Open Source Intelligent Personal Assistant System

By

Stephens Robert

20183128

Submitted in partial fulfilment of the requirements for the degree

BACHELOR OF SCIENCE: COMPUTING WITH SOFTWARE ENGINEERING

In the

Department of Computing

SCHOOL OF ENTERPRISE AND CREATIVE ARTS

UNIVERSITY CENTRE FARNBOROUGH

FARNBOROUGH COLLEGE OF TECHNOLOGY

Supervisor: Gbenga Ogunduyile

MAY 2017

DECLARATION

I hereby declare that the dissertation submitted for the degree of Bachelor of Science in Computing with Software Engineering in the Department of Computing, at the School of Enterprise and Creative Arts, University Centre Farnborough, Farnborough College of Technology, Farnborough, Hampshire, United Kingdom, is my original work and has not previously been submitted to any other institution of higher learning. I further declare that all sources cited or quoted are indicated and acknowledged by means of a comprehensive reference list.

……..

STEPHENS, R

20183128

Contents

List of Figures

Abstract

There are very few open source intelligent personal assistants. The examples that exist tend to offload most of their processing to external servers, instead of relying on the increasingly powerful mobile devices they run on. This paper asserts that natural language processing can be performed on modern mobile devices using a mixture of machine learning and rule based models. The resulting system can processes natural language in real time on a mid range mobile device. The system evaluates queries about certain objects and subjects, extracting the object and any context related to it. The produced system is not a competitive product to rival major proprietary implementations, but forms a good basis for an open source intelligent personal assistant that can operate independent of server connections.

Introduction

Digital intelligent personal assistants have become popular recently with many major mobile operating systems offering a service (Siri (Apple Inc, 2011), Google Now (Google Inc ,2015), Cortana (Microsoft, 2014), Alexa (Amazon, 2014)). A digital personal assistant aims to provide more convenience to the user. This often involves performing basic internet queries with voice commands and outputting the results using speech synthesis, but can also include many other services. Digital assistants can be used in many scenarios where a traditional computing device cannot be used safely or conveniently. This includes situations like driving where the user cannot look at the device or cooking where the user may have clean hands and not want to touch the device. Most services provided by current systems are proprietary and require access to the internet.

The nature of a digital assistant requires personal, identifiable information to be stored about a user. With digital security and privacy a major concern of the modern world, users currently cannot use a mobile digital assistant without compromising their privacy, or the security of their personal data and storing it on a proprietary off-site server. Proprietary systems have no transparency for the user and how their data is handled. An open source system allows the user to observe how their data is handled, and verify that their privacy is not compromised, while also allowing the system to be independently reviewed and improved.

The existing systems require a persistent internet connection, this allows access to user data and handles most of the language parsing that takes place. Modern mobile devices

have powerful processors, which should allow the proposed system's parsing to take place locally, the system could not produce answers from the internet without a connection but could still function with access to the local data (useful for user related reminders, calendar dates, calculations). Processor speed is not the only relevant specification for language parsing, but is vital for real time responses. Hardware specifications that could cause issues are the relatively little RAM that mobile devices have and slow flash memory, these will be key for system optimisations.

Intelligent personal assistants are often associated with their range of integrated services. In the case of proprietary assistants, this is made possible by ownership and operation of existing converged services (Google Now has access to Google maps, Google calendar, Gmail and more (Google Inc, 2015)), this is not feasible for the proposed project. If the system is designed to be expanded upon and uses open formats for as much of the integration as possible, existing open source services can be incorporated into the system at a later date (email parsing, calendar modification).

Research Areas

Research will be primarily focused on parsing natural English language and defining the intent of that language. Questions are diverse in their format and the system will need to identify how the question will best be solved before solving. The primary subject, or term, of the question will need to be identified, along with any context that helps define that subject. This will require an understanding of syntactic and lexical analysis of the english language before implementation.

There are many services associated with personal assistants that require specific tasks to be performed (sending email, editing calendar, searching specific items). It is unclear the amount of automation that can be achieved when performing specialised tasks and whether these services will need to be handled by special conditions in the knowledge engine.

Middle ware exists for many of the functions required of the personal assistant (Voice recognition, voice synthesizers, natural language parsers), these should be evaluated with the positives and negatives of using the existing solution versus developing a new one, especially where performance is concerned.

An appropriate programming language will need to be selected depending on platform availability, performance, and available middle ware.

Objectives and Intended Outcome

- To produce an open source intelligent personal assistant.
- A Language parser with:
    - Reliable question recognition.
    - Reliable primary term recognition.
    - Good recognition of context from questions.
- Utilise a knowledge engine which provides adequate answers to questions

The project will be evaluated against these objectives, and to what extent it fulfilled these. The project will also be evaluated on its strengths and weaknesses and compared with existing services.

End User Requirements

- The system will be open source with a permissive license and use open formats throughout
- The system must be functional without an internet connection. Functions like calculations are good candidates for no internet operation, however some functions like internet searches are impossible to perform without the internet.
- The system must take natural language input
    - the natural language interface must recognise questions reliably
    - the natural language interface must recognise primary terms reliably
    - the natural language interface must recognise context reliably

Background Review

Literature Review

(Lakhani and Hippel, 2002) offers some insight into the reasons for open source success and the motivations behind contributors, especially in the areas of ‘mundane’ work. The proposed system would benefit from open source contributions like these, especially in the ‘mundane’ porting jobs and the less mundane services integration. The main motivation put forward is the incentive to learn, which is opportune for the proposed system due to the learning opportunities porting and service integration provide (learning a new architecture, learning a new user interface library, learning a new device API). (Lakhani and Wolf, 2005) expands on the motivations, adding that a considerable fraction are motivated by the nature of free software being free. (Stallman, 2009) clarifies the importance of the definitions of ‘free’ software versus ‘open source’ software. This explains the ideological motivation behind contributors and the importance of selecting the correct license for developers to not only be willing but to have incentive to freely contribute. These are import distinctions for the project, since free development on extraneous features will make it more competitive.

(Hauswald et all. 2015b) describes an open source intelligent personal assistant in detail, focussing on the Sirius system. Sirius has very different goals from the proposed system but the early paper is very applicable to any open source assistant implementation, especially in respect to the emphasis on performance. The proposed system will need to be optimised to run on consumer grade hardware, therefore specialised performance optimisations for assistants are useful, however as Hauswald continues the solutions proposed

become less applicable to the proposed system as Sirius optimisations begin to focus more on large scale graphics processing servers, beyond the consumer grade available. Additionally, (Hauswald et all. 2015a) expands on the optimisations in a paper on DjiNN, especially for neural networks, and presenting a new infrastructure for warehouse scale computing. Hauswald presents unorthodox solutions for neural networking at large scales. Most of the solutions are not applicable for the proposed system, however the bottlenecks identified provide a good starting point for initial optimisations.

(Goldberg, 2016) writes a concise primer on current neural network technologies in relation to natural language processing. This focuses on the processing of text and using many different techniques. Contrasting techniques are compared for instance relatively efficient markov sequences versus more accurate recurrent neural networks, these are valuable for the proposed system since accuracy concessions may need to be made for performance. The paper is a very effective introduction to the current technologies in a specialised field of research.

(Guha et all. 2015) approaches the field of intelligent personal assistants form the perspective of a search engine and service provider. The paper focuses on mobile device usage as well as building a profile on the user using converged services. The detail and methods described for modelling the user are very useful especially with note to solutions that seem effective but do not scale well. While the paper demonstrates some impressively accurate modelling, this is only made possible by storing large quantities of personal user data in external databases. The methods and model described could be very useful for the proposed system, assuming they can be implemented without compromise on the users privacy.

(Chaudhri et all. 2006) writes a case study on the knowledge base created for the Project CALO assistant. The infrastructure described is robust and conventional, the author catalogues challenges and models that will prove useful in developing the proposed system since the knowledge bases are very similar. The knowledge base described was never completed, hence it is difficult to evaluate the success of the project, however the lessons learned will be valuable to apply to the proposed system. Also produced for Project CALO, (Myers et all. 2007) presents a use case of an intelligent personal assistant for a knowledge worker. While this paper has a less technical focus, it does present the possible conveniences of an assistant system, however in the professional field. The proposed system will benefit from the conclusions of the paper, especially in respect to the emphasis on the importance of personalisation in the assistants responses.

(Devi et all. 2016) briefly introduces the different distinct forms of speech recognition, and discusses their implementation. Many of the core concepts and terminology of speech recognition are also covered, along with a definition of the commonly used word error rate metric. It includes a comparison of speech recognition applications and their accuracy using the word error rate, however the majority of these applications are not for English recognition and will not be suitable for the proposed system. The paper concludes largely in favour of Hidden Markov models as the preferred method of continuous speech recognition. This is a useful paper for evaluating different techniques, however the provided applications are not applicable to the system. Many of the fundamentals are briefly explained, allowing a quick entry into the field of speech recognition.

The book, Speech and Language Processing (Jurafsky and Martin, 2016) is a comprehensive guide to the techniques, methods and theory behind natural language

processing. It builds a thorough understanding of the field while remaining accessible for people new to the area. In covering topics such as n-gram models, part-of-speech tagging and word senses, the book acts an effective introduction to the functionality of the OpenNLP (Apache Software Foundation, 2017) tool kit's tools, despite having no affiliation. This is highly valuable for the proposed system, since these techniques are highly likely to be implemented. The book approaches all explanations from both a computing and linguistic background. This is especially useful in the explanation of parts-of-speech tagging where it breaks down the fundamental structures of language and explains how the tags relate to them. This attempts to build a real understanding of the systems at work, as opposed to other papers which give a simple one to one mapping of the tags. Although later chapters deal with information extraction, the techniques discussed are not as applicable to the proposed system as they might seem at first. The statistical models discussed are similar to the name finding methods of OpenNLP, but these are unlikely to be implemented due to device resource limitations. The most useful chapter for our application is definitely the deep explanation of part-of-speech tagging.

(Santorini, 1990) focuses on part-of-speech tagging, giving both an overview and observations on the technique. A list of each tag with a brief description makes up the first part of the paper. This serves as an excellent quick reference guide to the Penn Treebank Project, and is very useful for any developer not well versed in linguistics. The second half of the paper is concerned with explaining and dealing with problematic phrases that are hard to tag correctly. The final sections of the paper offer a list of words that often do not tag as expected, along with some general tagging conventions. These will be useful for validation and general parsing when handling user input for the system. The paper is overall a very good reference guide for development of any system using the Penn Treebank part-of-speech tags.

Existing Solutions

Sirius (Clarity Lab, 2015) is an open source intelligent personal assistant, that achieves portability by offering a web front end. It uses an experimental server design incorporating several different languages including python, c++ and java. The assistant uses a custom deep neural network for voice and image recognition. While a user can install and use Sirius, it is difficult and comes in many different configurations. This is due to Sirius not being targeted at the individual user, but at data centres. Sirius focuses on the bottle necks behind large scale intelligent personal assistant use and presents experimental solutions, including the DjiNN infrastructure and the offloading of neural networks onto a dedicated graphics processing unit. While performance is a important factor for an intelligent assistant, the solutions offered by Sirius are not applicable to the proposed system since the database transactions will be too few to take advantage of a custom infrastructure, and the most targeted devices do not have a dedicated graphics processing unit.

Mycroft (Mycroft AI Inc, 2017) Mycroft claims to be the worlds first open source voice assistant, and shares a lot of similarities with the proposed system. Mycroft is aided by its open source nature in two key areas, portability and additional services. Mycroft boasts a wealth of additional services that have been integrated by third party developers, these services are vital to remain competitive against other assistants with large corporate funding. This is a good strategy for independently developed systems but does require the overhead of a modular design. Mycroft has also been ported to many different devices by third party developers allowing it to integrate into more areas of life and adding general convenience. Both of these difficult and time consuming problems are kept competitive by licensing under a permissive open source license. This supports the openness of the proposed system by

offering additional benefits. Unlike the proposed system, Mycroft does require the use of proprietary external services and an internet connection for processing and data retention. This harms the aim of privacy that Mycroft presents, and can also be inconvenient without an internet connection. Mycroft mostly uses Python and handles speech recognition using proprietary middle ware.

Google Now (Google Inc, 2015b) is a proprietary intelligent personal assistant that uses the many services provided by Google to its advantage. It is integrated into most android devices and the chrome browser, while Google also offers apps for many competing devices. The assistant is very effective and free to use, with the condition of agreeing to the Google terms of service. Google collects and analyses data on its users including recording voice interactions for analysis, scraping email contents and generating a profile of the user. This can be a privacy concern, especially when this information is stored on Google servers, with no way for the user to delete or edit it reliably. Google Now is very similar to Apple's Siri Siri (Apple Inc, 2011) and Amazon's Alexa (Amazon, 2014)) in that they are vastly more competitive than open source alternatives when considering available services and voice recognition accuracy.

