodes. This approach is very efficient and easy to implement. To unify two sentences, strict unification follows the dependency tree structure as-is in a recursive manner. To start the process, we give the root nodes of both sentences.

$$\begin{aligned} \alpha_{\cos}(x, y) &> k \\ \text{tag}(x) &= \text{tag}(y) \\ |C_x| &= |C_y| \end{aligned} \tag{5.16}$$

At every node we perform the checks described by equations 5.16: we ensure the nodes are similar *enough*, they have the same part-of-speech tags and they have the same number of children to guarantee same structure. We do not explicitly check for dependency tags as the sentence structure as a whole is ensured to be the same. When either of the nodes are variables, if they have a value then the unification unwraps the variable's value and proceeds as normal. Otherwise, the `tag`s are compared and the variable value is assigned. The unification still continues to check the children of the variable node.

$$\theta_{\text{strict}}(\text{Mary is happy},\ \text{Mary is not happy}) = \text{False} \tag{5.17}$$

Since the strict unification follows the dependency structure, any mismatches will cause it to fail. Therefore, we only use strict unification in cases we know the sentences are sanitised and do not originate from a noisy source such as the user. For example, all internal proofs use strict unification as rules inside a knowledge base are manually written in a matching fashion. The biggest advantage over similarity based unification functions is that it can handle negated sentences. Equation 5.17 demonstrates how negated sentences do not unify even though they are very similar to the extent they differ by only the negation.

5.2.2 Fuzzy Unification

To better handle noisy input such as user queries and external text, we require a greedier unification function. Fuzzy unification was built on sentence similarities to handle extremely noisy input. By noise, we mean how different the input is from the expected; a simple measure of noise could be defined as $1 - \alpha(S, T)$ since the lower the similarity the more different, noisy the sentences are. As there are many ways to say something, just relying on a matching dependency structure performs poorly. With fuzzy unification, we take fuzzy similarity (section 5.1.4) and threshold its output while unifying variables along the way. Therefore, it acts as a greedy unification method over noisy input.

$$\theta_{\text{fuzzy}}(S,T) = \begin{cases} \text{True}, & \frac{\alpha_{\text{fuzzy}}(S,T)+\alpha_{\text{fuzzy}}(T,S)}{2} > k \\ \text{False}, & \text{otherwise} \end{cases} \tag{5.18}$$

The construction of the fuzzy similarity is straightforward and simply thresholds the output of fuzzy similarity both ways, equation 5.18. We perform the similarity measure both ways as it is not commutative. In cases where there are no variables, the fuzzy unification process is solely dependant on the output of the similarity metric. This situation is useful during information retrieval as the corpus text will not contain any variables and the queries will be grounded.

$$x_v \triangleq \exists y \in T \, \text{tag}(x) = \text{tag}(y) \wedge \text{dep}(x) = \text{dep}(y) \tag{5.19}$$

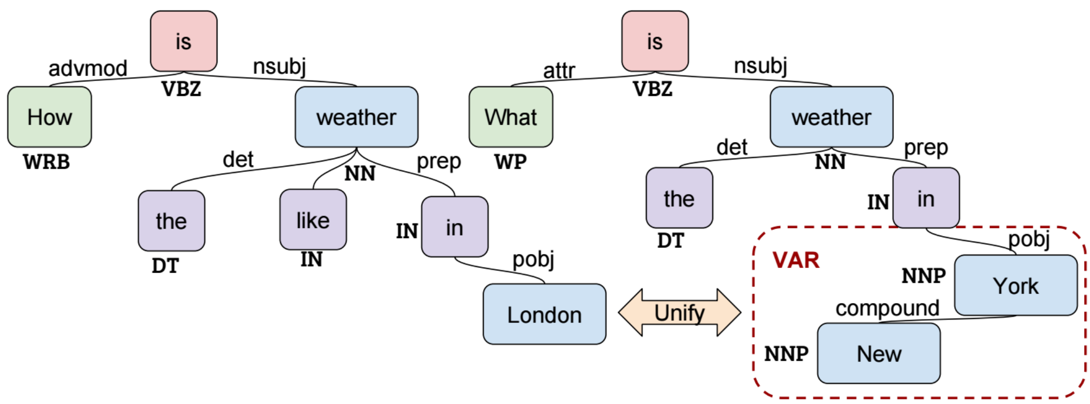


Figure 5.1: Fuzzy unification example on slightly different sentence structures.

When we encounter a variable without a value, we search for a token in the other sentence that has the same *characteristics* as the variable's placeholder token. Characteristics often include part-of-speech tags and dependency tags as in equation 5.19 but can differ based on how strict or lenient we want the unification to be. This aggressive approach works well when there is a single substructure matching the variable template such as shown in figure 5.1. When there are multiple options the first one is matched and if there is a second variable with the same template, the values are assigned in the order they appear. It is also important to note that only variables in the left sentence S are assigned values, variables in T are untouched and the unification process is run both ways. This reciprocal run is caused by lack of commutativity of fuzzy similarity.

$$\theta_{\text{fuzzy}}(\text{What is the weather in X:London;}\quad \text{In Paris, I wonder how the weather is}) = \text{True} \tag{5.20}$$

$$\theta_{\text{fuzzy}}(\text{Mary is happy; Mary is not happy}) = \text{True} \tag{5.21}$$

An successful example match with a very noisy input is shown in equation 5.20. The unification process extracts out the correct token `Paris` from the sentence and unify. However, the downside of having a similarity based unification method is revealed with simple negated sentences such as in equation 5.21. The sentences are *almost* identical and therefore have a very high similarity measure way above the set threshold to successfully unify. As a result, we use fuzzy unification on external sources such as user input and strict unification internally within a knowledge base.

Chapter 6

Deep Question Answering

In this chapter we focus on information retrieval methods used by Kevin to benefit from vast amounts of information available online and augment its text processing capabilities, particularly answering informational queries (section 3.1). When accessing external text sources such as Wikipedia[1] and news articles, it is necessary to extract the *relevant* parts of a given document. This situation is in contrast to structured API requests that external functions perform as part of intent parsing in which a query is transformed into a known action within a domain.

Information retrieval and machine question answering allows Kevin to expand its knowledge base tremendously. However, as many online resources were written for human consumption, they are very noisy and prove challenging to process. In this chapter we look at sentence selection using pre-trained neural networks and vector embeddings as well as a custom neural network to extract correct answers from sentences.

6.1 Similar Selection

When processing large chunks of text, we might want to find the most *relevant* sentence to another sentence that we are searching for. Similar sentence selection allows Kevin to extract a single sentence from a given set of sentences such as a paragraph or document using similarity measures presented in section 5.1. It turns out, when the sentence being searched is a query, we end up in a question answering scenario that efficiently builds on sentence similarities. The similarity measure then can be considered as a confidence value for the answer, the higher the more information we found the more confident we are.

$$s_a = \operatorname*{argmax}_i \alpha_{\text{fuzzy}}(q, P_i) > k \tag{6.1}$$

[1]We only use the English version throughout this project at https://en.wikipedia.org/

When we consider one of the not well-founded similarity measures such as maximal token similarity (section 5.1.3), we discovered that the metric is based on how much of the tokens are embedded within the other sentence. In other words, such metrics gave us a way of telling *how much information of the given sentence is present in another*. Thus, given a paragraph we can simply search for a sentence P_i which contains most information about our query q as described by equation 6.1. In our implementation for sentence selection we use fuzzy similarity for reasons described in section 5.2.2.

How big are microchips?
In 1958 Jack Kilby invented the microchip. **Microchips are tiny but can store lots of information.** They helped make computers smaller. In the 1970s computers were smaller and cheaper so people started to use them at home. In the 1980s computer games were very popular. Lots of people bought computers just to play games.

Table 6.1: Example sentence selection from A2 children's comprehension text.

An example is shown in table 6.1 on how the correct sentence containing the answer is selected in a children's comprehension exercise obtained from the British Council[2] (Note the selection is run on the entire text not just the excerpt provided). In this case, the selected sentence contains some information about size with token `tiny` and `microchips` which the query is interested in. Therefore, the fuzzy similarity metric is high enough to indicate that this sentence *might* contain the answer maximising the similarity value for the query and the selected sentence.

A more complicated real-world example taken from Wikipedia's YouTube page[3] summary is shown in table 6.2. The sentence selection results are underlined in the summary for each query. In this case the full context is given to demonstrate how correct and incorrect sentences are selected in an extremely noisy environment such as Wikipedia. In the first example **(1)**, the selection fails and returns the answer to *what YouTube is* rather than who created the service. This situation is a common when the query is short enough that the meaning can sway to other sentences which might not contain the answer but not always the case as in example **(2)**. In the third case **(3)**, the correct sentence is selected despite the addition noise to the query that made it longer than necessary. Finally, the fourth example **(4)**, the selected sentence contains the *most* information about making money due to tokens such as `earns` and `revenue` which boost its fuzzy similarity measure across other sentences. Therefore, by using vector embeddings as a similarity measure, the sentence selection scheme is efficient enough to handle large noisy corpus.

[2] https://learnenglishkids.britishcouncil.org/en/reading-practice/computers accessed 2017-04-14

[3] https://en.wikipedia.org/wiki/YouTube accessed 2017-05-8

As an optimisation, we pre-process the query in order not to create bias in the fuzzy similarity measure. For example, we remove the punctuation to lower the bias against sentences which contain question marks. Another optimisation is to remove stop words such as `the` and question words like `why` as they do not contribute significantly to the *meaning* of the sentence. However, this process lowers the number of tokens that the fuzzy similarity will work on especially in short queries reducing amount of information the similarity measure is looking for.

Since the underlying fuzzy similarity measure just computes vector products of tokens, sentence selection also has a good performance when dealing large chunks of text. In fact, we do not cache parsing results in Kevin when accessing external resources and can afford to re-process the texts without having a big performance impact on the response time. Another good feature from the perspective of a digital assistant is that we return full sentences that sound more natural and often contain some extra information. This situation creates a better user experience and a smoother conversation.

6.2 Guided BiDAF

Once we can select a sentence which might contain the answer of a query, we can also try to extract the answer slice from the context sentence. Assuming the answer is a continuous slice of a given sentence, the guided bi-directional attention flow model attempts to find the start and end indices of that slice token wise. Therefore, we can embed our sentence selection scheme into a deep neural network to get an answer from a wider context. The guided BiDAF model is based on the BiDAF [9] model (described in section 3.4.4); however, our network is shallower and narrower due to resource and time constraints surrounding the project.

$$\exists i, j \text{ s.t. } 0 \leq i \leq j < |S| \wedge \operatorname{argmax} P(I = i) \wedge \operatorname{argmax} P(J = j) \tag{6.2}$$

Equation 6.2 assumes the answer is in the context sentence and take the indices[4] that maximise the probability of being the start $P(I = i)$ and the end $P(J = j)$. This construction gives the final result as a slice of the original context sentence given a query, $f(S, Q) = S[i : j]$. Therefore, the input of the neural network is a context sentence S and a query Q giving the output of two discrete probability distributions that correspond to random variables I and J. If the answer is not in the context sentence then the function is undefined although it will give some, most likely wrong answer.

[4] We formulate the equations to index from 0 to match the implementation.

6.2.1 Network Architecture

The guided bidirectional attention flow model consists of five layers that given a larger context it extracts the specific answer to a given query. Since we pipe the larger context through sentence selection, the neural network is designed to work on smaller contexts usually the size of a single sentence that contains the answer unlike the original model which runs on the entire context.

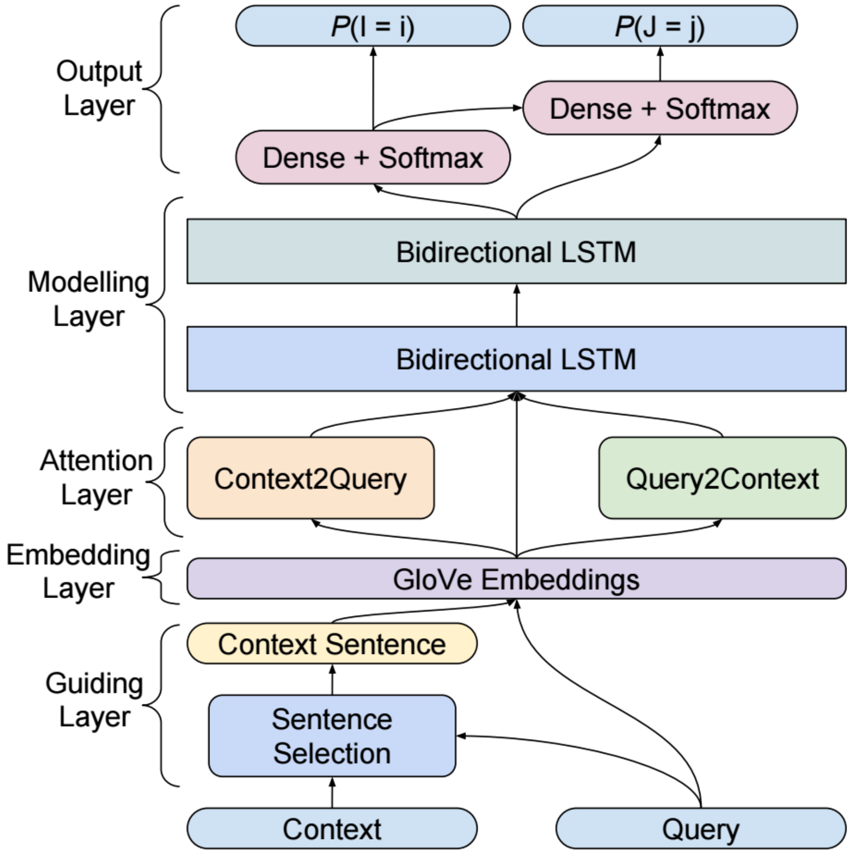


Figure 6.1: The architecture of the network used in guided BiDAF.

Figure 6.1 shows an overview of our network architecture. The network operates at the *token* level including punctuation as valid tokens. Therefore, the maximum granularity of the answers achieved will be limited to the natural language processing library's tokenisation.

1. **Guiding layer** consists of a sentence selection mechanism that given a larger context narrows down to a single sentence that might contain the answer. This layer reduces the upstream neural network size significantly since the size of the context input is now smaller. We use methods described in sentence selection section 6.1 to pick out a context sentence.

2. **Embedding layer** converts every token in the context sentence and the query into their corresponding vector representations. Vector representations might already

be used in sentence selection during similarity calculations; however, this layer re-embeds every token using the existing GloVe [3] vectors to get clean vector inputs into the upstream neural network.

3. **Attention layer** computes weighted vector representations of the context and query tokens as described by the original BiDAF model (section 3.4.4). We build the shared similarity matrix using the cosine similarity of vectors $S_ij = \alpha_{\cos}(C_i, Q_j)$ which encodes the similarity between every token in the context sentence and the query. Then the `Context2Query` and `Query2Context` is computed as defined in the paper. The output of the attention layer is concatenated with the original context embeddings to provide the information known about the context sentence together with the given query.