Methodology

Device

The android device used for development is a Wileyfox Swift smartphone (Wileyfox, 2016). The phones most relevant specifications follow:

- Qualcomm Snapdragon 410 8916 quad-core processor at 1.2GHz
- 16GB of flash memory (12.06GB available)
- 1.8GB RAM (around 800MB available)
- Android 6.0.1 operating system

The phone is a mid range device and is used as a baseline for testing, although some limitations imposed on the system could be remedied with a more powerful device. The processor is relatively slow compared with current flagship devices (The Samsung Galaxy S8 has an octa-core processor with four 2.3GHz cores (Samsung, no date)), however this is not the main bottleneck of the system. The system uses trained models for several processing tasks, these are large binary files that must be accessed from RAM (although a method could be devised to stream from flash memory, this would drastically slow the processing time). The current system uses between 200mb and 500mb which prevents the device from running other tasks simultaneously, this would be less of an issue on modern flagships due to increased memory available(The Samsung Galaxy S8 has 4GB). Since the targeted platform is Android 6, however, the minimum required RAM is around 1.8 GB (Google Inc, 2015a),

so this is an acceptable benchmark. The System must work on a minimum of this ram, although no multitasking will be available to the user on devices with this minimum, and it does hinder the available processing models for the system. As it stands, the current system uses two primary models, one for tokenization and one for parse tree generation. These use the majority of the systems memory, preventing other models from being used, and thus preventing available features (Such as accurate name, time, location, organisation, currency and data recognition within sentences). These features have been partially implemented without models, but are not as accurate.

Figure 1: The Wileyfox Swift used for development

Android Memory Management

The Androids Runtime (ART) environment is based on the java virtual machine, however with a more significant focus on performance. For instance, instead of Just In Time (JIT) compilation (A process where the application is compiled when needed), ART implements Ahead Of Time (AOT) compilation, compiling the application at installation, reducing processor load and power consumption when compared to JIT.

The Garbage collector is a shared feature of both the Java and Android environments, a system designed to automatically clear unused objects from memory. ART introduces a more aggressive Garbage Collector that can be somewhat overzealous when it comes to performance. The ART Garbage Collector has very little documentation on specifically what states an object must be in for it to be collected and destroyed. Official documentation concerns itself with optimisation tips and techniques that have been followed within the system, especially when dealing with the large model objects (For instance the large models are initialised as static, as per the recommendation (Google Inc, no data), to prevent collection). In practice the Garbage collection and general android optimisation system will destroy applications and objects if they are using what is deemed an excessive amount of system resources, without having an explicit focus. Due to the systems use of the majority of the devices RAM, the garbage collector undoubtedly targets it for immediate collection, this results in the system being closed whenever focus is lost. This severely limits the implementation of the system and it's reliability.

Android has two main application paradigms, activities and services. Activities are akin to a traditional graphical user application, with dialogs and graphics displayed and interacted with. Services are applications that run in the background, performing tasks when

required, without necessarily any graphics or dialogs. A service would be most appropriate for the system, since it only requires voice input and speech output, and would free the rest of the screen for multitasking. This is prevented by the garbage collector destroying the service due to it running in the background without a primary focus. This forces the system to be run as an activity, with the primary screen focus, to prevent closure. The solution is not to restart the system whenever destroyed, since it takes a non trivial amount of time to initialise the models.

Input

Voice recognition is an important factor in a personal assistant for ease of use. The ability to discern information or control a computer without the need to focus on the device or control it using peripherals is extremely useful for many situations.

There are several generally recognised forms of speech recognition systems, they are not necessarily distinct from each other and can contain common elements. The definitions are not strict but act as a guideline for identifying different systems. According to (Devi et all. 2016), they are as follows:

- Speaker dependant system
    - This is only suitable for a single user and is trained to their voice specifically. The training must be performed by the user and can be lengthy and repetitive.
- Speaker independent system
    - This will recognise most user voices without the need for training.
- Discrete speech recognition

    - The software can only recognise a single word at a time, so the user must pause between each word to allow the system to delineate words in a statement.

- Continuous speech recognition

    - The system can recognise all words in a naturally spoken sentence.

- Natural language system

    - The system can recognise continuous speech, but will also process the statement and respond in kind.

From this we can ascertain that a natural language system is the ideal for the developed system, as it incorporates some of the other main components of the system, namely processing and spoken output.

CMU Sphinx (Carnegie Mellon University, 2015)is capable of locally processed continuous speech recognition on desktop computers, however the accuracy is reasonably low (this could be due to tests being performed with an English accent since the models used were trained with US English, additional training could improve the accuracy) however the official android port has disabled this due to performance concerns on mobile platforms. The supported recognition systems that remain are as follows:

- Keyword recognition

    - This recognises a given keyword or phrase, the accuracy increases with the number of phonemes (the individual sounds that fundamentally make up all words in a language) featured in the phrase. This is a highly accurate input method,

however can only recognise one key phrase at a time, lending itself to the tasks of wake up or a limited discrete speech recognition system.

- Grammar search
    - This recognises distinct words from a given dictionary. The accuracy decreases, the larger the dictionary due to the difficulty in distinction between what phonemes construct what words.
- Language model search
    - Similar to Grammar searches in that this recognises distinct words, however this uses a language model instead of a dictionary. Sphinx supports two types of models:
        - Finite state grammar
            - Sentences are identified by working through words linearly with finite state transitions. These cannot be trained and transitions must be defined manually, making them lengthy to produce, especially for large dictionaries.
        - Statistical language model
            - Provides the probability of a word occurring in a sentence, allowing for words to be accurately identified from the given phonemes. This is a trained model and is much more accurate than finite state grammars.
- Phonetic search

- This does not recognise words at all, but instead recognises the phonemes used as input.

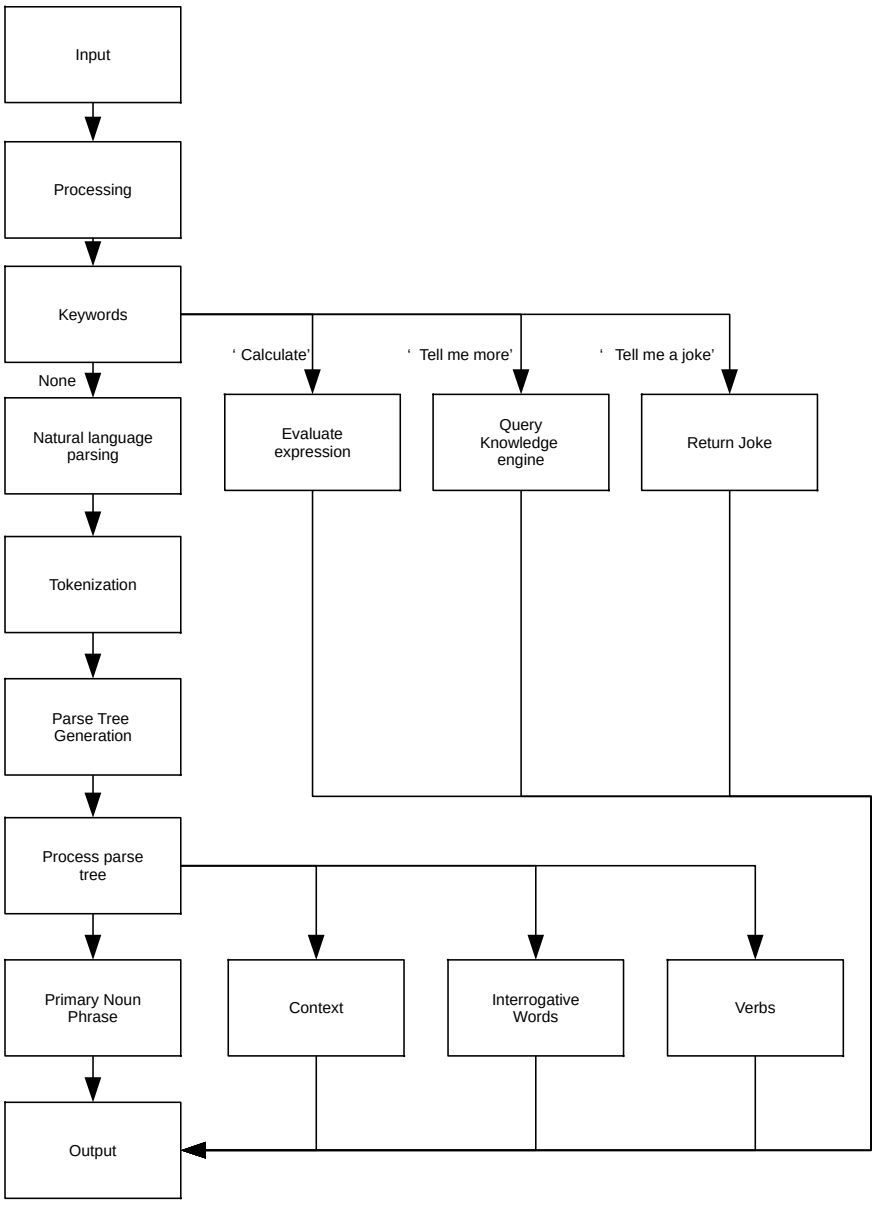
A highly trained language model search could offer close to continuous speech recognition results, however this is more suited to discrete recognition. A discrete system could have been implemented but has significant disadvantages when considered with the other goals of the project. A discrete system would use a very small grammar model, with specific command words. This would produce sentences and statements that are highly rigid in their structure, and possibly remove the need for any complex natural language parsing. Therefore, in order to showcase and allow more complex queries, continuous speech recognition is vital for the system. After evaluation, CMU Sphinx cannot provide this, and the current system has been implemented using the native android speech recognition service.

The Android speech recognition system utilises a hybrid system of both on device and off device processing. It is a built in library and handles continuous speech recognition well. The recogniser does not allow continuous background recognition and must be initially launched and brought to the primary focus. This could be accomplished by implementing Sphinx as a keyword recogniser to act as a wake up word and initiate the continuous speech recognition. Due to the existing concerns about memory allocation, implementing two speech recognition systems would only apply more strain to the system.

Processing

Once input has been received, this must be processed. Natural language processing is used to determine the intent of the statement, and prepare the appropriate output. This happens in distinct steps and will be broken down here.

Figure 2: The processing flow of the system



Keywords

The initial statement is checked to determine whether it starts with any specific command words. For now the pool of commands is small, but the system is flexible and can be easily expanded, the current commands are as follows:

- "Calculate"
    - The system checks if the first word of the statement is "calculate" and then parses the remainder of the statement as an equation. The calculation uses the EvalEx library (Klimaschewski, 2017) to evaluate the equation and can handle addition, subtraction, multiplication, division, and decimal numbers. The system then produces the answer to the query.
- "Tell me more"
    - The system checks if the statement is exactly the command before processing. This command requests more detail about the most recent query. This is then processed and a more detailed description of the queried object is produced.
- "Tell me a joke"
    - The system checks if the statement is exactly the command before processing. The system the produces a randomly selected joke from a predetermined roster.

If the statement satisfies none of these commands, the system moves onto more detailed analysis of the statement.

Tokenization

Tokenization splits the statement into meaningful elements, in lexical analysis these are known as ‘tokens’. The system uses a statistical tokenizer using statistical models provided by Apache (Apache Software Foundation, no date). The level of tokenization differs per system, depending on the form of parsing required tokens can be anything from single words , to entire phrases. We break the statement down to each distinct element, essentially anything separated by white space is a token. Punctuation is also tokenized as a separate element, however additional white space is discarded. It is possible this task could be completed by a series of conditional statements, as opposed to using a statistical model. This process formats the statement in preparation for further parsing.

Part of Speech Tagging

While part of speech tagging is a distinct process, the system performs it during parse tree generation as part of optimisations made. The statistical model required for tagging is also included with the parse tree model, so to prevent redundancy and save space in RAM these have been performed together.

This process tags the individual tokens with their relevant part of speech. Parts of speech are essentially categories for tokens that define their syntactical and grammatical context in a sentence. This helps narrow down the problem of parsing a sentence from possibly infinite combinations of words, to a combination of known tags. The tags are assigned using machine learning and a statistical model to determine the context of each token in the sentence. Machine learning is used because the mapping of tag to word is not explicit and can vary due to context, for instance the word cook:

'Please could you cook some food for me.'