4. **Modelling layer** captures relationships between the context words with respect to the given query. Following the original model, we use two layers of bidirectional LSTM units (section 3.4.3) to compute new vector representations of context words. The bidirectional nature of the LSTM layers allow the network to compute the vector of a token based on tokens that appear before and after it incorporating contextual information within the sentence. The output of the modelling layer are vectors $\mathbf{M} \in \mathbb{R}^{2d \times C}$ for each context sentence token. The dimension is doubled due to the concatenation of bidirectional output of each LSTM unit.

5. **Output layer** feeds on the modelling layer to produce two discrete probability distributions that encode the slice of the context sentence as the answer. We flatten the output of the modelling layer into a 1-dimensional vector and use two dense layers with softmax activation to produce the output. The output of the start index is also fed to the end index dense unit in order to improve the predictions. As a result, the end index prediction loss is consistently lower than the start index during training.

6.2.2 Model Differences

We deviated from the original model mainly because of resource and time constraints surrounding the project. However, we also made some active changes that better suit Kevin and the network's guided nature. There are three main differences of the guided BiDAF model: it is shallower, narrower and has only dense layer at the output.

- **Shallower** than the original. We omitted the character embeddings and the contextual embedding layer entirely. These changes were done to reduce the memory footprint of the network as we didn't have the resources to train with extra two layers. The shallower network meant we degraded the accuracy of the answers and failed to recognise complex query-context pairs without a contextual embedding layer.

- **Narrower** context size since we already have a guidance mechanism to pick out a sentence which might contain the answer. Instead of passing the entire context which may contain a lot of tokens, in the region of thousands for Wikipedia summaries, we have a much smaller context that only accommodates a single sentence. This reduction meant all upstream layers were narrowed with less LSTM units; thus the training time for the network got reduced enough to be reasonable on a CPU.

- **Dense at output layer** instead of an LSTM unit on the end index prediction. Since the network already lacked rich contextual embeddings, an LSTM didn't contribute to a better end index prediction. Therefore, we converted the layer to a dense and gained a performance improvement to again reduce our training time.

The BiDAF model was designed to process the entire context as a whole of the SQuAD [71] dataset; therefore, it cannot answer questions over large context. On the other hand, the guided BiDAF can scan the large corpus to select a sentence and extract an answer. As a result, Kevin can answer queries over large context compared to a limited context size of the original implementation.

The sentence selection does not have be limited to a single sentence. We can indeed return a window around the most likely sentence for the neural network to find an answer. However, that would require a larger context input size for the neural network increasing the size of each layer. This approach might be useful for the original BiDAF model such that it wouldn't be limited to fixed size contexts but can be *guided* by some context selection mechanism.

6.2.3 Training and Limitations

We implemented guided BiDAF using Keras [73], a wrapper library for TensorFlow [2]. We used a maximum context token size of 60 and a query token size of 24. Therefore, the network does not handle any sentences larger than the maximum input size. The original paper reported a training time of 20 hours on two Titan X GPUs. However, due to the reductions made to the guided version, we were able to train our guided model in 16 hours on a Intel i7-4790[5] CPU. The network was trained using the SQuAD [71] dataset with a heuristic to check whether the sentence selection returned a sentence which contained the answer. As such, we ignored training examples in which the sentence selection failed as well as the sentences which did not fit our network size. We are not

[5] http://ark.intel.com/products/80806/Intel-Core-i7-4790-Processor-8M-Cache-up-to-4_00-GHz reference last accessed 2017-05-21

certain if these omissions in the training set affected the model result in a noticeable manner. As anticipated, the guided model did not perform as good as the original due to the omissions and restrictions achieving a loss value of ≈ 4.21 on the training set.

The reductions in the network meant it ran much faster and provided a lightweight integration into Kevin. However, the question answering accuracy is severely reduced to the extent that it got a exact match score of 23.6% on the corresponding dev dataset. We analysed why the exact match score was extremely low and identified few key limitations of the guided model: wrong sentence selection, no answers across sentences, no contextual distinction, lack of character embeddings and limited context size:

- **Wrong sentence selection** occurs on the context and renders the upstream neural network answer extraction impossible. If the wrong sentence is selected that does not contain a slice for the answer, the guided model fails at the first layer. The output becomes some maximal probability over the wrong sentence finding similar answers such as a number if the query was looking for a numerical answer.
- **No answers across sentences** can be outputted. Since we select a single sentence which might contain the answer, if the answer spans across sentences, for example with co-references, the guided model again fails at the first sentence selection layer. It cannot handle multi-sentence contexts by design and there are frequent cases of this situation in the dev dataset obtained from the Wikipedia articles.
- **No contextual distinction** is made in the selected context sentence since the model lacks the contextual embedding layer. Therefore, the guided model cannot answer complex relationships such as distinguishing `may` and `May` tokens. It is strictly limited to single word embeddings provided by the GloVe embedding layer.
- **Numerical and symbol based** answers such as dates and times are not properly recognised since the model lacks a character embedding layer. For example, measurement units, percentages, different date notations and information within parenthesis are often out of reach of the model's understanding. Since such cases are common on factual Wikipedia pages, the model performs very poorly.
- **Limited context size** caused most queries not to be considered and got a score of zero by default. The context size was not adjusted to fit the data but to provide a reasonable training time to handle shorter sentences.

In the worst case, the neural network seems to output the entire context sentence back. We believe this behaviour was caused by sentence selection in the training phase. Since all context sentences were checked to contain the answer, the neural network at worst would just have to output the sentence back knowing it contains the answer. Therefore, the entire neural network is coupled to the sentence selection and in such cases act as the identity function on the context sentence provided.

6.2.4 Output Examples

Despite the limitations of the guided model, it can extract some answers from correctly selected context sentences. This neural network integration allows Kevin to handle noisy input at a finer granularity then just sentence selection or maximal token matching with fuzzy unification.

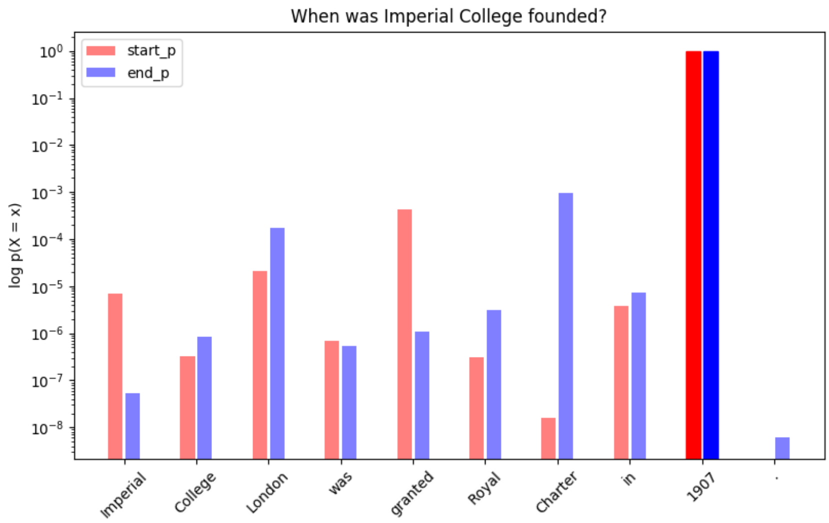


Figure 6.2: Factual answer extraction using guided BiDAF.

Figure 6.2 demonstrates a single token answer extraction on a factual sentence. In this case, the probabilities of both the start and end indices are highest at the `1907` token. Thus, the correct answer is returned from the neural network. Due to the `when` token in the query, the neural network looks for a token representing date or time. The year is the only instance of such a token and as a result the confidence, the joint probability of the answer is almost 1.

A more complicated answer is presented in figure 6.3 representing a causal relationship between John and the reason he is tired. The neural network correctly extracts "a hard working student" as the answer. When we analyse this example, we notice that besides John getting tired, the only piece information provided is about him being a hard working student. Therefore, the networks picks up on that information. We notice that the second choice for the indices is the token `tired` most likely because it is already contained in the query. This example also demonstrates a slice, range of the context being returned as the answer rather than a single token.

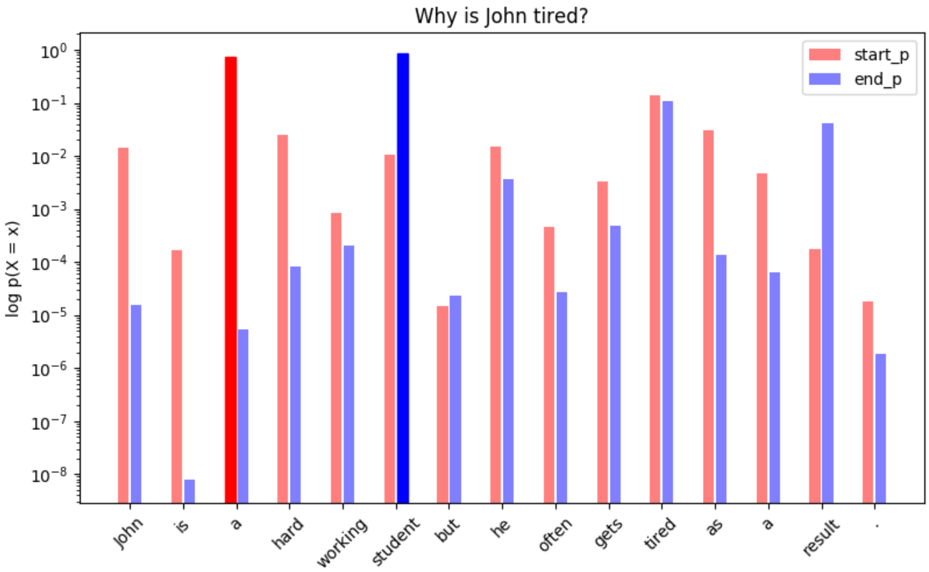


Figure 6.3: Causal answer extraction using guided BiDAF.

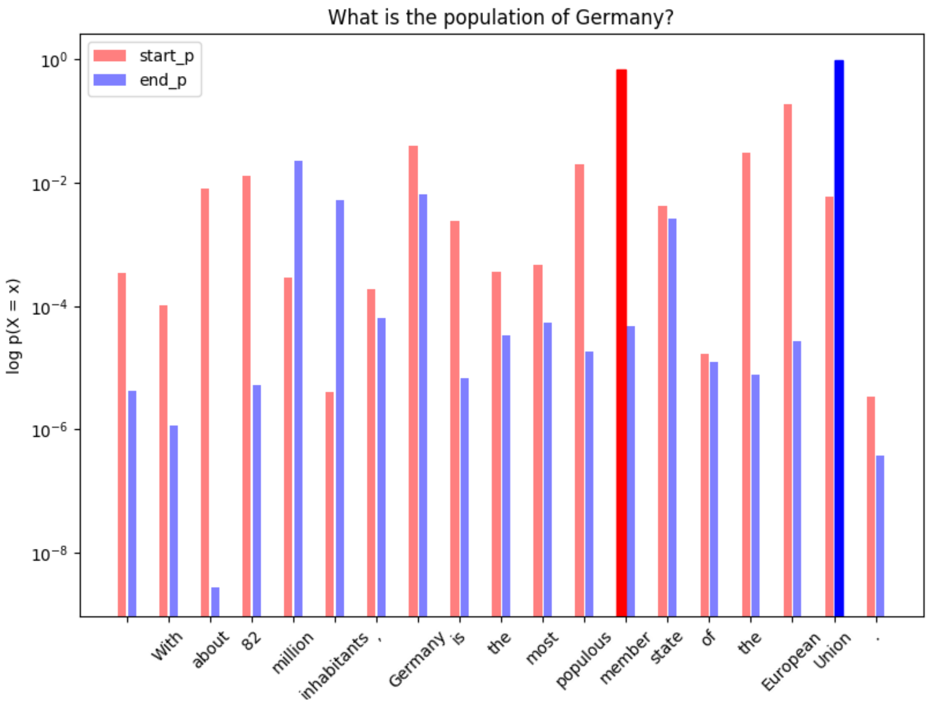


Figure 6.4: Failed answer extraction example of guided BiDAF.

We observed a failed example from a sentence selected from Germany's Wikipedia page[6]. The neural network outputs "member state of the European Union" as the answer instead of "86 million". We believe the reason for this incorrect answer is the lack of contextual embedding layer. With just word embeddings, the probability distribution

[6] https://en.wikipedia.org/wiki/Germany accessed 2017-05-28

spikes after a very similar word token. In this case, the token `population` in the query skews the distribution towards and after the token `populous` in the context sentence. The original BiDAF model does answer this query correctly on the entire context of the Wikipedia summary.

(1)Who founded YouTube?
(2)Who owns YouTube?
(3)Do people need to register to watch videos on YouTube?
(4)How does YouTube make money?

YouTube is an American video-sharing website headquartered in **(1)San Bruno, California**. The service was created by three former PayPal employees – Chad Hurley, Steve Chen, and Jawed Karim – in February 2005. **(2)Google** bought the site in November 2006 for US$1.65 billion; YouTube now operates as one of Google's subsidiaries. The site allows users to upload, view, rate, share, add to favourites, report and comment on videos, subscribe to other users, and it makes use of WebM, H.264/MPEG-4 AVC, and Adobe Flash Video technology to display a wide variety of user-generated and corporate media videos. Available content includes video clips, TV show clips, music videos, short and documentary films, audio recordings, movie trailers and other content such as video blogging, short original videos, and educational videos.
Most of the content on YouTube has been uploaded by individuals, but media corporations including CBS, the BBC, Vevo, and Hulu offer some of their material via YouTube as part of the YouTube partnership program. Unregistered users can only watch videos on the site, while registered users are permitted to upload an **(3)unlimited number of videos and add comments to videos**. Videos deemed potentially offensive are available only to registered users affirming themselves to be at least 18 years old.
YouTube earns advertising revenue from Google **(4)AdSense**, a program which targets ads according to site content and audience. The vast majority of its videos are free to view, but there are exceptions, including subscription-based premium channels, film rentals, as well as YouTube Red, a subscription service offering ad-free access to the website and access to exclusive content made in partnership with existing users. As of February 2017, there are more than 400 hours of content uploaded to YouTube each minute, and one billion hours of content is watched on YouTube every day. As of April 2017, the website is ranked as the second most popular site in the world by Alexa Internet, a web traffic analysis company.

Table 6.2: Guided BiDAF with sentence selection underlined on Wikipedia summary of YouTube.

Table 6.2 shows the results of the guided BiDAF model together with the sentence selections underlined. In the first case **(1)**, since the sentence selection fails, the neural network has no chance of getting the correct answer. Instead, it outputs *where* YouTube specifically is because it was looking for an entity that might have founded YouTube. The second query **(2)** is correctly answered as the sentence selection was correct; the context sentence contains two slices with Google and the network selects the first one as buying something is often related to owning it. The third case **(3)** is an example in which the sentence selection succeeds but the neural network fails to extract the correct answer. Since context sentences often do not contain `yes` or `no` tokens the network struggles to figure out what the query requires. As a result, the closest answer to *what* registered people can do with videos on YouTube is extracted. Finally, the fourth case **(4)** is the best example in which both the sentence selection and the neural network working together to pinpoint the correct answer "AdSense".

Overall, the guided model can efficiently scan through large context and then use a smaller neural network to extract the answer. However, the omission of character embeddings and contextual layers significantly reduces its accuracy as well as failing when the guiding layer incorrectly selects the context.

Chapter 7

Implementation

In this chapter we cover the implementation of Kevin and how it is organised internally. We first look at the natural language processing libraries used as a foundational layer; then cover the internal representations of objects handled. Finally, we detail out the query pipeline and how an input is processed by Kevin.

7.1 NLP Back End

Kevin relies heavily on downstream natural language processing tasks such as part-of-speech tagging (appendix A.1) and dependency parsing (appendix A.2) to be already handled before any of its components start to manipulate the input. Thus Kevin builds on top of these libraries and assumes a working foundation. During the project, we considered two natural language processing libraries: Spacy from Explosion AI [74] and CoreNLP from Stanford [75].

Although we experimented with both libraries, Kevin exclusively uses Spacy to process all natural language input and output. This design choice is mainly influenced by the dependency parsing discrepancies in CoreNLP that Kevin was not designed to handle elaborated in the CoreNLP section 7.1.2. Both libraries present major consistency problems in parsing and yield a challenging foundation to build an upstream framework that assumes the lower level tasks are successful. Often many failures such as not recognising an intent manifest themselves in the form a failed parsing or entity recognition at the natural language processing level.

7.1.1 Spacy

Spacy is an natural language processing library that provides: tokenization, part-of-speech tagging, dependency parsing, named entity recognition and automatic vector embedding using GloVe (section 3.4.2). This diverse and rich tool-set allows Kevin to focus on upstream tasks with one exception: co-reference resolution. The library does not provide co-reference resolution and we had to therefore implement basic co-reference resolution in order to accommodate basic follow-up questions with pronouns.

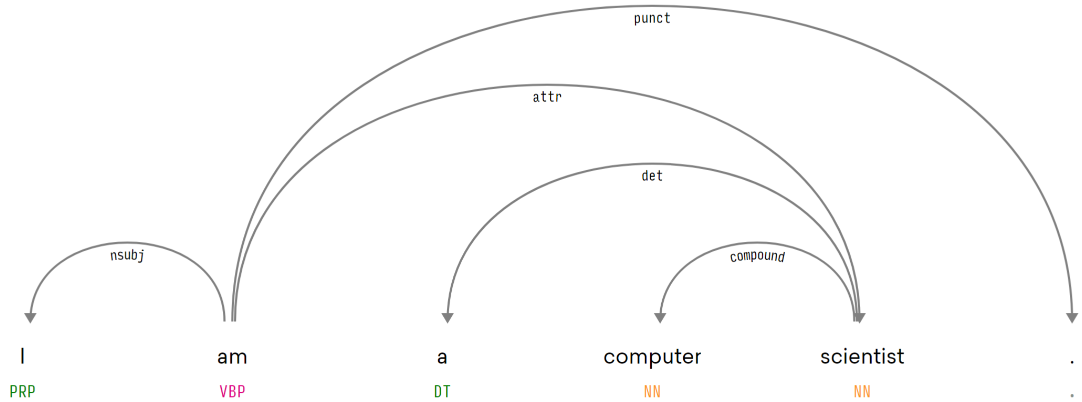
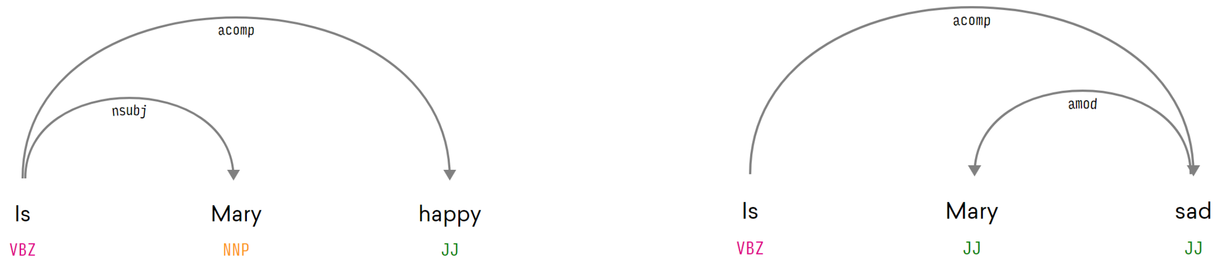


Figure 7.1: Example parsed sentence using Spacy.

Figure 7.1 visualises the output of Spacy[1] on a sentence and how part of speech tagging is combined with dependency parsing. It is important to note that the dependency structure is a tree with the root representing the binding verb (VB) of the sentence. These outputs are then used as-is by Kevin to construct internal representations and manipulate natural language.



(a) Correct proper noun (NNP) parsing (b) Incorrect adjective (JJ) parsing

Figure 7.2: Example of complete structure change and failed parse in Spacy.

There are many examples in which Spacy parses incorrectly even if a slight change is applied. In figure 7.2a, the sentence "Is Mary happy" is correctly parsed and Mary is identified as a proper noun (NNP); however just changing the adjective completely disfigures the parsing and outputs an incorrect representation seen in figure 7.2b. Such failures cause major disruptions in upstream processing as Kevin relies on correct parsing. When there are discrepancies in short simple queries, Kevin struggles to comprehend what the user it trying to say and subsequently fail recognising an intent, query or even a basic command.

[1]Using Displacy at `https://demos.explosion.ai/displacy/` accessed on 2017-05-06.

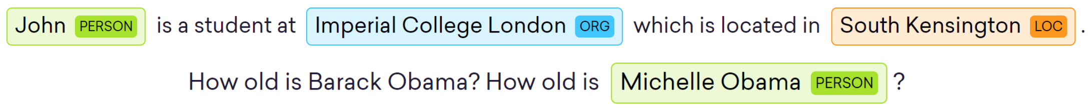


Figure 7.3: Named entity success and failure cases in Spacy.

Figure 7.3 demonstrates[2] cases in which the named entity recognition (NER) in Spacy succeed and fail. In the failure case, despite identical sentence structure "Barack Obama" is not recognised as a named entity. This situation is again a major problem for upstream processing. For example, if an intent such as "How old is someone?" is looking for a named entity person, asking the age of "Barack Obama" will fail not because the intent was incorrect but because the person wasn't recognised in the query.

We try to compensate these parsing errors during upstream processing such as ignoring entity recognition in certain cases to allow the intent parsing to function as expected. However, this situation creates an unstable infrastructure since in general the number of checks performed on the parse output is reduced. For example, to avoid the case in figure 7.3, we can drop named entity recognition and just rely on part-of-speech tags to give us proper nouns (NNP) to match an intent. As a result, the intent becomes less reliable since a proper noun could be a location name as well as person, forcing the intent to output the age of a location resulting in undefined behaviour.

7.1.2 CoreNLP

CoreNLP is a suite of natural language processing pipelines which when combined together can provide tokenization, part-of-speech tagging, dependency parsing, named entity recognition and co-reference resolution [75]. It does not bind vector representations by default but provides more robust named entity recognition and a co-reference resolution that Spacy cannot handle well. The vector representations of words can be attached as an extra step at the end of the pipeline.

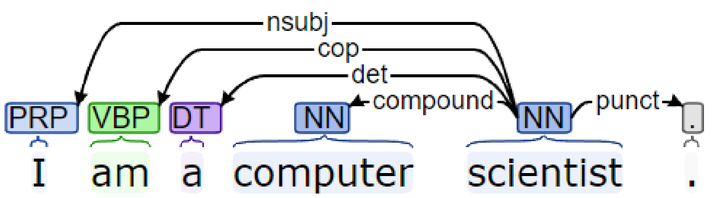


Figure 7.4: CoreNLP example parsed sentence output.

Figure 7.4 demonstrate and example output of CoreNLP[3] as well as the reason why we decided not to use it as Kevin's primary natural language processing library. In

[2]Using Display-Ent at https://demos.explosion.ai/displacy-ent/ access on 2017-05-06.
[3]Using the server visualisation tools that are packaged with CoreNLP.

the example sentence, the dependency structure is rooted at `scientist` rather than the binding verb `am`. This root structure change violates a design principle which states that the root should be a verb if any so that the arguments would be sub trees. Kevin is designed to use substructures which bind under the root verb in order to extract arguments and recognise intents. However, in this output case that assumption is false and Kevin no longer functions properly and fails for example to recognise that the user is a computer scientist.

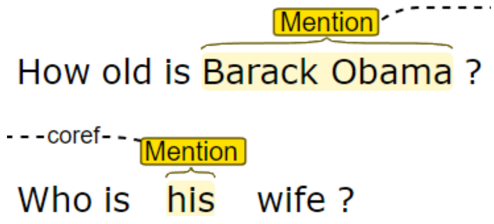


Figure 7.5: CoreNLP co-reference resolution example.

Despite the challenges in the dependency parsing with CoreNLP, the named entity recognition is more robust than Spacy and provides co-reference resolution as shown in figure 7.5. The parsing recognises "Barack Obama" and successfully resolves "his" in the follow-up sentence. These features play an important role in following a conversation and understanding the context from the perspective of a digital assistant.

We decided not to use CoreNLP in our project because of the fact that the dependency parsing often violates the assumption that the root of the tree is the binding verb. This assumption turned out to be so important in the design to overcome other features of CoreNLP such as co-reference resolution. The crucial component that gets affected is the similarity measures (section 5.1) discussed in this report that expect the root dependency structure to be based on verbs and arguments as children.

7.2 Internal Representation

In this section we discuss the constructs manipulated by Kevin internally. We present the Parse Trees (`PTree`) that wraps the output of a natural language processing library as the framework primitive and then build aggregate structures such as `Rules`. These objects form the basis of Kevin's framework and his behaviour; they are summarised in table 7.1.

Although the objects present a general abstraction for a natural language framework, they closely follow our implementation in Python and their corresponding class names for ease of reference. In complex constructs such as neural network integration, these classes will get overridden to provide a transparent mixture of external components, detailed

PTree	Abstracts a parse tree output from NLP libraries.
Var	Placeholder for PTrees inside other parse trees.
Expr	Represents a provable expression wrapping various other objects.
Rule	Denotes expressions in the form of material implication.
KB	Aggregates rules into knowledge bases that Kevin accesses.

Table 7.1: Summary of internal objects used by Kevin.

in the free form section 4.8. The external components can and do use these internal constructs to interact with Kevin internally such as asking an expression to be proved.

7.2.1 Parse Trees (PTree)

Parse Trees (`PTree`) wrap around the output of a natural language processing library to have consistent internal representation of sentences. They form the building blocks for any natural language abstraction in Kevin. A `PTree` is an ordered recursive tree structure in which each node represents a token in the original sentence. They are designed to capture individual sentences and a collection of them represent a parsed document.

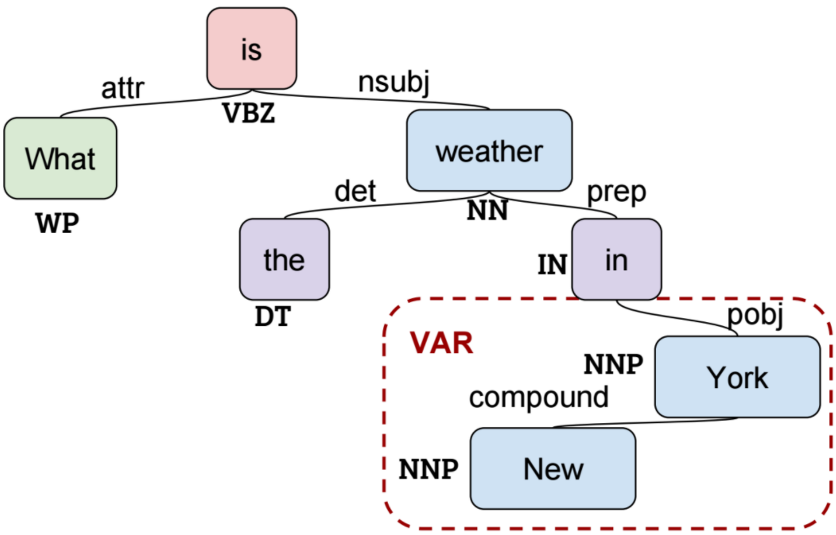


Figure 7.6: Diagram showing a Parse Tree (PTree).

Figure 7.6 provides a visual representation of a `PTree` capturing "What is the weather in X:New York". It looks very similar to output of the natural language processing libraries (figure 7.1) because in its basic form it wraps around every token following the dependency structure. Thus, the in-order traversal of a `PTree` gives back the sentence that was parsed. An important feature is the variable subtree denoted `VAR`. A variable subtree allows another `PTree` to hold its place giving way to unifications discussed in section 5.2. These subtrees can vary in length and do not necessarily need to match the structure of the variable template; any other subtree can hold its place.

7.2.2 Expressions (Expr)

Expressions are provable objects that wrap around other primitives such as parse trees as well as other expressions. By provable we mean they yield semantics that the proof engine will invoke in order to *prove* an expression. The meaning varies with the objects or expressions being wrapped and is summarised in table 7.2. By using the base `Expr`, we can create new expressions with different semantics that will integrate into the logic framework. However, in either case, the upstream components are oblivious to the internal behaviour of an expression for example whether they call external APIs or not.

`ExprPTree`	Wraps around a `PTree`.
`ExprCall`	Represents a callable external function.
`ExprNbf`	Negation by failure of another expression.
`ExprAnd`	Denotes logical AND of two expressions.
`ExprOr`	Denotes logical OR of two expressions.
`ExprRule`	Wraps another entire `Rule`.

Table 7.2: Different kinds of `Expressions` in Kevin.

The semantics of expressions depend on the objects they represent. While some expressions have the expected meaning such as `ExprAnd` meaning logical AND, others such as `ExprCall` maintain different semantics. Often user given input is regarded as already proved with some exceptions such as constraint checking. The proof semantics of expressions are summarised in the table 7.3.

Expression Type	Proof Semantics
`ExprPTree`	Must be user given or head of a `Rule`.
`ExprCall`	Wrapped function must run without errors.
`ExprNbf`	Wrapped expression proof must fail.
`ExprAnd`	All sub expressions must succeed.
`ExprOr`	At least one sub expression must succeed.
`ExprRule`	Follows material implication proof of `Rule`.

Table 7.3: Proof semantics of `Expressions`.

Besides the proof semantics, every expressions maintains a *resolution* (display) semantics that allow Kevin to present the response expression to the user. This meaning of an expression is completely different from its proof semantics and is invoked after the proof engine is complete, detailed in query pipeline section 7.3. In general, the display semantics tell Kevin how to convert the result expression of the logic framework into a string. The summary of the display semantics of expressions are provided in table 7.4.