The tags assigned are:

```
[VB] [MD] [PRP] [VB] [DT] [NN] [IN] [PRP] [.]
```

In this sentence, the word cook appears as the base form of a verb (VB). However this is highly dependant on the context, the word can also be used like this:

'Please could you ask the Cook to prepare some food for me. '

The tags assigned are:

```
[VB] [MD] [PRP] [VB] [DT] [NNP] [TO] [VB] [DT] [NN] [IN] [PRP] [.]
```

Here the word acts a singular proper noun (NNP). These tags are based on not only the words form itself (for instance the noun is statistically more likely to be proper due to the capital letter) but also it's position int the sentence as well as its neighbours (including the determiner (DT) 'the' before a word means it is statistically more likely to be a noun).

Parts of speech are separated into two types, open and closed classes. Closed class types have a fixed membership of words that are very unlikely to change, for example pronouns ('he', 'she', 'it') or determiners ('the', 'a',). However open classes are constantly gaining new members, for example nouns ('xbox') and verbs('to Google'). Closed classes could possibly be identified with conditional statements, however machine learning is required to accurately identify the appropriate tags for open classes based on trained data.

Tags

The tags so far discussed are using the Penn Treebank scheme of tagging, created by the University of Pennsylvania. This is the tag scheme that OpenNLP (Apache Software

Foundation, 2017) recognises and processes. It defines a number of tags that cover words, phrases and clauses. A full listing of tags is available in the appendix.

Parse Tree

A statistical chunking model is used to group tokens into contextual groups. The chunker performs the same task as part of speech tagging , with two additional layers of tagging, clauses and phrases. Clauses are the smallest unit of grammar that contains a complete statement, these normally consist of both a predicate and a subject. A subject is the object which the rest of the clause relates to. In a clause with a subject, the remainder of the clause would be the predicate, if there is no subject, the clause may be entirely predicate. Phrases are grammatical units made up of one or more words built around a head item. The head of a phrase is the word that determines the type of phrase it is, all other words in the phrase modify the head and are known as dependants. Heads are significant because they determine the direction of branching. If the head occurs near the beginning of a phrase, it is said to be head initial, this causes rightward branching. Conversely if the head occurs near the end of a phrase, it is said to be head final and causes leftward branching. In English, head initial is the most common type of phrase.

Given the statement:

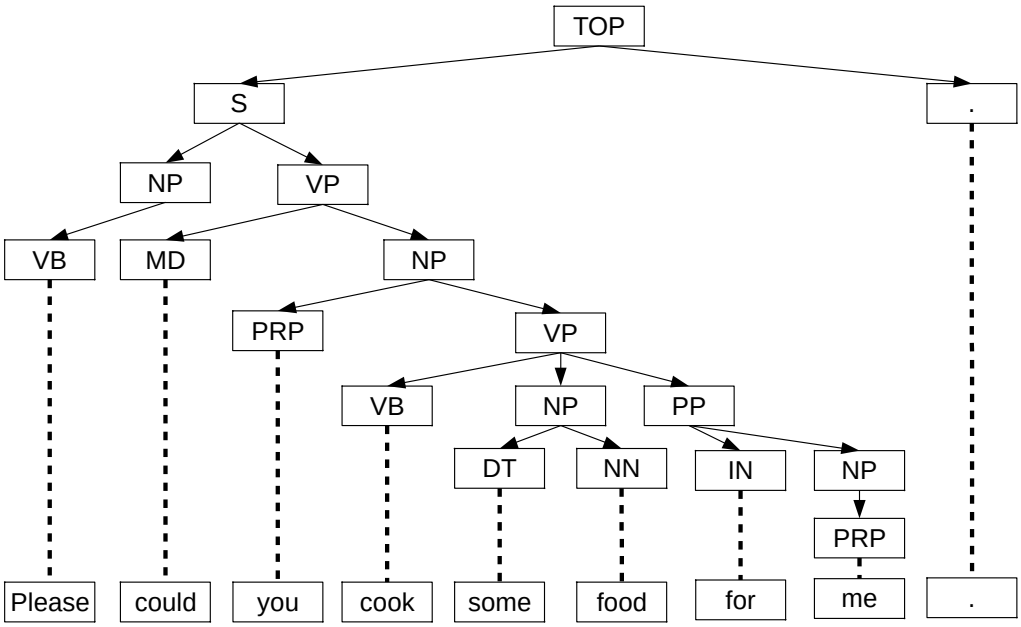
'Please could you cook some food for me. '

the parser will produce this tree:

```
(TOP (S (NP (VB Please)) (VP (MD could) (NP (PRP you)) (VP (VB cook) (NP
(DT some) (NN food)) (PP (IN for) (NP (PRP me)))))(. .)))
```

This is a linear representation of the parse tree, a visual representation follows.

Figure 3: A diagram of the parsed output

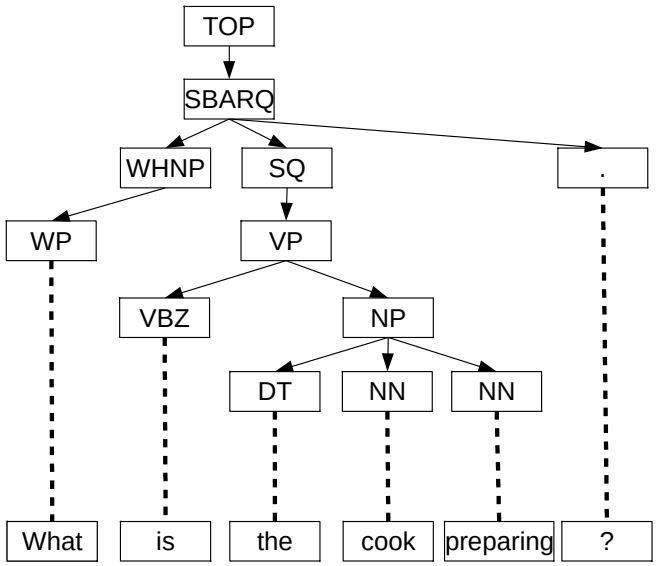


This provides a better representation of what the parser has produced. We can observe the nested clauses and phrases produced by the chunking process clearly. The TOP tag is a special tag that denotes the beginning of the tree. This is a simple statement and consists of a single clause (S). If we examine the phrase *'cook some food for me'* we can determine that it is a verb phrase (VP) headed by the verb *'cook'* (VB). We can note that the tree branches rightward. The statement is now in a format that can be processed by the system.

‘What is the cook preparing?’

```
(TOP (SBARQ (WHNP (WP What)) (SQ (VP (VBZ is) (NP (DT the) (NN cook) (NN
preparing))))(. ?)))
```

Figure 4: A diagram of the parsed output

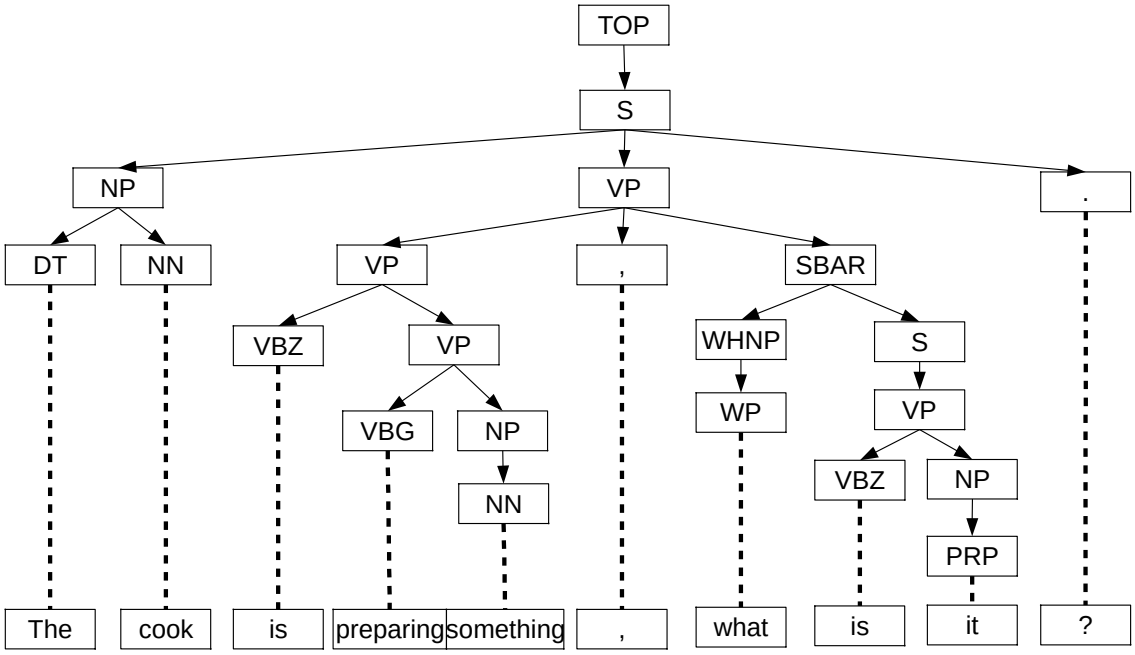


‘

The cook is preparing something, what is it?'

```
(TOP (S (NP (DT The) (NN cook)) (VP (VP (VBZ is) (VP (VBG preparing) (NP
(NN something))))(, ,) (SBAR (WHNP (WP what)) (S (VP (VBZ is) (NP (PRP
it))))))(. ?)))
```

Figure 5: A diagram of the parsed output



These examples demonstrate three different types of parse tree branching. Familiarise yourself with the structure of these linear parse trees. From here onward, only the linear representation will be presented.

Co-reference resolution is the process of resolving context of identifiers in prose. OpenNLP implements this using a statistical model, however the accuracy and reliability is low. The model is large and the implementation is specialised, using features of the Java virtual machine not present in Androids runtime environment, it also requires significantly more processing power than all previously described natural language processing techniques. This provides issues for both porting and optimisation on the mobile platform. As a result the system implements an optimised rule based co-reference engine, that has less features than the statistical model, but is accurate to a satisfactory level when processing a variety of questions and statements.

Parse Tree Processing

Questions exist which are ambiguous, to resolve ambiguity context is often provided. Given the question:

'What is Android?'

```
(TOP (SBARQ (WHNP (WP What)) (SQ (VP (VBZ is) (NP (NNP Android))))(. ?)))
```

This question could refer to the mobile operating system, or a robot. Although contextually a human would know we are referring to the operating system due to the subject of this report, the natural language processor does not have a wider field of context and can only use the given question as a basis. A human could possibly infer this through the missing determiner ('An android' would be more correct when talking about a robot), but this is subtle and leads to ambiguity. To apply context for the system, we must pose a question with context.

'What is the Android operating system?'

```
(TOP (SBARQ (WHNP (WP What)) (SQ (VP (VBZ is) (NP (DT the) (NNP Android)
(NN operating) (NN system))))(. ?)))
```

Specifying 'the Android operating system' provides the context needed to resolve this ambiguity, however relies on knowing that Android is an operating system. This is a valid question but a user asking this question may not have prior knowledge of the subject, and may not know to stipulate 'operating system'. Note that the parse tree has identified that 'the Android operating system' is a noun phrase (NP), and after a breadth first search this will become the systems primary term. If the user knows that Android is somehow related to mobile devices, they may use this context. The previous questions format does not apply well to this new context ('What is Android mobile?'). This sounds unnatural, a user would be more likely to use a naturally formatted query such as:

'On mobile, what is Android?'

```
(TOP (S (PP (IN On) (NP (NN mobile)))(, ,) (SBAR (WHNP (WP what)) (S (VP
(VBZ is) (NP (NNP Android)))))(. ?)))
```

Using this format presents multiple clauses, and multiple root noun phrases. As a human we can tell that the primary term should be 'Android', however it could equally be 'mobile' in the eyes of the system. This is where a rule must be introduced to discern the primary term. Here the term appears as the noun phrase in the last clause, while the other noun phrase is merely context. The same question asked in a different arrangement will help evaluate this rule:

'What is Android on mobile?'

```
(TOP (SBARQ (WHNP (WP What)) (SQ (VP (VBZ is) (NP (NP (NNP Android)) (PP
(IN on) (NP (NN mobile))))))(. ?)))
```

Here the primary term and the context both appear in the last clause as separate noun phrases, therefore the rule must make a distinction between noun phrases in the final clause. 'Android' is the subject of the clause 'is Android on mobile', this renders the remaining words as the predicate. Since the parse tree does not differentiate between subject and predicate, we must. Similarly to heads in phrases, subjects are often located at the beginning of the clause. A clause with a subject at the end is known as an inverse clause and rarely occur in English. When these do occur they are usually statements rather than questions. (for example the statement 'In a pocket, in the trousers, there is a phone.' is an inverse clause where 'phone' is the subject). If we apply this to the example and choose the first of the two root noun phrases in the final clause, 'Android' becomes our subject. Due to this, the assumption has been made that in all questions the first root noun phrase of the last clause is not only the subject of the clause, but also the primary term of the question.While this is a generalisation it applies well to the problem.

This handles resolving the primary term, however does not account for when the subject of the clause is a pronoun. Given the question:

'Android, what is it?'

```
(TOP (NP (NP (NNP Android))(, ,) (SBAR (WHNP (WP what)) (S (VP (VBZ is) (NP
(PRP it)))))(. ?)))
```

The primary term of this statement will be determined as 'it'. This will not allow the system to resolve the query properly, thus the object in which the pronoun is referring to must be identified. Pronoun resolution is a difficult problem that perplexes even statistical models due to it's inherent ambiguity.

For example 'the master told the cook to make his lunch' has an ambiguous pronoun, whose lunch is being made, the cooks or the masters? A human could infer it is probably the master's lunch due to the implied master cook relationship, however this is not a certainty and assumptions must be made. In the above example there is no ambiguity and the pronoun can be substituted with the first noun phrase that occurs. With the introduction of context this ambiguity can often present itself, for example the query in the form:

'Android on mobile, what is it?'

```
(TOP (NP (NP (NP (NNP Android)) (PP (IN on) (NP (NN mobile))))(, ,) (SBAR
(WHNP (WP what)) (S (VP (VBZ is) (NP (PRP it)))))(. ?))
```

The system must resolve whether the primary term is 'Android' or 'mobile'. By applying the same generalisation, that the pronoun refers to the first root noun phrase, 'Android' will be selected. However this is not always the case.

'on mobile, there is an operating system called Android, what is it?'

```
(TOP (S (PP (IN on) (NP (NN mobile)))(, ,) (NP (EX there)) (VP (VBZ is) (NP
   (NP (DT an) (NN operating) (NN system)) (VP (VBD called) (S (NP (NNP
Android))(, ,) (SBAR (WHNP (WP what)) (S (VP (VBZ is) (NP (PRP it)))))))))
                                    (. ?)))
```

In this case the system will determine the primary term to be 'mobile' since this is the first occurring root noun phrase excluding the existential there (EX). 'Android' and 'operating system' will be detected as context which will allow the query to return the correct result, however this is not ideal. 'Android' should be the primary term, but is not due to the ambiguity and the assumed rule. When evaluating the assumption the context of it's use must be considered. The queries asked of the system are used to resolve a definition of an object, if this is accomplished the system has succeeded. If the objective was to accurately ascertain the primary term, this assumption would be unacceptable. To accurately determine the pronouns subject relies on many contextual nuances of the subjects of the clauses of the question. In a rule based system there would need to be many handled edge cases to the point where it could be considered infeasible due to the infinite combinations of open classes in the English language. Similarly abstract language use will produce unintended results, for example the question 'what on earth is Android' will provide the context of 'earth' when this is merely a figure of speech. This is an appropriate application for machine learning for these reasons, however this would significantly harm performance and the accuracy will not necessarily be much improved. Another consideration is the likelihood of this format of question being asked, users are more likely to be succinct when asking a simple query as opposed to this roundabout sentence.

The parser will often include determiners in the noun phrases it identifies. If the primary term begins with a determiner this is removed by the system to improve queries. While there are legitimate items that begin with a determiner, these usually resolve correctly without. As opposed to items given erroneous determiners which can disrupt the query.

A pseudo code representation of this rule based engine follows:

Figure 6: Simplified pseudo code for the rule based engine

```
SET questions to an empty array
SET interrogativeWords to an empty array
SET verbs to an empty array
SET nounPhrases to an empty array

SET primaryNounPhrase to empty

FOR each node in parse tree
    IF node is a question clause THEN
        ADD node to questions
    ENDIF
    IF node is a interrogative word THEN
        ADD node to interrogativeWords
    ENDIF
    IF node is a verb THEN
        ADD node to verbs
    ENDIF
    IF node is a root noun phrase THEN
        IF node is the first root noun phrase in the last question clause THEN
            SET primaryNounPhrase to node
        ELSE
            ADD node to nounPhrases
        ENDIF
    ENDIF
ENDFOR

IF primaryNounPhrase is a pronoun THEN
    SET primaryNounPhrase to nounPhrases[0]
    REMOVE primaryNounPhrase from nounPhrases
ENDIF

IF primaryNounPhrase starts with a determiner THEN
    REMOVE determiner from primaryNounPhrase
ENDIF

END
```

Final Processes

The OpenNLP library provides many more processing options to allow extraction of names, places, organisations, times or any other type of data in language given an adequately trained model. These are reasonably accurate and fast but again require more space in memory, preventing their use for the system. A replacement is partially implemented using the reference engine, which extracts the primary noun phrase of a question. This substitutes for the purpose of these models, however cannot ascertain the type of data the phrase is.

Once the primary term and context have been identified, the system constructs a query for the MediaWiki API to extract information from Wikipedia. The result of the query differs with the certainty of the object. If an entry exists for the object the introductory sentence will be returned. However if this is not certain, or more context has been provided, a snippet of the most relevant part of the most relevant entry will be returned. In the cased of a 'Tell me more' command, the query is repeated with the stipulation that a larger snippet of article be returned.

Output

Similarly to the input method, an output that does not require the user to focus on a screen or interface is extremely useful for many different situations. While still a non-trivial problem, satisfactory sounding performant speech synthesis has existed for a significant period of time.

Android provides a text to speech API that provides a simple way to initialise and queue speech output to a background service, providing a standardised method for speech synthesis on android. By default this API uses the Google Text-to-Speech service however it

will use whatever the user has installed and set as their default text to speech engine. This allows for a high level of user customisation without any additional complexity levied on the developer. The downside to this is that the interface is android specific and this adds difficulty to implementing platform independence. To comply with the objective of open source, an alternative to the google service was identified and intended to be used.

There are several distinct forms of speech synthesis, a synopsis of the most commonly used according to (Lemmetty, 1999) follows:

- Articulatory Synthesis
    - This technique uses a computational model of the vocal tract to synthesise speech. This is one of the most accurate techniques but is relatively computationally expensive. Other systems, such as formant, can produce speech with a similar quality, but cannot emulate the idiosyncrasies of biological speech (for example transients, short high amplitude sounds that occur irregularly during changes in the vocal area). Articulatory synthesis can emulate these without using special cases, which is a major advantage in creating more realistic speech.
- Formant Synthesis
    - A formant ,in phonetics, is a resonant frequency of the vocal tract. Formant Synthesis uses a collection (usually between three to five) of these in an acoustic model along with additive noise and frequency modifiers to generate a waveform that resembles speech. The acoustic model is often a simplified model of the physical properties that govern the creation of waveforms. The speech generated is often notably unnatural, but it is relatively easy to achieve highly intelligible

results with this technique. Since the waveforms are entirely algorithmically produced, any waveform can be produced, resulting in a highly flexible speech synthesis system. A wide variety of voices, intonations, and rhythms can be achieved by varying the model and it's modifiers. This system must store a model, but this is relatively small since it is primarily algorithms. The system is also relatively cheap to compute.

- Concatenative Synthesis
    - One of the most computationally cheap methods to achieve natural sounding speech, Concatenative synthesis relies on splicing prerecorded samples together. The efficiency of the system depends on the sample length used, shorter samples (for example phonemes) can result in an increased processing times, while longer samples (for example entire words) may require significantly more storage space. The larger the samples used, the less natural the output may sound, generating discontinuous sounding sentences. This can be offset slightly with additional processing to change tempo and intonation. Due to the nature of prerecorded samples, it is difficult to achieve significantly different voices, without storing another database of samples. The storage used by this method, even with the minimum number of samples required, is relatively large.

The most suitable method for embedded and mobile devices is Formant synthesis due to it's relatively low processing and storage requirements.

The eSpeak text to speech engine (Dunn, 2017) uses formant synthesis and presents a highly flexible speech synthesis system. It supports over 45 languages, and regional accents

for those languages. However the current Android port has been abandoned and does not function on modern devices.

The Google Text-to-Speech service is highly optimised for android devices, although it is difficult to ascertain how due to its proprietary nature. The system uses external model files for voice and language, and these are around 2mb to 3mb in size, this seems small, but could easily fit samples of the most basic phonemes. The output sounds reasonably natural, and is very intelligible. It functions well without deferred processing over the internet. These features could both be attributed to either formant or Concatenative synthesis. The only clue given is the 2010 purchase of Phonetic Arts by Google, a video game technology company creating performance focused speech synthesis based on sampled voice (Jennings, 2010). This suggests Concatenative synthesis, however it could possibly be a hybrid of both, modifying samples using an acoustic model. We can only speculate due to the closed nature of the service.

Open Source and Licensing

Software licensing is an important issue in any project and must be treated with respect. The terms ‘Open source’ and ‘free software’ are often misunderstood to be synonymous, however there are clear, albeit subtle, distinctions between the two. In common parlance they are often used interchangeably to refer to the same software, however the distinctions are important when speaking philosophically, and more importantly, legally.

The use of the word ‘Free’ in terms of software may merely mean software without cost, however when speaking in terms of licensing, free software is considered in terms of the

freedom of the user and how the software respects those freedoms. (Stallman, 2017) defines free software in much greater detail.

Similarly, in legal terms Open source software does not merely refer to software that the source code is available for. However it does usually refer to software with available source that explicitly includes a license that permits inspection, modification and enhancement (Opensource.com, no date).

A discussion of the most relevant licenses to the system and it's middle ware follows:

- Apache 2.0
    - A permissive license that protects copyright and license notices in the source. It allows modification and distribution but also commercial use. The license protects the trademark of the software and removes the liability of the original author. OpenNLP uses this license (Apache Software Foundation, 2004).
- MIT
    - A permissive license that also allows modification, distribution and commercial use. This license only requires that the license and copyright notice remains in the source, while also protecting the original author from liability. EvalEx uses this license (Massachusetts Institute of Technology, no date).
- BSD
    - A permissive license that has very similar terms to the MIT license. CMU Sphinx uses the BSD style Carnegie Mellon University license (Carnegie Mellon University, 1999).

- GNU General Public License v3.0
    - A copyleft license that allows commercial use, modification and distribution. The copyright and license notices in the source must be protected. Unlike the previous permissive licenses, if a GPL 3.0 licensed library is used by software, that software must also be licensed under the GPL 3.0 and the source must be distributed. This is designed to proliferate the spread of free software, but can ward off some developers. eSpeak uses this license (Free Software Foundation, Inc, 2007).
- Creative Commons Attribution-ShareAlike 3.0 Unported
    - While not specifically a software license, this license can govern many types of product. It allows copying, distribution and adaption of the work while requiring attribution and any alteration of the work must use the same license. The license protects some of the authors liability. Text retrieved from Wikipedia is protected under this license (Creative Commons, no date).

Although mostly open source, Android is not considered free software (Stallman 2011). Free software applications can be used on Android, however some of the freedoms expected will be infringed by default due to the operating system. Many libraries provided by Android function as software as a service, thus nullifying the right of any software using these libraries to be free software (Stallman, 2016). This includes the speech recogniser and speech synthesis libraries utilised by the system. These are important considerations, while the system is open source and distributed without cost, it would be disingenuous to call it free software.

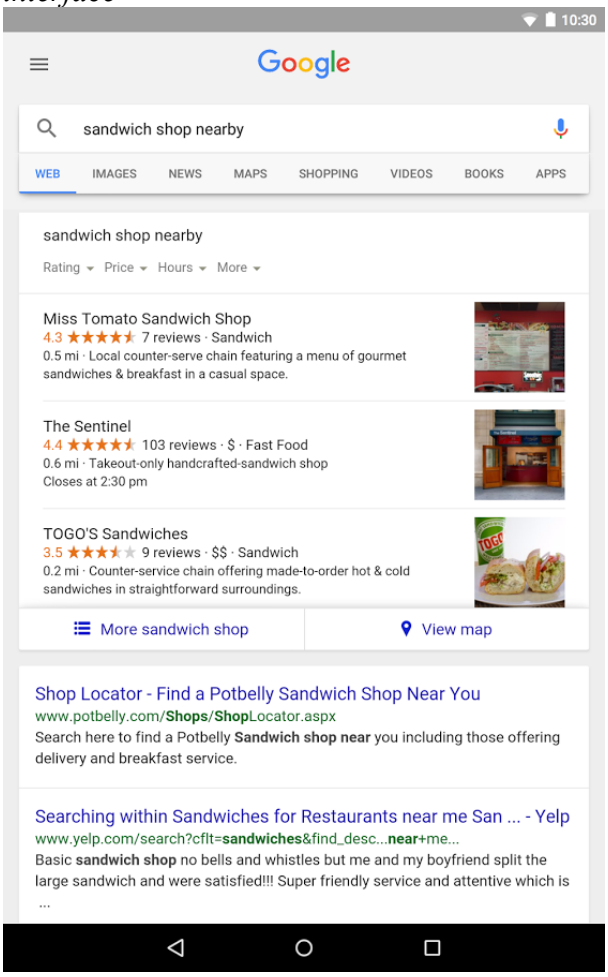
Design

Many aspects of the systems design have already been covered in the methodology above, however this section will cover aspects that are inherent to design that have not been already mentioned. This includes elements such as the visual design, but also the general design of user interaction. The internal component design, especially the natural language processing methods were covered in detail in the methodology. Here we have covered the more subjective elements of design and their justifications.