As with the proof semantics, when new expressions are created their display semantics could be altered. When displaying an expression it is allowed to have side effects which many external functions such as changing context rely on. Having a soup of semantics

Expression Type	Display Semantics
`ExprPTree`	String representation of underlying `Ptree`.
`ExprCall`	Return value of wrapped external function.
`ExprNbf`	Display wrapped expression, often not used.
`ExprAnd`	Display concatenation of all sub expressions.
`ExprOr`	Randomly select a sub expression to display.
`ExprRule`	Add wrapped `Rule` to local context.

Table 7.4: Display semantics of `Expr`essions.

proves debugging difficult, however, provides flexibility to extend the framework beyond logical constructs.

Classical Negation (ExprNot)

Although Kevin has an internal representation to express classical negation (`ExprNot`), it is not implemented. The reason we omitted this construct is because we push the negation into the sentences rather than capturing them at the expression level. It is not always the case that the negated form the of the sentence in natural language denotes the classical negated version of the positive sentence; however, those cases do not arise often in the context of this project to be a problem.

$$\neg(\text{Mary is happy.}) \leftrightarrow \text{Mary is not happy.}$$

In the example case above the semantics of having a classical negation outside of a positive sentence is equivalent to having a negated sentence. For that reason, Kevin currently does not handle classical negation although the infrastructure for it does exist.

7.2.3 Rules (Rule)

A `Rule` captures multiple expressions into processable units that comprise of different semantics depending on the stage in pipeline. Therefore, they do not strictly represent material implication semantics. Rules can be viewed as a collection of expressions at different positions such as the head, query and body.

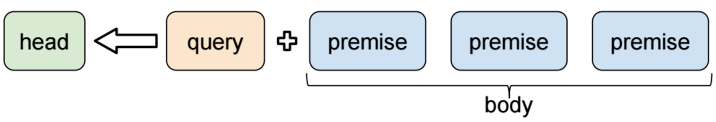


Figure 7.7: Simplified representation of a Rule.

Figure 7.7 shows how expressions are organised within a rule. The head comprises of a single expression that the rule *supports*. The query represents user input that can trigger the rule. It is important to note that not every rule has a user query trigger and it could be empty as such. Finally, the body is a collection of premise expressions that need to be satisfied in order for the head to be obtained.

$$\text{It is X:22 degrees.} \leftarrow \text{What is the temperature?; weather} : () \rightarrow X$$

In the above representation we see a more complex interaction incorporating query `ExprPTree` that invokes the rule which retrieves the weather information via an `ExprCall`. The temperature result is substituted into the variable X that is outputted to the user as part of the display semantics of the parse tree.

Ephemeral Rules

An ephemeral rule is a rule that acts and behaves the same as a normal rule with one exception: once a answer expression is proved, if the proof chain contains an ephemeral rule, it is removed from its corresponding knowledge base. Ephemeral rules allow temporary reasoning deformations that change the behaviour of Kevin. They are not widely used except in certain niche cases such as Easter egg follow-ups that are only invoked after a certain catch phrase is picked up.

7.2.4 Knowledge Bases (KB)

A `KnowledgeBase` is a set of rules that Kevin interacts with in order to answer user queries. Depending on how its interpreted, a knowledge base can form a *deductive database* [68] in which the contents of a knowledge base is the logical consequence of its rules. At runtime, a knowledge base might shrink by removing rules or grow by addition of rules. There is no condition that the contents of a knowledge base be consistent and in fact end up in an inconsistent state if the correct rules and constraints are not added or checked. This situation creates a bag of unchecked rules that the proof engine tries to makes sense of following the proof chain. It is important that *a knowledge base does not impose any logic programming semantics*, just collects rules into processable units.

Kevin can maintain multiple knowledge bases at the same time in the order of preference. This structure allows Kevin to have user customised knowledge bases that hold rules specific for that user as well as common knowledge bases that share rules for all users. For example, a user might change Kevin's name to something else that would be recorded in a user specific knowledge base and if a user adds a capability to check weather information that could be added to a common base. Thus, knowledge bases can organise rules for personalisation and shared knowledge.

7.3 Query Pipeline

In this section we describe the steps Kevin takes in order to process user input. Kevin has a multi-stage pipelined architecture that can short circuit at any point to give an output. From a birds eye view point, Kevin gets a text input and produces a text output handled by input and output modules. The *interface* to Kevin is modular and can take different forms such as voice input piped through an external speech-recognition module. Therefore, at its core Kevin is a service that interfaces to end points such as web and mobile that provide plain text to be processed.

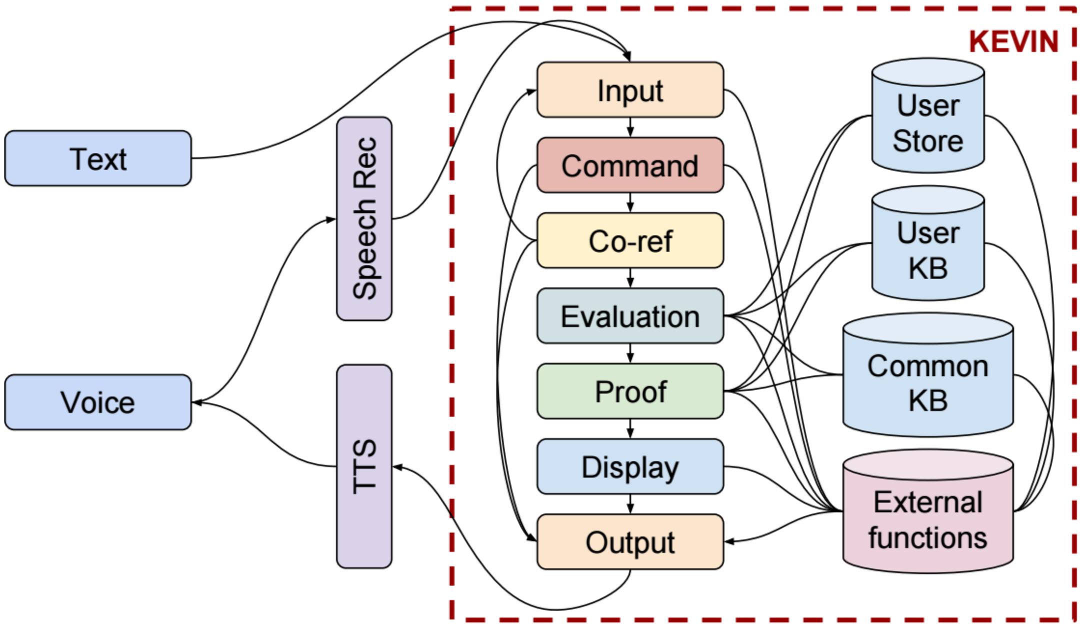


Figure 7.8: Monolithic multi-stage Kevin architecture diagram.

Figure 7.8 displays how different modules interact within Kevin along with the knowledge bases to process user input. The flow isn't necessarily linear and can be redirected based on what the user input triggers. This architecture proves difficult to debug as interactions between main modules are tightly coupled within an instance with many redirections possible due to side effects. A distributed architecture can be formed by separating the data from the main modules.

7.3.1 Input Module

The input module acts as the point of entry for all requests made to Kevin. It accepts plain text input as well as `ExprPTrees` (section 7.2.2). The functionality of the input module is as follows:

- **Convert input into internal representation** if it is just plain text. Plain text input is converted into `ExprPTree`. If it is already an `Expression`, then it passed unmodified.

- **Store history of past inputs** that accumulate as context for other components such as co-reference resolution. It is implemented as a queue of a certain history size.

When a user query is given, it must first arrive at the input module. However, when other components, even external functions interact with Kevin, they can bypass the input module because the instance of Kevin is exposed within after passing the input module. This design creates a rather convoluted execution as almost every module can reach any other module; however, given the experimental nature of the project we wanted to have maximum flexibility not knowing how the implementation would work out from the start.

7.3.2 Command Module

The command module is responsible of redirecting input to other modules and external functions. As a result, it can short circuit the system bypassing the standard flow. For example, an external function that requires follow-up user input such as a location for a weather intent, can ask the command module to redirect the next input to a callback function. Then the user's next input will invoke the callback function which in this case could output the weather information at the given location.

The command module also allows direct commands to be executed within Kevin to probe and debug its internal state. Commands are considered the same as external functions but are invoked with a forward slash in front. For example, `/reload [kbname]` will reload the knowledge base with the given name. When a command is recognised, the corresponding function is called and the return value is passed to the output module directly for user display short circuiting the pipeline.

The non-linear flow of the command module renders debugging difficult. This design is not well suited for multiple chained callbacks. For example, if a external function redirects input into another external function which also requires input, the query pipeline is short circuited twice; if the user provides unexpected inputs the behaviour of Kevin is undefined as the inputs are redirected away from its linear flow. As a result, we try not to chain redirects within the command module and consider simpler cases.

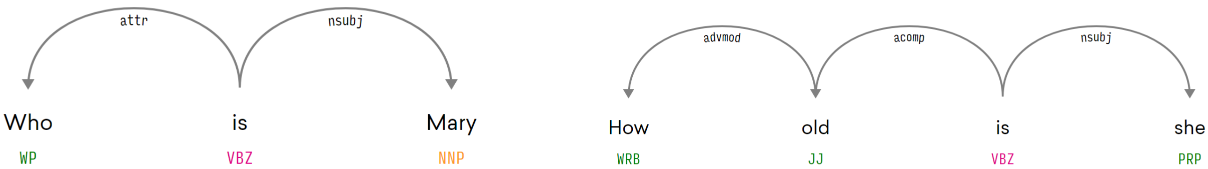
7.3.3 Basic Co-reference

The co-reference module attempts to resolve basic cases of pronouns across chained inputs. It tries to fill the lack of co-reference resolution feature in the underlying natural language processing library Spacy (section 7.1.1). It is designed to handle simple cases in which the pronouns occupy the expected subtree structure in the sentence. Therefore, it is a greedy dependency structure based co-reference resolution.

$$s' = f(s_u, S_p) \tag{7.1}$$

$$\exists i, j \text{ s.t. } dep(t_u) = dep(S_{ij}) \wedge tag(S_{ij}) = \text{NN} \tag{7.2}$$

The resolution follows equation 7.1 in which a user sentence s_u with a pronoun and a set of past sentences S_p are given to form a new sentence s'. Equation 7.2 expresses the dependency based co-reference resolution: if jth token of the ith sentence in the past sentences is a noun (NN) and matches the dependency of the pronoun, we substitute the pronoun. It is a very primitive *greedy* algorithm as it will match the first successful occurrence in the past sentences S. The previous inputs are searched from most recent and the resolution stops when a match is found.



(a) Corresponding past query (b) Query requiring resolution

Figure 7.9: Basic co-reference resolution based on dependency parsing.

An example is shown in figure 7.9b in which the pronoun "she" needs to be resolved. In these basic cases, if the past queries contains a sentence such as figure 7.9a, the algorithm successfully resolves she $\rightarrow$ Mary. The resolution succeeds on the dependency `nsubj` which both the pronoun and the proper noun follow. Even a basic co-reference resolution renders the interaction more fluid and easy for the user by following a more natural conversation.

7.3.4 Query Evaluation

When an expression reaches this point in the execution, an entry rule needs to be found in order for the proof engine to start. Therefore, the query evaluation module searches the knowledge bases for an appropriate rule that will allow a user given expression to be proved. The query part of a `Rule` (section 7.2.3) is used at this stage to match a successful rule. When a rule is matched, the `head` expression of the rule along with the rule is passed to the proof engine to prove the `head` starting from the matched rule. The head of the rule often becomes the answer to the user query if proved successful. Thus, the evaluation module figures out what Kevin should do for a given input.

$$r = \underset{r \in C}{\operatorname{argmax}}\ \alpha(e_u, r_q) \tag{7.3}$$

Given a similarity metric α (described in section 5.1), equation 7.3 captures the rule with the most similar query part r_q to the user given expression e_u for a given context C. This rule is used as the starting point for the proof procedure. It is important to note that the search for matching rule is limited to only the query part of a rule. This allows the evaluation module to find a match in a single pass across the knowledge bases and limits possible false positives that might appear in the body of other rules.

At worst case, the query parts of all the rules have to be checked giving $O(n)$ performance. However, the rules can be sorted based on frequency and a threshold can be applied so the search is terminated when a *good enough* rule is found. In practice this is often not a concern as most rules for specific applications are contained within contexts which significantly reduce the search space as only rules within the active context are considered.

7.3.5 Proof Engine

The proof engine is responsible of running through rules and satisfying expressions, proving them along the way. The proof semantics of `Expressions` are summarised in table 7.3 in section 7.2.2. The proof engine considers one rule at a time, proving strictly from left to right. Therefore, re-ordering the premises might change the semantics depending on how expressions handle internal state and backtracking. This situation is more of a concern with external functions as the order in which they are invoked might change the overall behaviour.

$$r = \underset{r \in C \wedge r \notin R}{\operatorname{argmax}}\ \alpha(e, r_h) \tag{7.4}$$

The starting point for the proof engine is an expression to prove e and a set of rules R which it cannot access. If a rule r is given, the proof engine starts to prove e using r, otherwise it searches the head of rules for a maximal match, described in equation 7.4 with a similarity metric α. The former case often arises from the evaluation module already passing a rule to start with and the head expression to be proved as an answer so that the search is not repeated.

$$\begin{aligned} \text{Mary is happy} &\leftarrow \text{Mary is not sad} \\ \text{Mary is not sad} &\leftarrow \text{Mary is happy} \end{aligned} \tag{7.5}$$

The proof engine performs a *depth first search* over the possible proof space similar to SLDNF (section 3.3.2). However, a crucial difference is during the search all active rules R in a search branch are remembered. This set of inaccessible rules ensures that a rule cannot jump to an ancestor rule that might cause a loop as shown in rules 7.5. Once the proof engine reaches a rule it has used before in the current branch, it will not be able to use it again; thus in the example case it not be able to prove either "Mary is happy" or "Mary is not happy" as they are dependant on each other.

The proof engine also stores the current state of the search for every expression for backtracking. The expressions can behave differently during backtracking depending on their implementation. For example, external functions might return different results based whether they are invoked a second time during backtracking. All these non-uniformities yield an operational semantics rather than a logical one. *When the operational semantics are unknown, the behaviour of Kevin is undefined.*

7.3.6 Resolution (Display)

The display module tries to convert the proven answer expression into something that could be displayed to the user commonly converting it to plain text. Most often this is a parse tree (`PTree`) that is converted to a string. However, expressions might have different display semantics which are summarised in table 7.4 in section 7.2.2. The display module invokes the display function of the expression that is passed to it and passes the result to the output module.

The display module does some extra checks on top of invoking a display function on the expression. It ensures certain conditions are satisfied before the result is outputted to the user. These checks safe guard the user from possible internal failures. The checks performed by the display module are summarised below:

- **Variable check**, if a parse tree is being displayed, ensure that all variables are bound if any. This check ensures that the user does not get a partial answer from a variable containing parse tree.
- **Check external function return** is not empty. If a `ExprCall` is being displayed, the function might return an empty result which display module handles. Otherwise the user might get a blank response.

Given the possible short circuits within the query pipeline, the checks performed by the display module can be bypassed. For example, a direct command will invoke the output module in the current implementation so that internal messages are displayed as is avoiding the display module entirely.

7.3.7 Output Module

The output module binds to interfaces that allow some form of output to be displayed to the user. The final output of Kevin is often plain text. Therefore, other components such as text-to-speech (TTS) are connected externally. The crucial role the output module plays is to send a response to every interface that a user might be bound to. For example, if a user is connected from multiple clients using difference interfaces such as web and terminal, the output module pushes the results downstream to all those interfaces. However, it is also possible for a single interface to handle multiple instances such as a web application with multiple users. In general, an instance of Kevin is bound a single interface to keep executions isolated and easier to debug when errors occur.

The module also provides a shortcut for any internal component to *say* something to the user. This feature is often used in external functions to output intermediary results or by timers which invoke the output module to inform the user. The internal exception handling mechanism also invokes the output module when `ProofError`s are raised at any given time. Therefore, the output module can act as an exit fast strategy for informing the user about something.

Chapter 8

Evaluation

The evaluation chapter concentrates on testing key parts of Kevin on independent datasets. We picked out two main recurring themes that allow Kevin to function and tested the underlying concepts on comparable datasets. First, we will look at similarity measures described in section 5.1 and how they compare on a similar question dataset provided by Quora [4]. Secondly, the sentence selection built on top of the similarity metric is evaluated against the SQuAD dataset [71] which is also used to train the guided BiDAF model (section 6.2). Finally, we look at example interactions achieved by Kevin on real user case scenarios and their corresponding limitations.

The evaluation methods discussed are not exhaustive enough to draw definitive conclusions about Kevin as a whole since there are many sub components with different use cases. Due to the heterogeneous structure, it is difficult find a dataset or create a scenario that would judge Kevin end-to-end. As a result, we use the evaluations in this chapter as indicators to understand the benefits of some critical components as well as their limitations and not to evaluate Kevin as a digital assistant.

8.1 Similarity Metrics

In this section, we try to evaluate the various types of similarity measures presented in the sentence similarity section 5.1. We take the first public dataset made available by Quora [4], an online question answering platform. The dataset contains around 400,000 question pairs that are annotated as whether they are *logically distinct* or not. In other words, if the questions pairs can be answered the same then they are identified as duplicates. There have been attempts at determining whether a question pair is duplicate or not, for example using tree-structured LSTMs [76]; however, in our case, we are interested how our similarity measures behave given the data set and not necessarily try to predict duplicity.

id	qid1	qid2	question1	question2	is_duplicate
447	895	896	What are natural numbers?	What is a least natural number?	0
1518	3037	3038	Which pizzas are the most popularly ordered pizzas on Domino's menu?	How many calories does a Dominos pizza have?	0
3272	6542	6543	How do you start a bakery?	How can one start a bakery business?	1
3362	6722	6723	Should I learn python or Java first?	If I had to choose between learning Java and Python, what should I choose to learn first?	1

Table 8.1: Example question pairs from the Quora [4] dataset.

Example questions pairs from the dataset are shown in table 8.1. An important observation is that non-duplicate question pairs differ only slightly and are non-duplicates in a finer semantic context. As a result, we not only consider the pairs annotated duplicate or not but also random pairs from the dataset for evaluating the similarity metrics. This approach allows us to compare completely different questions from similar but semantically distinct question pairs.

For every similarity measure, we evaluate over the dataset collecting the similarity between duplicate, non-duplicate and random pairs giving us three frequency histograms. These frequencies are organised into buckets of size $k = 0.05$ creating 200 buckets over the similarity range 0 to 1. We also normalise the frequencies as the data size for each category is not equal. We organised the evaluation based on what technique is used: average, token and tree-structure based.

8.1.1 Average Based

Average based similarity measures rely on creating a single vector representation for the entire sentence. Hence, they can lose information as tokens get averaged over longer sentences. The only difference between bag-of-words from just averaging was the normalisation function $g(i)$ that substituted the named entity variables with fixed vectors. Since variables have placeholders, it was important not to add bias towards similar entities within a category such as locations. The bag-of-words similarity is, thus, used for matching user questions with the query part of a rule to start the answer computation in the proof engine.

Figure 8.1 compares the two average based similarity metrics. There isn't any notable difference between the two methods besides some random fluctuation created by the random sampling. This result is expected as most of the questions do not contain named entities to normalise; therefore, for the majority of the data bag-of-words acts like the document average over all tokens in the questions.

The similarity of duplicate pairs is higher and closer clustered together indicating the metric is capturing semantic differences by a small margin. The duplicates standard deviation is ≈ 0.11 whereas non duplicates are spread more with a standard deviation of

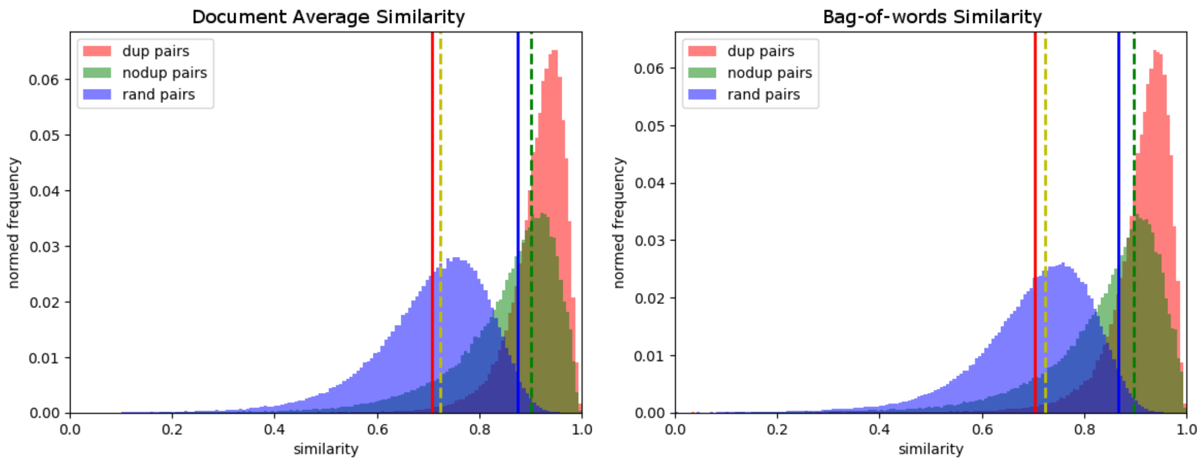


Figure 8.1: Comparison of average based similarity measures used by Kevin.

≈ 0.17. The random pair similarities lean further towards left with an average of ≈ 0.72 indicated by the solid red line. The results overlap the most around 0.80 to 0.82 region in which most permutations of sentences give that similarity measure; thus, *the threshold for Kevin is set to 0.82 in the current implementation* when matching user queries in the evaluation module (section 7.3.4).

8.1.2 Token Based

Token based similarity measures look at each token and find how much of the information represented by the token is *embedded* in the other sentence. They are very greedy similarity measures to find any possible sub tree structure that could match the given query. The main difference between maximal token and weighted (fuzzy) similarity measure is the slight consideration of the dependency structure of the sentence detailed in section 5.1.4. The fuzzy similarity measure is used for fuzzy unification which tries to find the *most likely* sub structure for a variable based on token similarity and the dependency structure (section 5.2.2).

We observe a very skewed histogram for both maximal token and fuzzy similarity measure in figure 8.2. In the case of maximal token similarity, there is a spike in the 1.0 similarity bucket for both duplicates and non-duplicate pairs. This feature is caused by considering any permutation to be equal to the given sentence, thus yielding a value of 1. Even if the question pairs differ by only 1 word, such as when rephrased, they will be in the upper ≥ 0.995 region to be placed in the highest bucket. Since those slight permutations have very high similarities, the buckets in the region of ≈ 0.98 demonstrates a dip to almost no examples before the spike at the end. A good feature about the token similarity is the almost perfect bell shape of random pairs with the average and median overlapping. Since it checks for token embeddings, random questions are less likely to

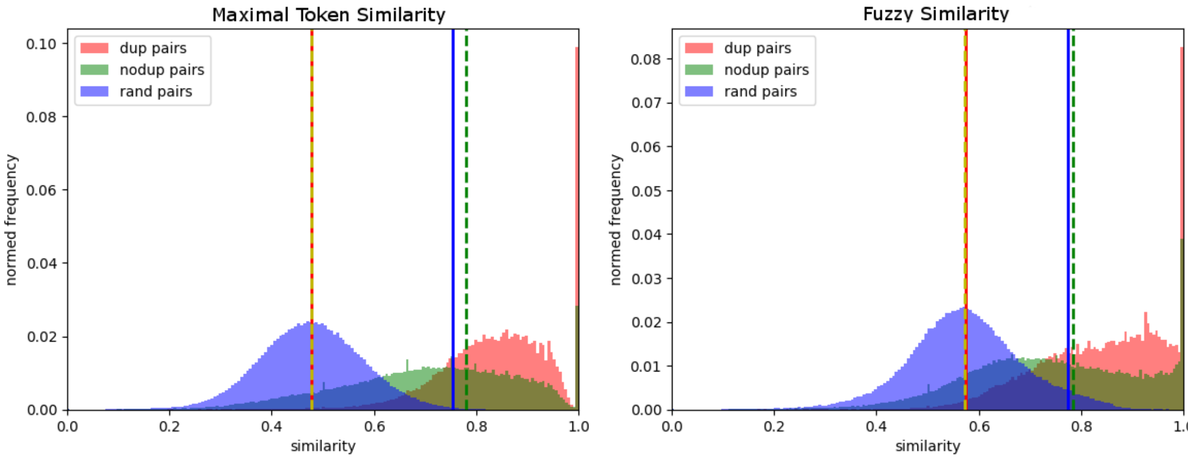


Figure 8.2: Comparison of token based similarity measures.

contain the same tokens besides the most frequent ones such as `what` or `who`. Therefore, a bell shape appears centred around the average of ≈ 0.47 far left of duplicate and non-duplicate pairs average at ≈ 0.78.

For the case of fuzzy similarity, the most distinctive difference is the lack of drop in buckets leading up to the spike in the end. Unlike the token similarity which accumulated more examples in the last bucket with a normed frequency of ≈ 0.097, the fuzzy similarity has less question pairs with ≈ 0.078. This change is the result of fuzzy similarity taking into account the dependency structure and being slightly more robust to permutations of sentences. Hence, the buckets in the region of 0.9 are filled with examples that contain similar tokens but difference structure. It is also notable that more duplicate pairs score in that region that non-duplicates, most likely due to duplicates having similar sub structures then non-duplicates.

In both similarity measures, the non-duplicates are smeared across a large similarity region ranging from 0.4 to 1. The average for both is ≈ 0.77 above the random pairs but with a much higher standard deviation. In either case, the point at which the random pairs are most separated from the duplicate and random pairs is centred just below the average of non-duplicate pairs, around ≈ 0.70; therefore, *the fuzzy unification threshold in Kevin is set to 0.70 to minimise the false positives* from triggering the wrong rules.

8.1.3 Tree Structure Based

Tree structure based similarity measure take into account the dependency structure as their main metric. Therefore, slight changes in the tree structure can significantly lower the similarity of a pair. The tree edit similarity is based on the tree edit distance [72] detailed in section 5.1.5. The edit similarity purely considers the edit distance; whereas, the hybrid is a weighted sum of both edit and fuzzy similarities for some weight coefficient

k. Since the computation time of tree edit distance is relatively much higher than any other vector based similarity, neither edit nor hybrid similarity is used within Kevin, although it is implemented.

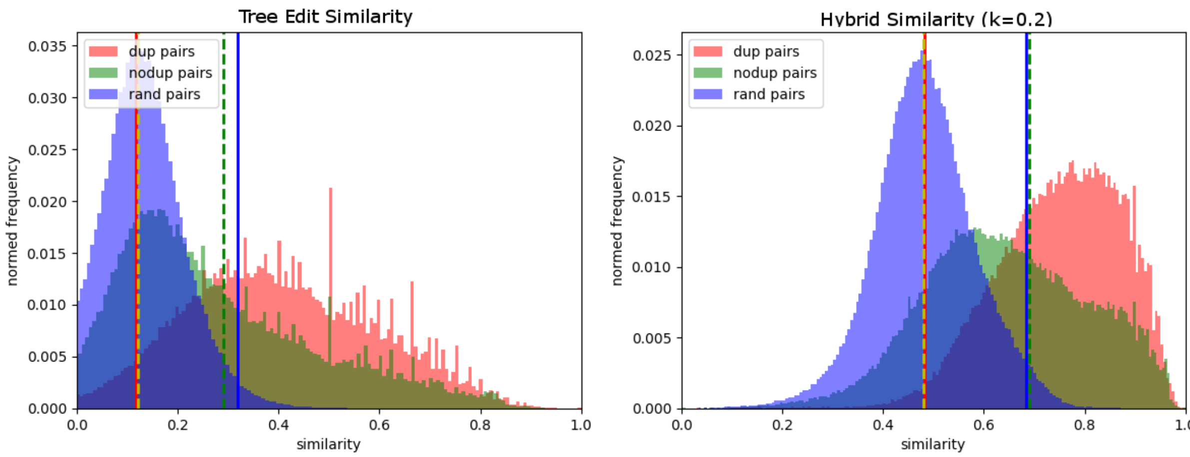


Figure 8.3: Comparison of tree structure based similarity measures.

Figure 8.3 captures the histograms for the two similarity metrics. The most noticeable difference is the left skewed histogram for the tree edit similarity. This result is expected as this measure is extremely susceptible to changes in structure and it is almost never the case that two questions are asked the same way for duplicate and non-duplicate pairs. It is also very unlikely that a random question pair will have the same structure; therefore, the graph bulges at the lower end of the similarity measure. The random pairs average at ≈ 0.1 again with a bell shape from the random sampling and the duplicate, non-duplicate pairs average at ≈ 0.32. The maximum overlap is achieved at the median of duplicate, non-duplicate pairs marked with dashed green line at ≈ 0.29. As it currently stands, the tree edit similarity is not a good measure for calculating semantic similarity between questions pairs as even the duplicate pairs are skewed below 0.5 with high overlap. A final feature is the spike observed at 0.5; this spike is the result of just addition or removal of tokens. For example, `Hi` and `Hi.` only differ by a full stop and the edit distance will be 1, the insert cost of the full stop. Therefore the edit similarity will be $\alpha(\ldots) = \frac{1}{2} = 0.5$ producing the spike for many other structurally similar examples in the dataset.

Hybrid similarity, on the other hand, gives the best distinction between duplicate and random pairs. At a weight of $k = 0.2$, the metric takes into account only 20 percent of the tree structure and 80 percent the fuzzy similarity. Therefore, we push the fuzzy similarity towards the shape of edit similarity just about to get a nice centred random pairs with an average of ≈ 0.5 and duplicate pairs at ≈ 0.8. Despite the best distinction achieved using hybrid similarity for this dataset, the heavy computation deters us from using it within Kevin.