Visual Design

Ideally a personal assistant system such as this will work as a background process and not necessarily require a graphical user interface. Due to constraints explained above, this is not possible for the current system, and must implement a basic graphical user interface to enable the speech recognition. Other mobile personal assistants have implemented user interfaces. Both Google Now and Siri implement an interface that supplements their primary voice interaction with visual output.

Figure 7: Screenshot of the Google Now interface

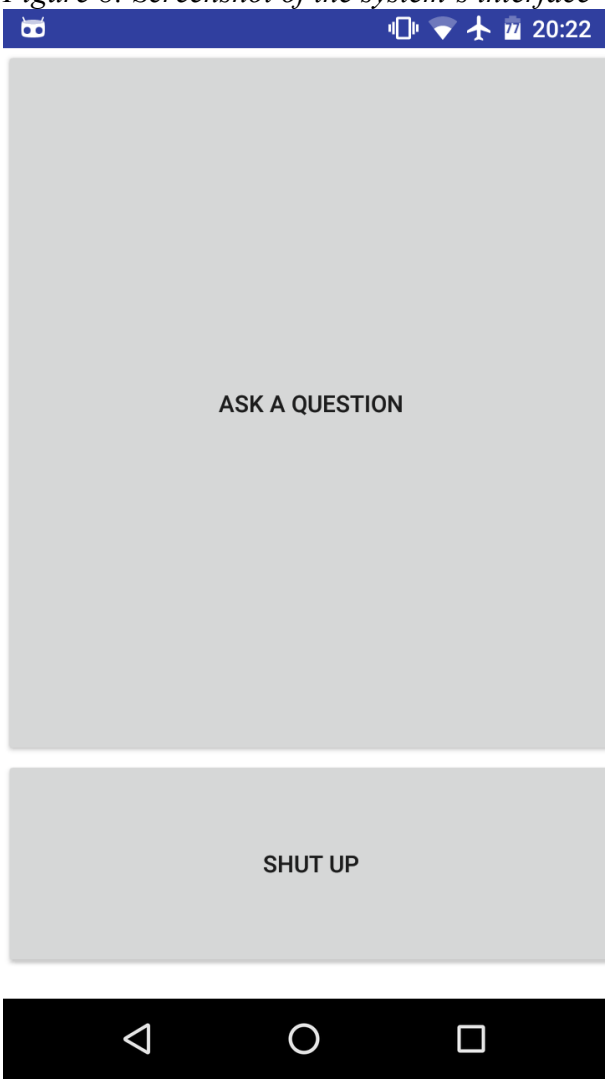


(Google Inc 2017)

Google Now's interface primarily presents query results and provides a simple text search and tabs for sorting results. This is essentially an implementation of the Google search page for web browsers within Google Now. This does not allow for quick dissemination of information at a glance and definitely does not allow navigation without looking and focussing on the device due to the small buttons. Siri has a very similar interface. This establishes that the current implementations attempt a fundamentally different goal than the interface for the system.

The interface should be optimised to require as little interaction as possible, it unfortunately cannot be completely hands free, but it must be as close to this as possible. Another element of usability is the ability to use without looking or focussing on the screen, so the design should be optimised for use without the user looking. Large buttons with consistent locations allow for ease of use without focussing or even looking at the device, this is also helped by a simple two button design. The large buttons also enable use with any capacitive surface, not necessarily just fingers. For example an elbow can be used when hands cannot, this is useful fo scenarios like cooking.

Figure 8: Screenshot of the system's interface



Audio Interface

Interfaces are not only visual in nature, a robust audio interface is vital for a mobile assistant. Development has been primarily focussed on allowing natural language search queries to be evaluated and resolved allowing users to pose a question to the system as they would to another person. This improves the user interaction greatly as this is second nature to most people.

The system also makes use of some set commands for certain actions. The ‘tell me more’ command returns more information from the current query, while the ‘tell me a joke’ command returns a randomly selected joke. The ‘calculate’ command treats any words after the command as an equation to solve. While these commands are statically formatted unlike the previous queries, they are more efficient. There is no natural language processing required and the user can say a short command instead of formatting their request as a full sentence. Commands are a good complement to natural language input, especially for tasks that involve exact procedures, or figures that require a very specific output. The ‘calculate’ command is the most relevant example of this, however the other commands have very specific outputs as well, making this input format appropriate for them.

Personification

A key tool in a good audio interface is user engagement. To help achieve this Apple personifies their mobile assistant by giving it a name and natural language responses. Google Now does not implement this much, however the new Google Assistant looks to engage the user in two way conversation as a means of personification. While the system itself is not very intelligent by human terms, creating the illusion of humanity can significantly improve

the users relationship and engagement with the system, as well as their perception of it's intelligence.

Efforts have been made to personify the systems outputs. Whenever a query is returned with an uncertain result, the system prepends a preset uncertainty warning from a random pool of warnings. This allows the user to intuitively understand that the response is not necessarily the correct answer, and sets a conversational tone that encourages responding in natural language rather than static commands.

Figure 9: An example of an uncertainty warning

'I'm not sure, but this could be it: '

All errors and validation issues are communicated to the user using natural language in an effort to not break immersion. Using terms like 'shut up' for halting speech output attempt to make the simple function seem informal. In a similar vein to Siri, jokes can be returned on request, this improves the informal atmosphere of the system and allows further personification. The jokes are deliberately computer related and inoffensive to continue the theme of an artificially intelligent virtual assistant, while remaining appropriate for all users.

Figure 10: An example of a joke

'I would tell you a UDP joke, but you might not get it.'

Testing

Methodology

The system was tested with main functionality and all user input and output in mind. A primary focus of the system is context and primary term resolution so this has been tested separately. The majority of tests have been performed on the Wileyfox Swift development system, but where it has been appropriate to perform tests on additional systems, this has been noted.

Reference engine testing has been approached from the context the system will be used in asking object identifying questions, however the full output of the reference engine has been provided to allow further analysis. This test is performed in a separate environment using just the reference engine and natural language parsing tools, but is equivalent to the internal procedures of the final system. The system has been posed many variations of the same question to allow a more comparable analysis of the results. The full output is provided to allow insight into the structure of the engine and speculation about future improvements and additions possible.

The data retrieval abilities of the system are tested using the system itself to query a set of questions. These questions are based on the same set as previous tests, to allow analysis of the reference engines application, but additionally feature alternative contexts to demonstrate this resolution (the Android robot or operating system duality). There are also

erroneous tests and network connection tests to check basic validation performed by the system.

The input accuracy is tested by posing a statement to the Google speech recognition system and recording the direct output. This has not been a comprehensive test due to the closed nature of the system, preventing debugging and improvement by independent developers. Alternatively, this has been a test of adequacy to allow evaluation and assess the suitability to the task it performs for the developed system. Each statement was posed five times, with the mode (most commonly occurring) of the results recorded. The word error rate is also included, a lower score is better. Statements are again based on earlier tests to allow comparison, however new statements have been introduced to test more difficult nouns. Each test has been performed online and offline to allow analysis of the importance on network processing for Google's system, and to evaluate the ability of the system to remain accurate without a network connection. The tests involving background noise used a sample of television interviews played at just under the volume of the voice of the person performing the test. This is an attempt to simulate a crowd by exposing the system to multiple high quality voice inputs at once. In an attempt at fairness the samples were played from the same time stamp for each test. The questions were however posed using a person, so this is not an entirely fair test due to the idiosyncrasies of human speech.

Figure 11: The equation for word error rate

```
Word Error Rate = (Substitutions + Deletions + Insertions) ÷ N words
```

The systems output is primarily tested in terms of the output message that is generated rather than speech that is synthesised. The joke system is tested by posing the command and recording the output. This is performed twice to demonstrate that it functions identically with

or without network connection. The calculate command is tested against all supported mathematical operations as well as erroneous and incorrect equations to assess validation. The command is tested offline to demonstrate that it also functions identically online and offline. The command to retrieve more information from the current query is also tested here. This is evaluated in terms of extracting more information from a null query, an ambiguous query and a valid query. There is also a network connection test to check basic validation performed by the command.

Other than speech recognition, a set of buttons are provided to the user as input. These are tested on whether they perform the function they are intended to. The Shut up function is tested by first performing a query, and then attempting to halt the output of the response.

The likelihood of the system being destroyed by the garbage collector is tested using a series of scenarios. The applications presence is checked at 30 second, 5 minute, 10 minute and 30 minute intervals to ascertain whether it has been destroyed. Note that after every check the timer must be reset as the application has just been brought to focus and reset the garbage collector. These tests have been performed while disconnected from the Android debug bridge (ADB), so a stopwatch is used for timing. The scenarios chosen aim to mimic realistic background use, with the system minimised on it's own first, and then testing additional application use. The additional application used is a web browser browsing news articles with few images and no video, the application used to simulate high RAM use streams high definition video from the internet.

Due to the high initial loading time of the application, this has been tested separately, as well as on multiple devices. Ten results have been recorder per device and averaged. The time was measured using the ADB and a timestamp out put when the system first allows user

input. This is an effective method of accurately measuring loading time since it starts when the operating system reports the application has been requested to activate and stops when the application is actually functional. The devices used had full charge, were connected to a power source and had no background applications running (although Android does have many background processes, this is usually beyond user control). In addition to the Wileyfox Swift, a Motorola Moto G (5) (Motorola, no date) was used for testing. This mobile device is similar to the Swift in many respects with the key differences being an octa-core 1.4GHz processor and the Android 7.0 operating system, otherwise they are equivalent. The other device used is the Nvidia Shield Tablet (Nvidia, no date), this a tablet format mobile device as opposed to the phones previously used, with a much more powerful quad-core 2.2 GHz processor. It also uses Android 7.0, but other than this the ram and flash memory is again equivalent to the Swift.

The RAM usage is tested to gain more insight on the garbage collection methods and estimate the amount of RAM required by the application. RAM measurement is performed through the ADB and is accurate for the application itself but does not take into the account the resources used by background processes, for example speech recognition and speech synthesis resources are not accounted for. Measurements were performed five times each, using a fresh launch of the application, the average of each test was then calculated.

The Android Studio integrated development environment was used throughout testing to monitor specifications and debug messages over the ADB.

Test Data

A full listing of test data is included in the appendix.

Evaluation

Testing was largely successful with most tests meeting the expected outcomes. We will now cover observations and analysis of these results.

Determining the primary term was largely successful with most tests performing as expected. Extracting the primary term in the question ‘on mobile, there is an operating system called Android, what is it?’ (Test 7) failed, however due to reasons explained in the above system methodology, this is acceptable and can still be used to form a valid query. For all questions we can see that the relevant interrogative word has been extracted (what, when, who et cetera) and in test 6 the verb ‘called’ has been identified. These are not used in the current system, since it is intended to solve object identification questions, but they are useful for identifying different types of question and their intent. For instance the question ‘When was Grace Hopper born?’ will have the ‘When’ and ‘born’ detected as well as the primary term of ‘Grace Hopper’. This allows resolution of the fact about the object that is specifically required. The accuracy of these terms being detected is useful for future improvement of the system.

Most data is retrieved correctly. The system does run into issues with ambiguity when sentence fragments are being queried. Fragments that only consist of noun phrases are treated as a search term themselves, this allows for more efficient queries if the user does not wish to speak an entire sentence. However due to this, the system itself has to resolve the ambiguity and can often make a mistake (like the example in test 16, where an article related to but not about Android is retrieved). The system also handles fragments that are not valid for queries well. Snippets of an article are often retrieved when context is applied due to the uncertainty of which object is being queried, the first line of an article is usually retrieved when the

system is certain of the object being queried. This makes the output of test 11 especially successful, because it demonstrates that the additional context lead to a absolutely certain resolution, rather than an almost certain resolution. The system also informs the user if the network connection is unreachable reliably.

The Google speech recognition system performs admirably offline as well as online. We can observe the key difference network processing makes is the accuracy of detecting uncommon nouns and words. The results of the 'What is Android?' tests (25 and 26) are especially impressive due to the additional online processing allowing the system to resolve the ambiguity of 'Android' being used as a proper noun. This demonstrates the speech recognition is more than capable for the system and could be eligible for offline use for future improvements.

The tested outputs are all appropriate and expected. The calculate command performs well parsing text as equations, although it cannot calculate beyond simple arithmetic operations. The 'tell me more' command works effectively and the provided jokes are randomised.

Both button inputs function as intended.

The behaviour of the Android garbage collector defies expectations, never destroying the application during testing despite countless anecdotal evidence of this being commonplace. This is positive, but does suggest more testing is needed. Without much documentation, it is difficult to ascertain what factors cause the garbage collector to activate. These results demonstrate that it is more complex than the originally assumed parameters (high resource using applications that are not in focus get collected). Ideally the system

would operate continuously in the background throughout the daily operation of the device, a maximum test of 30 minutes does not test this scenario well but was chosen due to time concerns and the assumed garbage collection before the time was reached. This is especially confusing when tested with another high RAM use application, with little RAM to spare the testing device functioned without any noticeable issues or garbage collection.

The loading speed behaved as expected for two out of the three devices tested. The Moto G loaded the application significantly slower than the other devices, despite being considerably more powerful than the Swift. This is not an issue with Android 7.0 since it performed well on the Nvidia Shield, therefore it may be an issue with the hardware. The processor is faster than the swift but this could be hindered by other bottlenecks, for instance slow memory or flash storage devices could significantly slow the loading as the system decompresses models from flash to RAM. We can observe that the loading times per launch remain reasonably consistent for each device.

The RAM use tests shed some more light on these results. The garbage collector only activates when memory begins to run out, in order to save on processing. Interestingly the RAM use is reasonably inconsistent across repeated tests, implying that garbage collection is more heavily tied to the behaviour of background processes than originally thought. The RAM usage overall is acceptable, but it must be reiterated that these empirical readings are purely the memory use of the foreground application and does not account for the background services employed at the same time.

These results reveal some peculiarities about the Android platform that could have been useful during development time, especially concerning garbage collection. Overall however the tests were very successful and demonstrate that the system is fully functional.

Evaluation

Over the course of this evaluation we will assess key elements of the system. First decisions taken during the methodology and design stages will be evaluated, before moving onto evaluating the system against the goals set out at the genesis of the project.

Development

The system development progressed reasonably quickly and with few issues. During development a single device was used for execution and testing. This led to assumptions about the hardware and Android in general that were sometimes incorrect and had to be adjusted at later stages. A not insignificant portion of development involved porting and optimising the natural language parser for android use, especially when concerned with memory usage. Due to relatively low memory of the chosen device, significant portions of the natural language processor had to be removed to allow for the system to initialise correctly. With a larger variety of testing devices, specifically access to devices with larger memory, these features could have possibly been incorporated. As it stands these functions were partially reintroduced in the reference engine, although with less features.

The system initially began development as a python application due to the languages wealth of natural language processing resources, however switched to Java due to androids poor python support. Python applications can be installed on Android, however they often have limited access to system resources and must access system services through wrappers. This proved difficult to implement python natural language libraries that had specialised

requirements on Android, so development was halted and the system switched to the native Java language. This hurt development time significantly and could have been avoided with more thorough research at the analysis stage.

The Android implementation of Java itself also proved problematic. Android uses a non standard runtime environment which still uses the Java 7 language, a language which lost official support in 2015 (Java, 2015). Several of the middle ware libraries selected are currently using Java 8, most notably OpenNLP, and thus older versions were used for development. Due to the middle ware being open source this can be solved by removing any Java 8 specific features from the source and compiling with the older language version as a target. This would have slowed development significantly so was not performed. The libraries can be replaced at any time without the need to modify the system's code, so this is a key improvement that can be implemented at a later date with relative ease.

While the flow of information has been handled well by the system, the software architecture was given little design time and was modified during development. A key issue for a digital personal assistant is available services. The system being open source was in part chosen to allow additional services to be integrated by separate developers at a later date. In the current system it is not difficult to add additional commands, but a formal method integrated into the system architecture would have been a great improvement. An example that could be implemented in the future is a plugin architecture, although this would require major changes to the method in which commands are processed and selected. This would be more than worthwhile addition because it enables individual control of the system services for both users and developers. A plugin architecture can also aid cooperation using source

control, allowing separate modules to be added and loaded without ever having to edit the main processing code.

Input

The system primarily accepts input using speech recognition, but also utilises two capacitive buttons for initiating and halting speech requests.

The Speech recognition is very successful, with a highly accurate and fast recognizer. This is due to the use of the Google speech recognition system, it is very effective at what it is intended to do, but is not without significant downsides. The original intention was to use the CMU Sphinx recognizer, but due to performance and memory issues, this necessitated the switch to another system. The Google recognizer is simpler to implement, being a library provided by android itself, faster performing and more accurate. However it is not entirely free software, relying on network services beyond the control of the user. This could be a major privacy concern, data such as voice, the query made, and if the user has an account attached to the device, who made the query can be recorded by Google. This is the major advantage of CMU Sphinx in that it is entirely free and open source, with all processing performed on the device, unfortunately this is not practical for the current system and the compromise on freedom versus convenience had to be made. Another boon to Sphinx is that it can be trained personally, this allows the recognition of extremely uncommon words (for example unique nouns). While the Google recogniser is very good at recognising uncommon words, the user is at the whim of Google for the addition of new words.

Another compromise was the addition of the buttons to the application. These have been made extremely large to allow controlling the application without looking or focussing

on the device, however they dominate the visual design. Ideally these buttons would not have been needed, but due to resource allocation and performance concerns, they were a necessity. The system will not reliably persist in the background on android, and the Google speech recognition service cannot be used to recognise wake up words, this results in an application that must run in the foreground, and will only begin recognising speech after a button has been pressed. There are only two buttons to minimise the amount of physical input required, while also maximising the size of the buttons. These aspects are important to retain any use without looking or focussing on the device. To return to the scenarios given in the introduction, while cooking an elbow can be used to activate speech recognition without dirtying the hands, and the buttons can be pressed while driving without looking at the device. A slight benefit to this versus Google Now, is that the system is not always recording speech data when idle, waiting to be woken up. This is a significant privacy concern, as anything said within the proximity of the device may be recorded and may be sent to Google. With the current system, only the queries explicitly posed to the device have the chance of being sent to Google.

Processing

The system processes natural language input entirely on the device in real time. OpenNLP runs successfully on Android despite being intended for desktop use. The reference engine created for the system is very fast and very accurate. The engine only resolves questions that relate to finding out what an object is, however this is still very useful. A top consideration for improvement would be adding additional types of question to resolve. This would however become more untenable the more rules and special cases are handled. Machine learning can overcome these difficulties and should be considered if more question

types are to be added. Questions are a difficult subject to analyse, and the training data required for machine learning would have to be vast and most likely supervised in order to achieve proper entity reference and extraction. We can see this issue with OpenNLP's co-reference engine, being highly inaccurate, unreliable and unable to function on Android due to special requirements. Due to these issues it is difficult to advocate whether rule based systems versus machine learning are more appropriate to the problem at hand, although rule based systems are definitely more appropriate to the current Android implementation.

Due to memory issues the current system does not implement the OpenNLP name recogniser, or any of it's models. This would allow the system to detect noun phrases and resolve what they are specifically related to, whereas the current system only detects noun phrases. These would be useful to implement for the future but are not a great loss. These libraries do function well on Android, however they require large statistical models to be available in ram.

While the selected speech recognition system can recognise many different languages, and OpenNLP has models that support several major languages, the processor only supports the english language. The reference engine is specifically targeted at rules governing context and subjects that are inherent to the english language, thus cannot be easily adapted to other languages. If a new language is to be supported, it would be easier to write an entirely new engine for that language, rather than expanding the current to handle both. This is where machine learning has a significant advantage, the same system can handle both languages, it just needs additional training. Due to the open source nature of the system, additional language support could be added by developers who speak these languages, this gives even

the most niche languages the opportunity to be integrated (although in it's current configuration these languages would need to be supported by Google speech recognition).

Commands are processed separately to question resolution, but are still processed before output. The 'calculate' command is very useful for quick calculations when the user cannot focus on the device, however can only perform basic arithmetic. This is still appropriate since it can perform calculations that would prove difficult in mental arithmetic, for example division of numbers in the billions. The system imports a basic expression evaluator to parse input as equations, however this is designed for written equations rather than spoken. Spoken equations are very ambiguous and would require their own reference engine to resolve what the users intent is (for example the query 'the square root of sixteen plus nine' could mean '$\sqrt{16} + 9$' or '$\sqrt{16 + 9}$'. This difference relies more on intonation than any syntactical rule). These difficulties would take many more development hours to overcome, so the command remains in it's current state.

Output and Personification

The system returns responses to queries in real time with high intelligibility. Some results to queries have odd artefacts due to their source. For example, Wikipedia often begins articles with pronunciation or alternate spellings of words or objects. These could be removed using regular expressions, but this tends to interfere with other aspects of the articles, especially on technical subjects. The speech synthesis engine also explicitly notes quotation marks by initialising the word quote when they open and close, removing theses can fundamentally alter the intention of a sentence. In the interest of preserving the original meaning of the article these issues have been left intact. When context is given that does not align directly to an individual article the system will search the most deemed relevant article

for information related to the context. This sometimes returns fragmented sentences due to a limit set on the number of words returned. This quite often returns a more relevant query, but can return information that does not make sense. In these cases using the 'tell me more' command will return relevant information from the article.

Unlike Google Now or Siri, there is no visual output of the result. If the aim is to not distract a user, this is a positive, however digital personal assistants have many more applications than just this scenario. If the user cannot hear the auditory output, or a certain element of the output is difficult to interpret they will have to repeat the query until they can. This could result in a poor user experience. The solution is relatively simple, adding a text area on screen for the result would allow visual confirmation of the spoken output, this could also incorporate additional elements such as related images. This would force the large buttons to be diminished, nullifying their reason for being so large, and causing issues for many users. If the technical issues of background operation and wake up words can be solved (removing the need for large buttons), visual output should be considered, however not until then.

The system is personified reasonably well. Alternate outputs for the most often repeated text are randomly selected to prevent the system sounding to repetitive. Messages that convey important information do not have alternate text to prevent the message from being misunderstood (for example a network connection problem). The addition of jokes helps personify the system further. Most personal assistants are given a human name to personify them further, these also act as a useful and inconspicuous wake up word (for example 'Siri' or 'Alexa'). Due to the absence of wake up words on the current system this

has not be implemented, although all that is required is that a name is chosen for the system, rendering this issue very easy to solve in the future.

Open Source

The software developed is in full accordance with the licenses of the included middle ware. The system is open source, but should not be referred to as free software. The use of the Google speech recognition service nullifies this, however the continuous speech recognition offered by this service is unparalleled in the free software world and is fundamental to the success of the system. The system can still benefit from most of the positives of open source in that it can be collaboratively improved, however it can not be guaranteed to be completely private. Being open source would allow a developer to replace the Google recogniser with a free software recogniser if a viable alternative is developed. When evaluating the issue of freedom, the context of where the system is used must be considered. As an Android application, the system is intrinsically tied to the Android operating system which has many issues with freedom itself. If software freedom is a vital requirement for a user they might not use Android, which excludes them from the usership of the system by default.

In it's current state, the system is in no way competitive with the common alternatives. It has three services, only two of which provide real usefulness to the user. In comparison systems like Google Now and Siri have hundreds of services available. The main competitive edge the system has against these proprietary assistants are the opportunities provided by open source. The system can add any service, however niche, as long as there is a willing developer. This could attract users that other systems do not cater to. Rapid expansion of services is a possibility, but very unlikely without a dedicated community

around the system. The service could possibly become more secure than these alternatives, with open source code allowing peer review, although this is not a real consideration while the Google recogniser is still used.

The competitive edge that the system holds against other open source assistants is that all of the processing tasks can be performed in real time on the device itself. While calculations are the only useful offline service currently implemented, questions are analysed entirely on the device with primary term and context resolved very quickly. This removes the need for a network connection to offload any processing to a remote server, allowing the system to be used in a vast number of scenarios where internet connection is not possible. This is a highly competitive feature of the device, allowing use in remote locations or with limited data coverage. An issue that arises is that the only natural language parsed query in the current system retrieves data from the internet. This is only data, with no processing performed online, however this prevents the system from performing any processed tasks without a data connection. As a solution services can be added depending on the required scenario at a later date, for example an offline copy of Wikipedia could be included and queried without ever connecting to the internet, providing a convenient and comprehensive encyclopedia reference in remote locations.