8.2 Sentence Extraction

In this section, we evaluate the different similarity measures in extracting the correct sentence from a context given a query. For this purpose, we take the dev dataset from SQuAD [71] and use it to select single sentences that contain the answer.

$$\exists i, j \text{ s.t. } f(C, Q) = C_s[i : j] = A \tag{8.1}$$

We consider a success using equation 8.1 which indicates that a slice from the selected sentence C_s is equal to the expected answer. This setup might have bias when the answer is contained within multiple sentences in the context such as named entities mentioned multiple times. As a result, it is a comparative measure for different similarity measure and not necessarily the accuracy of extracting the correct sentence. In other words, the evaluation is of similarity measures and not of sentence extraction.

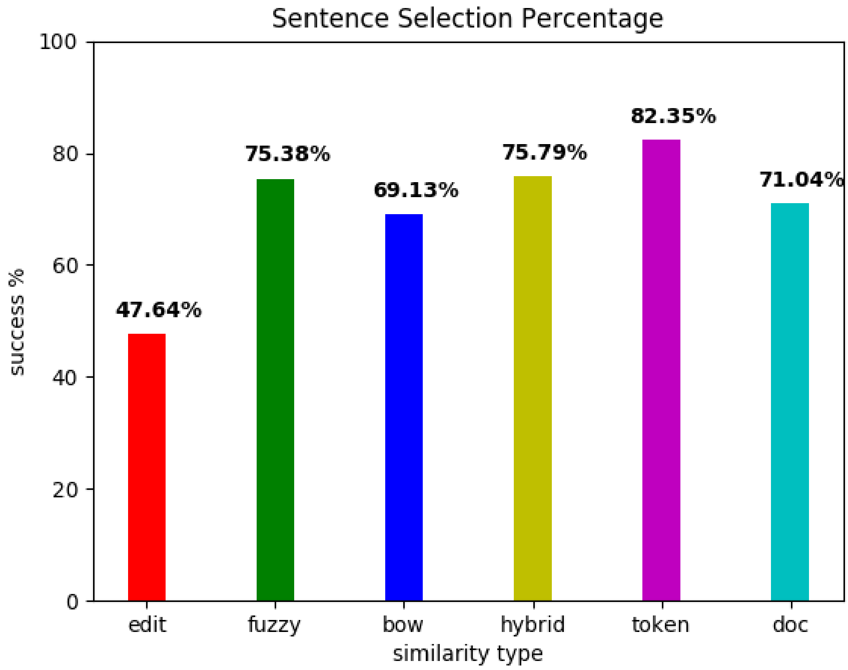


Figure 8.4: Comparison of different similarity measures for sentence extraction.

In figure 8.4 we can see the success rate of different sentence similarity measures. The best with 82.35% is the maximal token similarity due to its aggressive nature picking out sentences that contain the most information regardless of structure. In a noisy dataset, we expected for such a similarity measure to perform the best across more conservative measures. On the lower end is the tree edit similarity with 47.64%; as hinted from the question pair evaluation, it is unlikely that the query will resemble the sentence structure of the answer. Therefore, we reckon edit similarity requires preprocessing to change the query into a answer form such as converting `Who is X` to `X is`, but proves difficult as there are many cases to cover.

The fuzzy and hybrid similarity measures achieve success rates of ≈ 75% below maximal token similarity. The hybrid similarity adds the extra structural information to be ever so slight better than fuzzy similarity alone. However, the runtime cost of hybrid similarity is so high relative to simple vector operations, it is not worth the slight improvement. Both document average and bag-of-words similarity measures produce results at ≈ 70% below token based measures. This situation is expected as they lose information when collapsing the token vectors into a single sentence vector.

Since fuzzy similarity is used as the main measure for handling noisy input, we can say that Kevin's ability to extract correct sentences from contexts is ≈ 75% depending on the context provided. This accuracy is often enough to handle noisy user input and external text such as Wikipedia to a degree that Kevin is considered adequate. In more simpler scenarios such as every day tasks of setting reminders and asking weather information, this measure is enough to match correct rules and perform the required actions. Exploring more robust sentence selection and cross sentence information retrieval is for future work.

8.2.1 Performance

Mentioned along the previous evaluation sections, all similarity measure have different running times. The time it takes for Kevin to produce a response is as important as computing the correct answer. Therefore, we must balance the benefits of a similarity measure with the time cost it comes with. We will look at the running times of all similarity measures on the dev dataset used to find sentence extraction rates. The timing is the cumulative time in seconds to process the dev dataset containing around 7000 question context pairs.

Similarity	≈ Runtime (s)
doc	16.64
bow	17.41
token	34.16
fuzzy	41.64
edit	102.02
hybrid	130.96

Table 8.2: Comparison of runtime performance of different similarity measures.

The results are summarised in table 8.2 for each similarity measure. The average based document and bag-of-words perform the fastest as they are composed of just vector operations. The slight increase in time for bag-of-words is the named entity vector substitution function which adds an overhead of around a second. The runtime jumps to range of 35 seconds for token based measures as they consider every token individually. The runtime in token similarities increase proportionally with the length of either sentence yielding a time complexity of $O(nm)$ in which the n and m account for the number of tokens in the sentences being compared. The fuzzy similarity being recursive in nature takes longer than just checking maximum similarity for every token. Although the total runtime for entire dataset seem long, checking individual sentence pairs is still considerably slow. Therefore, for the results in the given runtime fuzzy similarity provides, it is used in Kevin as the main similarity measure.

For any measure containing the tree edit distance, the runtime is relatively much higher due to the complexity of the edit distance. The edit similarity suffers a significant performance hit as a result. Since hybrid similarity computes both edit and fuzzy, the runtime sums to the worst among the other measures. A runtime of 130 seconds causes considerable delay for computing an answer when scanning knowledge bases and rules; therefore, despite having a slight advantage in sentence extraction, it is not used within Kevin for its runtime cost.

8.3 User Interactions

In this section we look at possible interactions within the span of the current implementation. The example interactions touch on various components and usage cases discussed throughout the report, mainly in the logic framework chapter 4. In most cases, when Kevin doesn't know *how* to answer a user query it falls in to the following categories:

- **Lack of intents** often is the main reason queries fail. If the particular intent is not implemented then Kevin either fails or finds a closest match which might be wrong. For example, if the user asks for nearby restaurants then Kevin would not *know* how to answer since it doesn't contain any intents regarding restaurants or external functions that can find its location and list restaurants nearby. However, these situations are not a direct limitation of the framework used by Kevin but rather a matter of programmer time adding extra capabilities using intents. This situation is common practice in industry with the likes of Amazon Alexa [31] with new intents added every week following current global news and events.
- **Similarity measure false positives** can cause Kevin to pick up on rules other than the intended ones. This situation is a shortcoming of the fuzzy similarity measure used to determine what the user wants Kevin to do. In most cases, the similarity measure succeeds but the unification fails.
- **Unification fails** if the correct arguments for a rule cannot be extracted from the user query. This situation is common if the query is rephrased heavily with

structural changes on which the fuzzy unification relies on. Therefore, Kevin moves on the second best rule which might not be the intended one. This behaviour can be mitigated by restricting the evaluation process to a single rule check, though might miss out on correct rules due to failed similarity measures. For example, the user asks "What is the population of Berlin?" and Kevin will respond with the closest thing in its default context: the weather information at that location.

- **Sub context unknown queries** can look like Kevin failing to respond even though the query is redirected to another component such as an external search process. In these cases, there isn't any mechanism for Kevin to understand and mitigate the failure since almost all the responsibility of answering the query is delegated.

Since the overall interaction with Kevin is determined by so many different factors it is hard to evaluate individual components. Therefore, we will focus on cases in which Kevin does respond with an answer and the most prominent feature that generated the answer.

8.3.1 Sample Queries

From various interactions with people from the department of computing, we conducted manual tests in which users asked random queries within the default context. The default context means that the queries weren't routed to anywhere external and purely answered on the knowledge base available at the start. We asked 14 computing students ages ranging from 18 to 24 to interact with Kevin and collected common queries under various categories. We were able to only conduct user evaluation within the department due to time constraints of the project.

Most of the *useful* cases for a digital assistant come from intent parsing as seen in table 8.3. In fact, a majority of the queries were aligned with an external function to fetch some information such as traffic, restaurants, weather and everyday tasks. This indicates strongly that having a logic framework only aids little in answering such queries and the framework only contributes in evaluating the query by matching it to an intent. Thus, we believe from the point of a practical digital assistant having a logic based framework did not contribute as much as other components such as information retrieval.

A more robust intent parsing with many more external functions implemented would have covered most common scenarios in our user tests. However, those cases require the conversion of natural language into structured data that functions can use. This trend is common in many intent based digital assistants with platforms providing specifically that service such as API AI [63]. From a natural language understanding and reasoning perspective, the logic framework does yield potential use cases such as deriving conclusions

Query	Response	Type
hi	Hello there.	Scripting
How are you?	I'm fine and you?	Scripting
What time is it?	It is 13:57.	Intent parsing
What day is it today?	Today is Tuesday.	Intent parsing
What day is tomorrow?	Today is Tuesday.	Closest intent
Make me a sandwich	I don't seem to have sudo access.	Scripting
Is there a storm in London?	It is currently 14 degrees in London.	Closest intent
Is your name Kevin?	My name is Kevin.	Negation by failure
Do I need to do anything?	Your to-do list is empty.	Intent parsing
Remind me to take out the rubbish.	I've added that to your to-do list.	Intent parsing
Jack seems to be happy.	OK.	Fact generation
Is Jack happy?	Yes, Jack is happy.	Template deduction
Jack is now sad.	OK.	Constraint overrule
Is Jack happy?	No, Jack is not happy.	Negated deduction

Table 8.3: Sample interactions within the default context of Kevin.

using basic deduction. However, those cases are not as common as intents within the context of digital assistants.

8.3.2 Wikipedia Application

To consume external text and evaluate the information retrieval capabilities we implemented a basic Wikipedia application within Kevin. Given a query it will search for a *topic* and try to find the answer only in the summary section of the corresponding page. There are other smaller text comprehension apps that work on children's text examples; however, tackling the noisy data of Wikipedia articles is indicative of Kevin's sentence selection and extraction capabilities. We chose Wikipedia for its simple and easy to use API providing plain text summaries for every topic. News articles and other sources of information can be integrated in a similar fashion.

The application receives queries from Kevin and tries to determine the topic to search for. If a subsection is double quoted, it searches for that slice otherwise looks for the first named entity within the query if any. We present both the intermediate sentence selection result and the guided BiDAF (section 6.2) on the selected context sentence.

Table 8.4 curates some of the successful user queries that Kevin was able to answer within the Wikipedia application. In these cases, both the sentence selection and the guided BiDAF work successfully together to give an answer. It is common that if the sentence selection gets the correct sentence from the given Wikipedia summary, then the

Query	Answer (g-BiDAF extract in bold)
When was Samsung envisioned?	Samsung was founded by Lee Byung-chul in **1938** as a trading company.
Who founded Samsung?	Samsung was founded by **Lee Byung-chul** in 1938 as a trading company.
What is the capital city of Germany?	Germany's capital and largest metropolis is **Berlin**, while its largest conurbation is the Ruhr.
When was Science Museum London founded?	It was founded in **1857** and today is one of the city's major tourist attractions, attracting 3.3 million visitors annually.
When was Imperial College founded?	Imperial College London was granted Royal Charter in **1907**.
Who founded Imperial College?	Its founder, **Prince Albert**, envisioned an area comprising the Victoria and Albert Museum, Natural History Museum, Royal Albert Hall, and the Imperial Institute.
How many countries are in the European Union?	The European Union is a political and economic union of **28** member states that are located primarily in Europe.
How many moons does Jupiter have?	Jupiter has at least **67** moons, including the four large Galilean moons discovered by Galileo Galilei in 1610.
Why is "the sky" blue?	During daylight, the sky appears to be blue because **air scatters blue sunlight more than the sky scatters red**.

Table 8.4: Successful queries for Wikipedia application within Kevin.

BiDAF model can extract the correct answer. However, this collaboration is not always the case.

The cases in which either the sentence selection fail or the guided BiDAF model doesn't quite extract the correct answer are contained in table 8.5. Not all the examples are wrong; for example, the capital city of France is extracted correctly. The following question asking the population fails horribly and doesn't even get the correct sentence. Cases such as the question asking the author of Harry Potter succeed with over approximation by the neural network. It *plays safe* by extracting a larger slice than absolutely necessary including "British author" in this case.

Despite shortcomings of sentence selection and the neural network, with this application Kevin can cover a large variety of topics and harness the information available in its raw form. This type of information retrieval questions were very common; hence, they play a critical part in constructing digital assistants and with similarity measures Kevin is able to accommodate most of them.

As part of our evaluation we noticed that users preferred full sentence answers rather

Query	Answer (g-BiDAF extract in bold)
What is Wikipedia?	Wikipedia is a **free online** encyclopedia with the aim to allow anyone to edit articles.
What is the primary objective of Imperial College?	The university's emphasis is on **emerging technology** and its practical application.
Who wrote Harry Potter?	Harry Potter is a series of fantasy novels written by **British author J. K. Rowling**.
What is the colour of Calcium?	Calcium is a soft greyish-yellow alkaline earth **metal, fifth-most-abundant element** by mass in the Earth's crust.
Which films did Brad Pitt play in?	Pitt starred in the cult film **Fight Club** and the heist film Ocean's Eleven and its sequels, Ocean's Twelve and Ocean's Thirteen.
What is the capital city of France?	France is a unitary semi-presidential republic with the capital in **Paris**, the country's largest city and main cultural and commercial centre.
What is the population of France?	France is also a member of the Group of 7, North Atlantic Treaty Organization, Organisation for Economic Co-operation and Development, the World Trade Organization, and La Francophonie.

Table 8.5: Mixed queries for Wikipedia application within Kevin.

than the g-BiDAF extraction. Full sentence answers not only provide some extra information about the query but also create a richer experience overall. Rather than giving single answer such as dates and names, Kevin replies with full sentences which yield the necessary context of the answer. As a result, the default behaviour of the Wikipedia application is to return full sentences and bypass the upstream neural network.

8.4 Limitations

In this section we will detail some of the major limitations within Kevin. Although external components such as the natural language processing back end can create limitations, we are interested in those which arise from the framework Kevin uses. We will not consider lack of functionality due to missing implementation a limitation of the framework as Kevin has the capacity to expand and integrate with more external components.

8.4.1 Co-reference resolution

Since we opted to use Spacy as the only natural language processing library, we didn't have a robust co-reference resolution mechanism like provided by CoreNLP [75] (section 7.1). As a result, Kevin has very basic co-reference resolution support based on dependency structures. This situation hampers follow up queries as well as consuming external text limiting Kevin's capability to process pronoun chained information.

$$\begin{aligned}&\leftarrow \text{Who is Barack Obama?}\\&\leftarrow \text{Who is his wife?}\end{aligned} \tag{8.2}$$

An example in which the basic dependency co-reference resolution fails is given in queries 8.2. Since the structure of a possessive pronoun (PRP$) is different from the named entity in the previous query, the current system won't be able to resolve it. The dependency changes from `nsubj` as in subject to `poss` for the pronoun. For cases in which the pronoun appears in different structural context, Kevin fails to respond and shortcuts to replying "I'm not sure what you are referring to." as part of an error message.

$$\begin{aligned}&\leftarrow \text{Who is Barack Obama?}\\&\leftarrow \text{Does he have a wife?}\end{aligned} \tag{8.3}$$

A workaround would be to rephrase the second query as shown in queries 8.3 which will have matching dependencies and therefore be resolved. In this case Kevin will replace `he` with `Barack Obama` on the basis that they have the same dependency in the parse tree. Therefore, the overall co-reference resolution system available in Kevin is cumbersome and prone to error.

To achieve a smooth conversation with follow up queries, it is crucial to have a robust co-reference resolution. It might require the merger of both natural language processing libraries to work in the areas they excel in and provide better results. Currently, it is a limitation that cuts conversations short and discourages users from interacting further with Kevin.

8.4.2 Follow-up Queries

Following a weak co-reference resolution, we note that the logic framework does not provide a *natural* way of chaining rules together to provide a conversation. The framework acts on a single rule at a time which could have side affects such as setting context for the upcoming queries. However, it would require for almost every rule to have some side affect to glue a conversation together. These scenarios frequently come up when an intent requires more information than the user has currently given prompting a conversation such as ordering takeaway but not specifying when.

$$\begin{aligned} \text{You sure?} \wedge \text{:setcontext:yesno} &\leftarrow \text{Clear my to-do list.} \\ \text{Done.} \wedge \text{:cleartodo} &\leftarrow_{yesnocontext} \text{Yes.} \\ \text{Ok.} &\leftarrow_{yesnocontext} \text{No.} \end{aligned} \tag{8.4}$$

Rules described by equations 8.4 capture the cumbersome way of asking a simple yes or no question to the user. The context needs to be set and tracked properly in order for even a simple interaction to work. For more complicated cases, nested contexts as well as the rules that manage them become convoluted and not natural for someone to design. In this example, the `yesno` context allows the second and third rules to trigger completing the conversation. But context management is an overhead that the framework requires in order to even process scripted conversations.

$$\text{You sure?} \wedge \text{:redirect\_input} \leftarrow \text{Clear my to-do list.} \tag{8.5}$$

The conversation design is handled this way partly because the user can give a completely unexpected query such as asking the time when Kevin was expecting a yes or no. Such a context structure still allows for the user to exit the context and successfully answer another query. On the other hand, a query redirect as in rule 8.5 would not be able to process that correctly. The external function receiving the query is responsible of figuring out the correct answer; although it can reroute back to the evaluation module, the input is ultimately not considered by Kevin in the first hand. The current options for follow-up queries are summarised below:

1. Create **an application that consumes all queries** and bypass almost all of Kevin, including the logic framework. All Kevin will do is to delegate the query similar to the Wikipedia application (section 8.3.2). In this case, the application must maintain the internal state and what to expect. More importantly, the application has to handle unexpected input if any.

2. The programmer can be bold and force Kevin to **redirect the next input** to a single external function one by one. This approach is captured in rule 8.5 in which the next query will be redirected but unless specified the following queries will be processed by Kevin. These cases are easiest for single follow up questions such as asking for an address. However, since the query is redirected unexpected input must be handled.

3. The **logic framework contexts** are created and tracked in order to use rules that capture the conversation. This approach is the best option as it can handle

unexpected input naturally using the evaluation module and the knowledge base. For example, if the user asks the weather while ordering takeaway, Kevin will answer correctly and safely close the takeaway order context. However, it is tedious to create these conversations and might be susceptible to similarity and unification errors such another rule triggering instead of the expected one in a follow up question.

Using contexts carefully, a programmer can construct a graph based conversation in which nodes correspond to the current context and edges to queries that transition between them. However, we think such a structure is unwieldy and could benefit from a better design. As a result, it is *tedious* to create long conversations for intents such as ordering takeaway, booking restaurants and many others that require a continuous context aware query processing.

8.4.3 Single Sentence Selection

Another major limitation with consuming external text is that it is strictly limited to a single sentence. Any given context is reduced to one most likely sentence that contains the information and then processed upstream using the neural network. Therefore, if an information is contained across multiple sentences Kevin would not be able to understand. This situation does not account for run on sentences or sub clauses as they are considered a single sentence. We are interested in cases in which information is spread out across multiple sentences.

When did the argument resolve?
An argument regarding the height between China and Nepal lasted five years from **(1) 2005 to 2010**. China argued it should be measured by its rock height of 8,844 m, but Nepal said it should be measured by its snow height of 8,848 m. In **(2) 2010**, an agreement was finally reached by both sides that the height of Everest is 8,848 m, and Nepal recognises China's claim that the rock height of Everest is 8,844 m.

Table 8.6: Cross sentence information retrieval limitation example.

This situation is demonstrated in table 8.6 in which a context paragraph from Wikipedia's Mount Everest page[1] summary is given. The correct answer **(2)** is 2010 as that is when an agreement was reached. However, since the query tends towards a sentence with an argument we obtain the answer marked **(1)**. This answer is identical to the reference implementation of BiDAF model [9] described in section 3.4.4. So despite the original model able to process the entire context, when the query mischievously hints towards another

[1] https://en.wikipedia.org/wiki/Mount_Everest accessed 2017-06-02

sentence that contains similar keywords such queries fail. In this case, the closest slice of the first sentence to a date was "2005 to 2010" and hence the answer. The answer is not completely wrong as the argument is resolved in that time range although it can also mean the resolution took 5 years. Either way, the sentence in which the actual answer is not captured correctly.

There are many more examples within a noisy source such as Wikipedia that require multi sentence information retrieval. Currently, Kevin cannot handle any of them given the sentence selection only works by extracting a single sentence. A possible solution to get a window of sentences around the most similar sentence to be processed as an over approximation. The bigger the window size the safer since if Kevin returned the entire context every time for the user, the answer is very likely to be there but the user would have to figure it out. Additionally, the larger the window size, more noise has to incorporated as we might get sentences that do not relate to the answer we are looking for.

Extracting information across sentences is an active research area with the SQuAD [71] dataset. We believe it would add an extra edge in information retrieval from noisy sources and provide an even richer user experience in the context of digital assistants.

Chapter 9

Conclusion

Building a digital assistant requires many components to work together in order to compute an answer. Certainly with Kevin, we built a multi-stage processing pipeline that was inspired by logic programming with a lot of tweaks such as external functions and components. At its core, Kevin used a novel logic framework that worked with natural language primitives to process and respond to user input. By constructing the framework on vector representations of words, Kevin was able to *understand* similar queries and handle noisy input using fuzzy unification and sentence similarities.

User evaluations often demonstrated that most queries did not require a logic framework to perform higher level reasoning. In most cases, Kevin was bypassing the framework for other more frequently used components such as external functions and their corresponding intents. Therefore, *having a logic framework did not yield outstanding results* in spite of some reasoning constructs such as negation by failure. There were parts of the framework such as similarity measures used to find a matching rule, which provided good results and tools in face of noisy text sources.

In conclusion, although there are sections of the logic framework beneficial to the theme of digital assistants, overall it is shadowed by the common use cases such as intent parsing and areas in which neural networks already excel in. This result is bound to the context of digital assistants; however, in other cases such as story comprehension, the benefits of having a high level reasoning framework might prove more beneficial.

9.1 Summary of Work

Given the scale of the project and that it was implemented from scratch, there were various milestones that in general follow the structure of this report. In this section, we reflect on the key points of the project summarising the work undertaken while creating Kevin:

1. We started off by looking at existing open source digital assistants and if possible how we can integrate some logical reasoning into their frameworks. Lack of examples and bespoke processing pipelines pushed us towards creating our own framework that could handle natural language in a more straightforward manner.

2. Since the focus of the project was interacting in natural language rather than processing it, we searched for existing libraries (section 7.1) and picked one that most suited us. The library acted as a black box from our perspective consuming plain text and giving back annotated parse trees.

3. We implemented a basic framework that was inspired by logic programming and deviated from standard semantics in most cases in order to fit into a digital assistant theme. The implementation and the pipeline (section 7.3) consumed most of the project time as it was a unique logic framework with parse tree as its primitives.

4. To accommodate noisy input and handle more queries with less rules, we considered vector based similarity measures harnessing the power of existing, pre-trained neural networks. Using GloVe [3] vectors, we constructed multiple sentence similarity measures (section 5.1) and integrated into our logic framework particularly for rule selection as the first use case.

5. Using the similarity measures we then looked at how to augment our basic tree based unification to create *fuzzy* unification (section 5.2.2). Fuzzy unification was a key point for Kevin to understand noisy input and extract arguments putting the most common use case of intent parsing into perspective. Before fuzzy unification, Kevin's intent parsing capabilities were very limited; it required either no variables or identical query structures to work.

6. While investigating sentence similarities, an interesting use case became apparent and opened a new gateway into information retrieval. Using the similarity measures, Kevin started to find answers to free form queries in large noisy text sources such as Wikipedia. This sentence selection (section 6.1) lead to more investigation into neural networks to benefit the query answering capabilities of Kevin, particularly using a custom neural network guided BiDAF (section 6.2).

7. We then structured the logic framework as a *gateway* to all underlying functionality merging standard reasoning with external components such as the neural networks. Various use cases were described in the logic framework chapter 4. As mentioned, the logic framework started to act more and more as a query router than a reasoning platform for the frequent use cases.

8. Finally, we evaluated Kevin's various components against other datasets to adjust our thresholds and get some user feedback.

Across the entire project, building the logic framework consumed the most time as it was not clear how to integrate parse trees as a primitive. New semantics and a proof mechanism had to be designed in order to make it work. Overall, we got most value out of the sentence similarity measures based on vector representations as they allowed Kevin to handle noisy input in a much more robust manner.

9.2 Future Work

Although there many points to expand and explore further, we identified two main points that would benefit Kevin the most: fusing the logic with the neural networks and pushing towards multiple sentence, context aware comprehension. We believe these two areas are the key points for a more uniform, robust framework to *understand* natural language. There are other components such as managing follow-up queries; however, most improvements in those components would come from redesign rather than theory.

9.2.1 Neural Logic Amalgamation

A big theme in Kevin is the logic framework with neural network components running together. However, they are still separated in an oblivious manner. The logic framework doesn't know about neural back ends and vice versa. Therefore, there is no real unison both functionally and semantically to augment the processing of natural language. The root cause for this situation is the complexity of either achieving neural network comprehension results using logic or pushing logic constructs into neural networks. Currently, they are distinct components that perform what they do best, logic does reasoning and neural networks handle noisy text query answering. We believe there are two key ways to proceed forwards:

1. Let the logic framework handle noisy input as good as neural networks so that higher level reasoning can be performed on human digestible text sources. The current architecture works on templates and fails when dependency structures change. Either way, a logic framework by construction is structured and therefore might be ill suited for ambiguous noisy context such as natural language.

2. More promising option is the task of pushing higher level reasoning into neural networks and reduce the complexity of a structured logic framework. This approach would require a neural network to process binary relations and derive conclusions, tasks that are much more complex than query answering over given context. With ever more sophisticated neural network architectures and increasing computing power, we believe it would be an area to explore further.

Kevin provides a technical way of combining logic and neural networks. Future research would focus on merging ideas into one another rather than a technical implementation using both.

9.2.2 Multi Sentence Comprehension

Despite the uses of sentence similarity measures for extracting information, they were limited to a single sentence as described in section 8.4.3. Most context written for human consumption require multiple sentence comprehension as the information required could be spread across the content. The current limitation of single sentence selection or its window based versions do not really capture meaning across sentences. This scenario also presents itself when handling follow up questions that might contain required information from queries in the past such as telling the location of the user and then remembering that information for asking what the weather is like.

We think this area could benefit from research to better process larger contexts and more complicated queries within the context of digital assistants. One options could be to integrate context into the networks such as memory networks [77] that store vector representations of context. It might be transferable to answering follow-up questions or answering questions based on past information.

Appendix A

NLP

This appendix contains the part-of-speech and dependency tags used by the natural language processing (NLP) libraries during parsing. They are used in training the parsers and the output tags may vary depending on the corpus trained on. The tags referenced throughout the report are from the output of Spacy (section 7.1.1) unless explicitly stated otherwise.

A.1 Part-of-speech Tags (POS)

This section contains the part-of-speech tags used by the Spacy library. They are referenced from the Penn Treebank [78]. Part-of-speech tags identify the type of the word in a sentence such as nouns (NN) and adjectives (JJ). The parse trees contain this annotation for every token.

A.2 Dependency Tags (DEP)

This section contains the dependency tags used throughout the report. They are based on the ClearNLP project [79]. The tags are from the output of Spacy library unless stated otherwise. The dependency tags describe the relationship between a token and its parent such as subject of a verb.

POS Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table A.1: Summary of part-of-speech tags.

Dependency Tag	Description
acomp	adjectival complement
advcl	adverbial clause modifier
advmod	adverbial modifier
agent	agent
amod	adjectival modifier
appos	appositional modifier
attr	attribute
aux	auxiliary
auxpass	passive auxiliary
cc	coordinating conjunction
ccomp	clausal complement
complm	complementizer
conj	conjunct
csubj	clausal subject
csubjpass	clausal passive subject
dep	unclassified dependent
det	determiner
dobj	direct object
expl	expletive
hmod	modifier in hyphenation
hyph	hyphen
infmod	infinitival modifier
intj	interjection
iobj	indirect object
mark	maker
meta	meta modifier
neg	negation modifier
nmod	modifier of nominal
nn	noun compound modifier
npadvmod	noun phrase as adverbial modifier
nsubj	nominal subject
nsubjpass	nominal passive subject
num	numeric modifier
number	number compound modifier
oprd	object predicate
parataxis	parenthetical modifier
partmod	participial modifier
pcomp	complement of a preposition
pobj	object of a preposition
poss	possession modifier
possessive	possessive modifier
preconj	pre-correlative conjunction
prep	prepositional modifier
prt	particle
punct	punctuation
quantmod	quantifier phrase modifier
root	root
xcomp	open clausal complement

Table A.2: Summary of dependency tags.

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] 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 preprint arXiv:1603.04467*, 2016.

[3] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

[4] K. C. Shankar Iyer, Nikhil Dandekar, "First quora dataset release: Question pairs." https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs, 2017.

[5] C. Pozna, P. Foldesi, and J. Kovacs, "The personal assistant application, problem definition," (Piscataway, NJ, USA), pp. 851 – 6, 2013//. personal assistant application design;daily activity list planning;costumer assistance;human operator;optimum succession planning;objective information;subjective information;problem definition;mathematical structure;.