Objectives

We will now evaluate the system against the individual objectives proposed at the start of development.

- To produce an open source intelligent personal assistant.

The project has produced an application that can assist in providing quick reference and calculations. This is not as comprehensive as most personal assistants, but it is still assistance and is provided to the individual user. The system is entirely open source and complies with common open source licenses. The system does implement machine learning for the application of natural language parsing, so is arguably an intelligent personal assistant. By these terms this objective is fully satisfied.

- A Language parser with:
    - Reliable question recognition.
    - Reliable primary term recognition.
    - Good recognition of context from questions.

Questions are reliably recognised and evaluated. Primary terms are sometimes confused with contextual noun phrases, however this is quite rare and only occurs in long ambiguous queries. Context recognition is arguably better than good, and is quite reliable. The only place context is often misinterpreted is when retrieving data from the internet, this is not a part of the language parser but would be disingenuous to not include in an evaluation of the system as a whole. Over all this objective is satisfied.

- Utilise a knowledge engine which provides adequate answers to questions

Wikipedia acts as the systems knowledge engine and is arguably the most comprehensive collection of user sourced data in existence. This allows for almost any subject or object to be queried and a response returned. Wikipedia articles were never designed to give quick

definitions of objects, however the introductory paragraph often does a satisfactory job. Issues do arise when context is provided that does not align directly to an individual article, but the returned data can still be arguably adequate. This objective has been satisfied.

Conclusion

Taking the full evaluation into account the system is largely successful. It accomplishes all of the initial raised objectives and has no major flaws that prevent operation. The most significant issue is the difficulty with running the system as a background service, this is the root cause of many less significant issues and should be a high priority for fixing in future developments. The system presents several significant advantages over existing systems, and could be highly competitive with services if a development community is established. The biggest obstacle to this advantage would be the lack of a plugin architecture. The system should be adapted to this before any attempt to attract additional developers. A significant success of the project is that all processing is performed on the device itself in real time, with the only compromises of an initial lengthy loading time and high memory utilisation.

Overall the system has not only achieved the established objectives, but has also made significant advances in the field of open source intelligent personal assistants.

References

Amazon (2014) *Amazon Alexa*. Amazon.com.

Apache Software Foundation (2004) *Apache License Version 2.0*. [online]. Available from: https://raw.githubusercontent.com/apache/opennlp/master/LICENSE (Accessed 5 May 2017). [online]. Available from: https://raw.githubusercontent.com/apache/opennlp/master/LICENSE (Accessed 5 May 2017).

Apache Software Foundation (2017) *OpenNLP*. Apache Software Foundation.

Apache Software Foundation (n.d.) *OpenNLP Tools Models*. [online]. Available from: http://opennlp.sourceforge.net/models-1.5/ (Accessed 5 May 2017). [online]. Available from: http://opennlp.sourceforge.net/models-1.5/ (Accessed 5 May 2017).

Apple Inc (2011) *Siri*. Aplle Inc.

Berwick, R. C. (2012) *Penn Treebank II Tags*. [online]. Available from: http://web.mit.edu/6.863/www/PennTreebankTags.html (Accessed 5 May 2017). [online]. Available from: http://web.mit.edu/6.863/www/PennTreebankTags.html (Accessed 5 May 2017).

Carnegie Mellon University (1999) *Carnegie Mellon University License*. [online]. Available from: https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/LICENSE (Accessed 5 May 2017). [online]. Available from: https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/LICENSE (Accessed 5 May 2017).

Carnegie Mellon University (2015) *CMUSphinx*. Carnegie Mellon University.

Chaudhri, V. K. et al. (2006) A Case Study in Engineering a Knowledge Base for an Intelligent Personal Assistant. *SemDesk'06 Proceedings of the 5th International Conference on Semantic Desktop and Social Semantic Collaboration - Volume 202*.

Clarity Lab (2015) *Sirius*. Clarity Lab.

Creative Commons (n.d.) *Creative Commons Attribution-ShareAlike 3.0 Unported License*. [online]. Available from: https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License (Accessed 5 May 2017). [online]. Available from: https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License (Accessed 5 May 2017).

Devi, D. V. A. et al. (2016) An Analysis on Types of Speech Recognition and Algorithms. *International Journal of Computer Science Trends and Technology (IJCST)*. 4 (2), .

Dunn, R. (2017) *eSpeak*.

Free Software Foundation, Inc (2007) *GNU General Public License Version 3*. [online]. Available from: http://espeak.sourceforge.net/license.html (Accessed 5 May 2017). [online]. Available from: http://espeak.sourceforge.net/license.html (Accessed 5 May 2017).

Goldberg, Y. (2016) A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research 57*.

Google Inc (2015a) *Android 6.0 Compatibility Definition*. [online]. Available from: http://static.googleusercontent.com/media/source.android.com/en//compatibility/6.0/android-6.0-cdd.pdf (Accessed 5 May 2017). [online]. Available from: http://static.googleusercontent.com/media/source.android.com/en//compatibility/6.0/android-6.0-cdd.pdf (Accessed 5 May 2017).

Google Inc (2015b) *Google Now*. Google Inc.

Google Inc (2017) *Google Now Screenshot* [online]. Available from: https://lh3.googleusercontent.com/NKSry3lCpv1ZPXcqqzMVJ_Oq_Man9HOzMyj71nUnclKRdRsls7HEarBtcsw-xlezkTI=h900 (Accessed 26 May 2017).

Google Inc (n.d.) *Performance Tips | Android Developers*. [online]. Available from: https://developer.android.com/training/articles/perf-tips.html (Accessed 5 May 2017). [online]. Available from: https://developer.android.com/training/articles/perf-tips.html (Accessed 5 May 2017).

Guha, R. et al. (2015) User Modeling for a Personal Assistant. *WSDM '15 Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*.

Hauswald, J., Kang, Y., et al. (2015a) DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers. *Proceedings of the 42nd Annual International Symposium on Computer Architecture*.

Hauswald, J., Laurenzano, M. A., et al. (2015b) Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*.

Java (2015) *Java SE 7 End of Public Updates Notice*. [online]. Available from: https://java.com/en/download/faq/java_7.xml (Accessed 5 May 2017). [online]. Available from: https://java.com/en/download/faq/java_7.xml (Accessed 5 May 2017).

Jennings, R. (2010) *Google buys speech synthesis firm, Phonetic Arts* [online]. Available from: http://www.computerworld.com/article/2469756/mobile-apps/google-buys-speech-synthesis-firm--phonetic-arts.html (Accessed 5 May 2017).

Jurafsky, D. & Martin, J. H. (2016) *Speech and Language Processing*.

Klimaschewski, U. (2017) *EvalEx*.

Lakhani, K. R. & Hippel, E. von (2002) How open source software works: "free" user-to-user assistance. *Research Policy 32*.

Lakhani, K. R. & Wolf, R. G. (2005) Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *Perspectives on Free and Open Source Software*.

Lemmetty, S. (1999) Review of Speech Synthesis Technology. *Helsinki University of Technology*.

Massachusetts Institute of Technology (n.d.) *MIT License*. [online]. Available from: https://raw.githubusercontent.com/uklimaschewski/EvalEx/master/LICENSE (Accessed 5 May 2017). [online]. Available from: https://raw.githubusercontent.com/uklimaschewski/EvalEx/master/LICENSE (Accessed 5 May 2017).

Microsoft (2014) *Cortana*. Microsoft.

Motorola (n.d.) *Moto G (5th Gen.) - Android Smartphone*. [online]. Available from: https://www.motorola.com/we/products/moto-g (Accessed 5 May 2017). [online]. Available from: https://www.motorola.com/we/products/moto-g (Accessed 5 May 2017).

Mycroft AI Inc (2017) *Mycroft*. Mycroft AI Inc.

Myers, K. et al. (2007) An Intelligent Personal Assistant for Task and Time Management. *AI Magazine Volume 28 Number 2*.

Nvidia (n.d.) *New SHIELD tablet K1 for Gamers - Nvidia*. [online]. Available from: https://shield.nvidia.co.uk/tablet/k1 (Accessed 5 May 2017). [online]. Available from: https://shield.nvidia.co.uk/tablet/k1 (Accessed 5 May 2017).

Opensource.com (n.d.) *The open source way*. [online]. Available from: https://opensource.com/open-source-way (Accessed 5 May 2017). [online]. Available from: https://opensource.com/open-source-way (Accessed 5 May 2017).

Samsung (n.d.) *Samsung Galaxy S8 and S8+* [online]. Available from: http://www.samsung.com/global/galaxy/galaxy-s8/ (Accessed 5 May 2017).

Santorini, B. (1990) Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision). *University of Pennsylvania Scholarly Commons*.

Stallman, R. (2011) *Is Android really free software?* [online]. Available from: http://www.theguardian.com/technology/2011/sep/19/android-free-software-stallman (Accessed 5 May 2017).

Stallman, R. (2016) *Network Services Aren't Free or Nonfree; They Raise Other Issues*. [online]. Available from: https://www.gnu.org/philosophy/network-services-arent-free-or-nonfree.en.html (Accessed 5 May 2017). [online]. Available from: https://www.gnu.org/philosophy/network-services-arent-free-or-nonfree.en.html (Accessed 5 May 2017).

Stallman, R. (2017) *What is free software?* [online]. Available from: https://www.gnu.org/philosophy/free-sw.en.html (Accessed 5 May 2017). [online]. Available from: https://www.gnu.org/philosophy/free-sw.en.html (Accessed 5 May 2017).

Stallman, R. (2009) Why Open Source misses the point of Free Software. *Communications of the ACM - One Laptop Per Child: Vision vs. Reality*.

Wileyfox (2016) *Wileyfox - What If?* [online]. Available from: https://www.wileyfox.com/ (Accessed 5 November 2016).

Appendix

Penn Treebank Tags:

Clause Level

S - simple declarative clause, i.e. one that is not introduced by a (possible empty) subordinating conjunction or a *wh*-word and that does not exhibit subject-verb inversion.

SBAR - Clause introduced by a (possibly empty) subordinating conjunction.

SBARQ - Direct question introduced by a *wh*-word or a *wh*-phrase. Indirect questions and relative clauses should be bracketed as SBAR, not SBARQ.

SINV - Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal.

SQ - Inverted yes/no question, or main clause of a *wh*-question, following the *wh*-phrase in SBARQ.

Phrase Level

ADJP - Adjective Phrase.

ADVP - Adverb Phrase.

CONJP - Conjunction Phrase.

FRAG - Fragment.

INTJ - Interjection. Corresponds approximately to the part-of-speech tag UH.

LST - List marker. Includes surrounding punctuation.

NAC - Not a Constituent; used to show the scope of certain prenominal modifiers within an

NP.

NP - Noun Phrase.

NX - Used within certain complex NPs to mark the head of the NP. Corresponds very roughly to N-bar level but used quite differently.

PP - Prepositional Phrase.

PRN - Parenthetical.

PRT - Particle. Category for words that should be tagged RP.

QP - Quantifier Phrase (i.e. complex measure/amount phrase); used within NP.

RRC - Reduced Relative Clause.

UCP - Unlike Coordinated Phrase.

VP - Vereb Phrase.

WHADJP - *Wh*-adjective Phrase. Adjectival phrase containing a *wh*-adverb, as in *how hot*.

WHAVP - *Wh*-adverb Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing a *wh*-adverb such as *how* or *why*.

WHNP - *Wh*-noun Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing some *wh*-word, e.g. *who*, *which book*, *whose daughter*, *none of which*, or *how many leopards*.

WHPP - *Wh*-prepositional Phrase. Prepositional phrase containing a *wh*-noun phrase (such as *of which* or *by whose authority*) that either introduces a PP gap or is contained by a WHNP.

X - Unknown, uncertain, or unbracketable. X is often used for bracketing typos and in bracketing *the...the*-constructions.