[6] L. Friedman, "Hands-on with ios 6: Siri [personal assistant app for iphones and ipad]," *Macworld*, vol. 29, no. 12, pp. 50 –, 2012/12/. iOS 6 operating system;personal assistant app;Siri;iPhones;iPad;virtual assistant;.

[7] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, (Stroudsburg, PA, USA), pp. 63–70, Association for Computational Linguistics, 2002.

[8] R. Kowalski and M. Sergot, *A Logic-Based Calculus of Events*, pp. 23–55. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989.

[9] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *arXiv preprint arXiv:1611.01603*, 2016.

[10] M. L. Mauldin, "Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition," in *AAAI*, vol. 94, pp. 16–21, 1994.

[11] H. Loebner, "Home page of the loebner prize-the first turing test," *Online unter http://www. loebner. net/Prizef/loebner-prize. html*, 2003.

[12] A. P. Saygin, I. Cicekli, and V. Akman, "Turing test: 50 years later," in *The Turing Test*, pp. 23–78, Springer, 2003.

[13] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[14] J. Weizenbaum, "Computer power and human reason: From judgment to calculation.," 1976.

[15] A. ALICE, "Foundation," *Alicebot and AIML Documentation*, 2003.

[16] A. Saenz, "Cleverbot chat engine is learning from the internet to talk like a human," *Singularity Hub*, 2010.

[17] A. Lee, D. Sha, and C. Monteil-Anatra, "Artificial intelligence in games," *Korea Information Processing Society*, vol. 9, no. 3, pp. 69–76, 2002.

[18] M. Prasad, E. Scornavacca, and H. Lehmann, "Using wireless personal digital assistants in a restaurant: Impact and perceived benefits," (Sydney, NSW, Australia), pp. 69 – 74, 2005. Battery life;Ease-of-use;Hospitality industry;Mobile Technology;Perceived benefits;System crashes;Unreliable transmissions;Wireless personal digital assistants;.

[19] N. Maibaum, I. Sedov, and C. Cap, "A citizen digital assistant for e-government," (Berlin, Germany), pp. 284 – 7, 2002//. citizen digital assistant;e-government;mobile user device;PDA;public authorities;trusted interactions;secure interactions;digital ID cards;citizen-controlled communication;digital documents;digital identities;digital credentials;.

[20] M. Gordon and C. Breazeal, "Designing a virtual assistant for in-car child entertainment," (Boston, MA, United states), pp. 359 – 362, 2015.

[21] K. O'Shea, "Natural language scripting within conversational agent design," *Applied Intelligence. The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, vol. 40, no. 1, pp. 189 – 97, 2014/01/. natural language scripting;conversational agent design;semantic-based conversational agent;SCA;structural pattern;semantic content;natural language sentences;semantic sentence similarity measure;pattern-scripted languages;.

[22] M. Freed, J. G. Carbonell, G. J. Gordon, J. Hayes, B. A. Myers, D. P. Siewiorek, S. F. Smith, A. Steinfeld, and A. Tomasic, "Radar: A personal assistant that learns to reduce email overload.," in *AAAI*, pp. 1287–1293, 2008.

[23] P. Modi, M. Veloso, S. Smith, and J. Oh, "Cmradar: a personal assistant agent for calendar management," (Berlin, Germany), pp. 169 – 81, 2004//. CMRadar;personal assistant agent;calendar management;routine everyday task automation;email exchange;multiagent negotiation;schedule optimization;.

[24] O. Devisch, "Should planners start playing computer games? arguments from simcity and second life," *Planning Theory & Practice*, vol. 9, no. 2, pp. 209–226, 2008.

[25] A. Augello, G. Pilato, and S. Gaglio, "A conversational agent to support decisions in simcity like games," (Piscataway, NJ, USA), pp. 367 – 72, 2009//. SimCity like games;computational intelligence;intelligent conversational agent;decision support system;game-based model;virtual town;natural language interface;.

[26] T. Lucas, "Authenticate users securely through chat with converse.ai and twilio," 2017-1-17. https://blog.converse.ai/authenticate-users-securely-through-chat-with-converse-ai-and-twilio-dbd6752990e2.

[27] H. Cherchi, "Build your first bot with botbuilder," 2017-1-17. https://blog.recast.ai/build-first-bot-botbuilder/.

[28] Chatfuel, "Native messenger payments on chatfuel," 2017-1-17. http://blog.chatfuel.com/native-messenger-payments-on-chatfuel/.

[29] F. Holzmann, M. Bellino, R. Siegwart, and H. Bubb, "Integration of driver assistant systems as a virtual driver," (Meguro-Ku, Tokyo, Japan), pp. 19 – 24, 2006. Command levels;Driver assistant systems;Virtual drivers;.

[30] K. O'Shea, Z. Bandar, and K. Crockett, "A semantic-based conversational agent framework," (Piscataway, NJ, USA), pp. 8 pp. –, 2009//. semantic-based conversational agent framework;user input;natural language sentences;script interpretation;structural sentence patterns;.

[31] A. Weber and A. Alexa, "Amazon echo: The best user guide to master amazon echo fast," 2016.

[32] A. Laukaitis, O. Vasilecas, and V. Gediminas, "Natural language based heavy personal assistant architecture for information retrieval and presentation," vol. pt.2, (Berlin, Germany), pp. 54 – 63, 2007//. natural language;heavy personal assistant architecture;information retrieval;information presentation;information extraction;enterprise environment;information systems development;enterprise semantic Web;.

[33] P. Garrido, F. J. Martinez, and C. Guetl, “Adding semantic web knowledge to intelligent personal assistant agents,” vol. 687, (Shanghai, China), 2010. Current ability;Modular knowledge;Personal assistant agents;Personal assistants;Semantic repository;Specific tasks;Subject-centric computing;.

[34] M. Mehta and A. Corradini, “Developing a conversational agent using ontologies,” vol. Pt.3, (Berlin, Germany), pp. 154 – 64, 2007//. conversational agent;ontologies;3D graphical character;domain independent component;domain specific component;Google directory categorization;.

[35] K.-J. Chen and J.-P. Barthes, “Memopa: Intelligent personal assistant agents with a case memory mechanism,” (Berlin, Germany), pp. 1387 – 1397, 2007/08/. intelligent personal assistant agents;case memory mechanism;software agent;case-based reasoning;.

[36] K.-J. Chen and J.-P. A. Barthes, “Giving personal assistant agents a case-based memory,” *International Journal of Cognitive Informatics and Natural Intelligence*, vol. 4, no. 1, pp. 45 – 64, 2010. Agent model;Cognitive agents;Design and build;Generic models;Memory models;Personal assistant agents;Personal assistants;.

[37] C. K. Riesbeck and R. C. Schank, *Inside case-based reasoning.* Psychology Press, 2013.

[38] D. Poole, “A logical framework for default reasoning,” *Artificial intelligence*, vol. 36, no. 1, pp. 27–47, 1988.

[39] U. Zernik and A. Brown, “Default reasoning in natural language processing,” (Budapest, Hungary), pp. 801 – 5, 1989//. nonmonotonic logic;default reasoning;nonmonotonic inferences;natural language processing;truth maintenance;logic parser;Non-Monotonic Grammar;NMG;first-order logic;text analysis;constraint propagation;.

[40] K. Agreda, E. Fabito, L. Prado, M. Tebelin, and B. Comendador, “Attorney 209: A virtual assistant adviser for family-based cases,” *Journal of Automation and Control Engineering*, vol. 1, no. 3, pp. 198 – 201, Sept. 2013. combinatorial explosion;dynamic programming;ambiguous grammar;natural language processing;chart parsing algorithm;legal separation;child custody;legal guidance;conversational chatbot;family-based cases;virtual assistant adviser;ATTORNEY 209;.

[41] M. Tomita, “An efficient augmented-context-free parsing algorithm,” *Computational linguistics*, vol. 13, no. 1-2, pp. 31–46, 1987.

[42] K. Myers, P. Berry, J. Blythe, K. Conley, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe, “An intelligent personal assistant for task and

time management," *AI Magazine*, vol. 28, no. 2, pp. 47 – 61, Summer 2007. intelligent personal assistant;task management;time management;knowledge worker;time commitment management;human problem-solving skills;artificial intelligence;Belief-Desire-Intention agent system;human needs;.

[43] J. R. Bellegarda, "Large-scale personal assistant technology deployment: The siri experience," (Lyon, France), pp. 2029 – 2033, 2013. Commercial deployment;Context-awareness;Dialog systems;Natural language interaction;Personal assistants;Software integration;Spoken languages;Technology deployment;.

[44] P. Milhorat, S. Schlogl, G. Chollet, J. Boudy, A. Esposito, and G. Pelosi, "Building the next generation of personal digital assistants," (Sousse, Tunisia), pp. 458 – 463, 2014. Assistive technology;Dialog design;Digital assistants;Language understanding;Natural language understanding;Personal assistants;Research and development;Spoken dialog systems;.

[45] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," vol. 2015-January, (Istanbul, Turkey), pp. 223 – 238, 2015. Data centers;Emerging workloads;Implications for futures;Natural languages;Personal assistants;Server architecture;Total-cost-of-ownership analysis;Web service applications;.

[46] S. F. Nagata, H. Van Oostendorp, and M. A. Neerincx, "Interaction design concepts for a mobile personal assistant," vol. 65, (Amsterdam, Netherlands), pp. Oc–Technologies; Atos Origin Technical Automation; Glacimonto; Macromedia; SIGCHI.NL –, 2004. Context- awareness;Customer support;Efficient user interaction;Hand held device;Intelligent interface;Interaction design;Interactive display;Mobile web;On-line service;Personal assistants;Personalizations;Scenario-based design;Usage models;User acceptance;.

[47] J. M. Carroll, *Making use: scenario-based design of human-computer interactions*. MIT press, 2000.

[48] M. Shanahan, *Embodiment and the inner life: Cognition and Consciousness in the Space of Possible Minds*. Oxford University Press, USA, 2010.

[49] E. Carrasco, G. Epelde, A. Moreno, A. Ortiz, I. Garcia, C. Buiza, E. Urdaneta, A. Etxaniz, M. Gonzalez, and A. Arruti, "Natural interaction between avatars and persons with alzheimer's disease," (Berlin, Germany), pp. 38 – 45, 2008//. natural interaction;avatars;Alzheimer's disease;human computer interaction;cognitive impairments;virtual character;virtual personal assistant;television remote control;return channel;functional prototype;.

[50] B. Xu and Y. Yu, "A personalized assistant in 3d virtual shopping environment," vol. vol.2, (Los Alamitos, CA, USA), pp. 266 – 9, 2010//. personalized assistant;3D virtual shopping environment;virtual reality;data mining technique;e-commerce;personalized recommendation system;3D graphical representation;human like shopping experience;personalized avatars;collaborative interaction;.

[51] A. Easwara Moorthy and K.-P. L. Vu, "Voice activated personal assistant: Acceptability of use in the public space," vol. 8522 LNCS, (Heraklion, Crete, Greece), pp. 324 – 334, 2014. Amazon mechanical turks;Common features;Information privacy;Personal assistants;Public domains;Social acceptability;Usage patterns;Voice interfaces;.

[52] J. Payne, A. Szymkowiak, P. Robertson, and G. Johnson, "Gendering the machine: Preferred virtual assistant gender and realism in self-service," (Berlin, Germany), pp. 106 – 15, 2013//. virtual assistant;virtual worlds;virtual agent visual characteristics;education-based agents;gaming avatars;online help assistants;SSCO technology;self-service checkouts technology;gender preference;user gender;similarity-attraction theory;social role theory;.

[53] Z. Wang, N. Cheng, Y. Fan, J. Liu, and C. Zhu, "Construction of virtual assistant based on basic emotions theory," (Berlin, Germany), pp. 574 – 81, 2005//. virtual assistant;basic emotions theory;psychology model;real-time facial expression;voice recognition;.

[54] Y. Arafa and A. Mamdani, "Virtual personal service assistants: towards real-time characters with artificial hearts," (New York, NY, USA), pp. 9 – 12, 2000//. virtual personal service assistants;real-time characters;new generation user interfaces;human factors;affective dimension;autonomous agent behaviour;emotion;artificial heart computational models;multi-modal expression;entertainment;multi-agent systems;personal service assistant interface characters;agent communication language;.

[55] M. Steedman and J. Baldridge, "Combinatory categorial grammar," *Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell*, 2011.

[56] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, "Abstract meaning representation for sembanking," 2013.

[57] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, "Abstract meaning representation (amr) 1.0 specification," in *Parsing on Freebase from Question-Answer Pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle: ACL*, pp. 1533–1544, 2012.

[58] A. Mitra and C. Baral, "Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning.," in *AAAI*, pp. 2779–2785, 2016.

[59] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[60] W. Wobcke, A. Nguyen, V. Ho, and A. Krzywicki, "The smart personal assistant: An overview," vol. SS-07-04, (Stanford, CA, United states), pp. 135 – 136, 2007. BDI Coordinator agent;Calendar information;Natural language dialogue;Smart Personal Assistant (SPA);.

[61] A. Mishra, P. Makula, A. Kumar, K. Karan, and V. Mittal, "A voice-controlled personal assistant robot," (Wellesely Road, Shivaji Nagar, Pune, Maharashtra, India), pp. 523 – 528, 2015. Assistant robot;Bluetooth networks;Personal assistants;Personal robotics;Robotic assistants;Smart devices;Smart Mobile Phones;Voice-controlled;.

[62] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[63] API.AI, "What are contexts and how are they used?," 2017-1-17. https://api.ai/blog/2015/11/23/Contexts/.

[64] A. Lebrun, "Bot engine," 2017-1-12. https://wit.ai/blog/2016/04/12/bot-engine.

[65] A. McConville, "Building ana the insurance chatbot, with watson conversation and bluemix: Part 1 of 3 - bluemix blog," 2017-1-17. https://www.ibm.com/blogs/bluemix/2017/01/building-ana-insurance-chatbot-watson-conversation-bluemix-part-1/.

[66] M. H. Van Emden and R. A. Kowalski, "The semantics of predicate logic as a programming language," *Journal of the ACM (JACM)*, vol. 23, no. 4, pp. 733–742, 1976.

[67] W. Clocksin and C. S. Mellish, *Programming in PROLOG*. Springer Science & Business Media, 2003.

[68] M. Sergot, "Knowledge representation c491." Department of Computing, Imperial College London, 2016.

[69] J. R. Firth, "A synopsis of linguistic theory, 1930-1955," 1957.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[71] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[72] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM journal on computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[73] F. Chollet, "Keras." `https://github.com/fchollet/keras`, 2015.

[74] M. Honnibal, "Introducing explosion ai," 2016. https://explosion.ai/blog/introducing-explosion-ai.

[75] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.

[76] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[77] S. Sukhbaatar, J. Weston, R. Fergus, *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, pp. 2440–2448, 2015.

[78] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[79] J. D. Choi and M. Palmer, "Guidelines for the clear style constituent to dependency conversion," *Technical Report 01–12*, 2012.