Word level

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 (prolog version PRP-S)

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

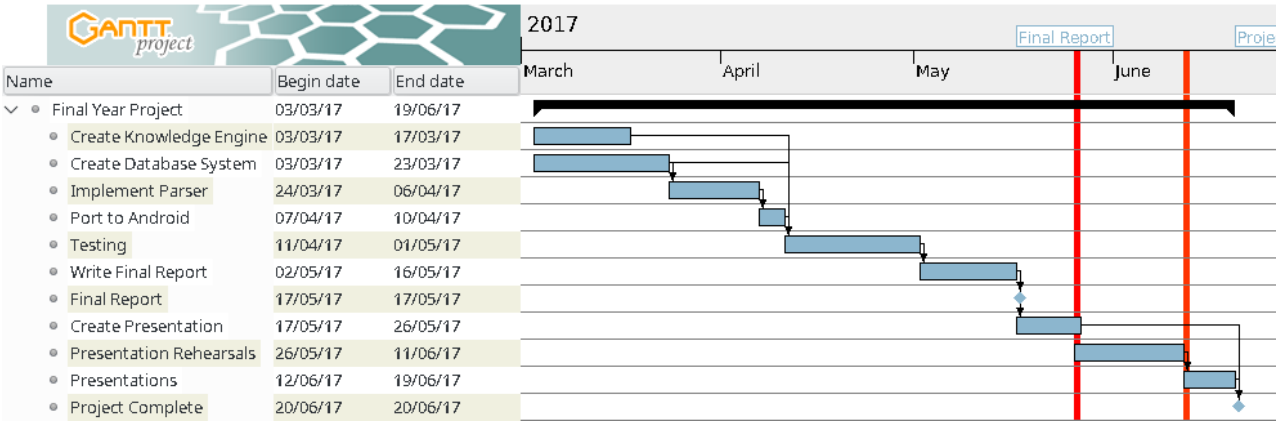
WP - Wh-pronoun

WP$ - Possessive wh-pronoun (prolog version WP-S)

WRB – Wh-adverb

(Berwick, 2012)

Gantt chart:


GANTT project
2017
Final Report
Proje
March
April
May
June
Name
Begin date
End date
Final Year Project 03/03/17 19/06/17
Create Knowledge Engine 03/03/17 17/03/17
Create Database System 03/03/17 23/03/17
Implement Parser 24/03/17 06/04/17
Port to Android 07/04/17 10/04/17
Testing 11/04/17 01/05/17
Write Final Report 02/05/17 16/05/17
Final Report 17/05/17 17/05/17
Create Presentation 17/05/17 26/05/17
Presentation Rehearsals 26/05/17 11/06/17
Presentations 12/06/17 19/06/17
Project Complete 20/06/17 20/06/17

TESTING

No.	Description	Data	Expected	Result	Action
Context and primary term resolution					
1.	Extracting the primary term from a question	‘What is Android?’	primary term: 'Android'	Questions: 'What is Android?' 'is Android' wh: 'What' Verbs: nouns: primary term: 'Android'	pass
2.	Extracting the primary term from a question with pronoun	‘Android, what is it?	primary term: 'Android'	Questions: 'What is it' wh: 'what' Verbs: nouns: primary term: 'Android'	pass
3.	Extracting the primary term from a question with context	‘On mobile, what is Android?’	primary term: 'Android' context: 'mobile'	Questions: 'what is Android' wh: 'what' Verbs: nouns: 'mobile' primary term: 'Android'	pass
4.	Extracting the primary term from a question with context	‘What is Android on mobile?’	primary term: 'Android' context: 'mobile'	Questions: 'What is Android on mobile?' 'is Android on mobile' wh: 'What' Verbs:	pass

				nouns: 'mobile' primary term: 'Android'	
5.	Extracting the primary term from a question with context and pronoun	‘Android on mobile, what is it?’	primary term: 'Android' context: 'mobile'	Questions: 'what is it' wh: 'what' Verbs: nouns: 'mobile' primary term: 'Android'	pass
6.	Extracting the primary term from a complex question	‘on mobile, there is an operating system called Android, what is it?’	primary term: 'Android' context: 'mobile' ‘operating system’	Questions: 'what is it' wh: 'what' Verbs: 'called' nouns: 'operating system' 'Android' primary term: 'mobile'	‘mobile’ is primary term. This is acceptable due to ‘Android’ being context and allowing query resolution.
7.	Extracting the primary term from a complex question	‘What is the Android operating system that is on mobile?’	primary term: 'Android operating system' context: 'mobile'	Questions: 'What is the Android operating system that is on mobile?' 'is the Android operating system that is on mobile' 'that is on mobile' wh: 'What' Verbs: nouns: 'mobile' primary term: 'Android operating system'	pass
8.	Extracting the primary term from a short statement.	‘Android.’	primary term: 'Android'	Questions: wh:	

				Verbs: nouns: primary term: 'Android.'	
Relevant data retrieval					
9.	Retrieving data extracted from the query	‘What is Android?’	‘Android is ambiguous’ is output	‘Android is ambiguous’ is output	pass
10.	Retrieving data extracted from the query	‘Android, what is it?	‘Android is ambiguous’ is output	‘Android is ambiguous’ is output	pass
11.	Retrieving data extracted from the query	‘On mobile, what is Android?’	Snippet of appropriate article.	First line of Wikipedia ‘Android (operating system) ’ entry is output	When context is applied the query usually returns a snippet. Due to ‘Android mobile’ having an exact Wikipedia entry, the first line is returned. Since this is more accurate than expected, this is a pass.
12.	Retrieving data extracted from the query	‘What is Android on mobile?’	Snippet of appropriate article.	First line of Wikipedia ‘Android (operating system) ’ entry is output	pass
13.	Retrieving data extracted from the query	‘Android on mobile, what is it?’	Snippet of appropriate article.	Snippet of Wikipedia ‘Android (operating system) ’ entry is output	pass
14.	Retrieving data extracted from the query	‘on mobile, there is an operating system called Android, what is it?’	Snippet of appropriate article.	Snippet of Wikipedia ‘Android (operating system) ’ entry is output	pass
15.	Retrieving data extracted from the query	‘What is the Android operating system that is on mobile?’	Snippet of appropriate article.	Snippet of Wikipedia ‘Android (operating system) ’ entry is output	pass
16.	Retrieving data extracted from the query	‘Android.’	‘Android is ambiguous’ is output	Snippet of Wikipedia ‘Android version history’ entry is output	This is Ambiguous and a fragment of a sentence causing the system to search for the most relevant entry. It is difficult to detect fragments accurately so the search is attempted.
17.	Retrieving data extracted from the query	‘In robotics, what is Android?’	Snippet of appropriate article	Snippet of Wikipedia ‘Android (robot) ’ entry is output	pass

18.	Retrieving data extracted from the query	‘What is Android in robotics?’	Snippet of appropriate article	Snippet of Wikipedia ‘Android (robot) ’ entry is output	pass
19.	Retrieving data extracted from the query	‘Android in robotics, what is it?’	Snippet of appropriate article	Snippet of Wikipedia ‘Android (robot) ’ entry is output	pass
20.	Retrieving data extracted from the query	‘in science fiction, there are robots called androids, what are they?’	Snippet of appropriate article	Snippet of Wikipedia ‘Android (robot) ’ entry is output	pass
21.	Handling Erroneous input	‘What is’	Rejects fragmented sentence	try another sentence message is output	pass
22.	Handling Erroneous input	‘Who is the’	Rejects fragmented sentence	Not recognised message is output	pass
23.	Offline check	‘What is Android on mobile?’	Detects there is no network access	Try checking network prompt is output	pass
24.	Offline check	‘What is Android in robotics?’	Detects there is no network access	Try checking network prompt is output	pass
Input accuracy					
25.	Simple statement recognition	‘What is Android?’	Detects ‘What is Android?’	Detects ‘What is Android?’ WER: 0	pass
26.	Simple offline statement recognition	‘What is Android?’	Detects ‘What is Android?’	Detects ‘What is android?’ WER: 0.33	Does not detect ‘Android’ as a proper noun without network access. This is still a pass
27.	Simple statement recognition with background noise	‘What is Android?’ speech sample used as background noise	Detects ‘What is Android?’	Detects ‘What is Android?’ WER: 0	pass
28.	Simple statement offline recognition with background noise	‘What is Android?’ speech sample used as background noise	Detects ‘What is Android?’	Detects ‘What is android?’ WER: 0.33	pass
29.	Uncommon noun recognition	‘Who is Luigi Boccherini?’	Detects ‘Who is Luigi Boccherini?’	Detects ‘Who is Luigi Boccherini?’ WER: 0	pass
30.	Uncommon noun offline recognition	‘Who is Luigi Boccherini?’	Does not detect ‘Who is Luigi Boccherini?’	Detects ‘Who is Luigi ocarina?’ WER: 0.33	Cannot resolve less common names without network access. This is a pass
Output					

31.	Jokes are output when prompted	‘Tell me a joke’	Tells a randomly selected joke	‘I would tell you a UDP joke, but you might not get it.’ is output	pass
32.	Jokes are output when prompted	‘Tell me a joke’ No internet connection	Tells a randomly selected joke	‘Yo Momma's so FAT. She can't handle files over 4 gigabytes.’ is output	pass
33.	Calculations are performed when requested.	‘Calculate 5 plus 4’	Answer is calculated	‘the answer is 9’ is output	pass
34.	Calculations are performed when requested.	‘Calculate 5 minus 4’	Answer is calculated	‘the answer is 1’ is output	pass
35.	Calculations are performed when requested.	‘Calculate 4 minus 5’	Answer is calculated	‘the answer is -1’ is output	pass
36.	Calculations are performed when requested.	‘Calculate 5 divided by 4’	Answer is calculated	‘the answer is 1.25’ is output	pass
37.	Calculations are performed when requested.	‘Calculate 0.5 divided by 0.4’	Answer is calculated	‘the answer is 1.25’ is output	pass
38.	Calculations are performed when requested.	‘Calculate 5 times 4’	Answer is calculated	‘the answer is 20’ is output	pass
39.	Calculations are performed when requested.	‘Calculate 1 divided by 0’	Divide by 0 is validated against	‘i cannot work that out’ is output	pass
40.	Calculations are performed when requested.	‘Calculate the square root of 25’	Square root is not supported	‘i cannot work that out’ is output	pass
41.	Calculations are performed when requested.	‘Calculate 5 billion plus 4 million’	Answer is calculated	‘the answer is 5 billion 4 million’ is output	pass
42.	Erroneous calculation	‘Calculate the Android operating system’	This is not an equation	‘i cannot work that out’ is output	pass
43.	Calculations are performed Offline.	‘Calculate 5 divided by 4’ No internet connection	Answer is calculated	‘the answer is 1.25’ is output	pass
44.	Additional information is provided on request	‘Tell me more’ previous query: none	No data is retrieved due to previous query having no data	‘I haven’t told you anything yet’ is output	pass
45.	Additional	‘Tell me more’	Repeats that	‘Android is ambiguous’	pass

	information is provided on request	previous query: ‘what is Android?’	query is too ambiguous	is output	
46.	Additional information is provided on request	‘Tell me more’ previous query: ‘What is Android on mobile?’	Retrieves more data from the last query	Introduction of Wikipedia ‘Android (operating system) ’ entry is output	pass
47.	Additional information is provided on request	‘Tell me more’ previous query: ‘What is Android on mobile?’ No internet connection	Detects there is no network access	Try checking network prompt is output	pass
Miscellaneous input					
48.	Speak button activates speech input		When pressed the Android speech prompt activates	When pressed the Android speech prompt activates	pass
49.	Shut Up button halts speech output	First line of Wikipedia ‘Android (operating system) ’ entry	When pressed the speech halts	When pressed the speech halts and ‘Shut up’ is output	pass
Garbage Collection					
50.	Life duration	Only application running Screen locked for 30 seconds	Still alive	Still alive	pass
51.	Life duration	Only application running Screen locked for 5 minutes	Still alive	Still alive	pass
52.	Life duration	Only application running Screen locked for 10 minutes	Still alive	Still alive	pass
53.	Life duration	Only application running Screen locked for 30 minutes	Collected by garbage collector	Still alive	pass
54.	Life duration	Only application running minimised for 30 seconds	Still alive	Still alive	pass
55.	Life duration	Only application running minimised for 5 minutes	Still alive	Still alive	pass
56.	Life duration	Only application running minimised for 10 minutes	Still alive	Still alive	pass
57.	Life duration	Only application running	Collected by garbage	Still alive	pass

		minimised for 30 minutes	collector		
58.	Life duration	Minimised with another application in use for 30 seconds	Still alive	Still alive	pass
59.	Life duration	Minimised with another application in use for 5 minutes	Still alive	Still alive	pass
60.	Life duration	Minimised with another application in use for 10 minutes	Collected by garbage collector	Still alive	pass
61.	Life duration	Minimised with another application in use for 30 minutes	Collected by garbage collector	Still alive	pass
62.	Life duration	Minimised with a high RAM application in use for 30 seconds	Still alive	Still alive	pass
63.	Life duration	Minimised with a high RAM application in use for 5 minutes	Collected by garbage collector	Still alive	pass
64.	Life duration	Minimised with a high RAM application in use for 10 minutes	Collected by garbage collector	Still alive	pass
65.	Life duration	Minimised with a high RAM application in use for 30 minutes	Collected by garbage collector	Still alive	pass
Loading speed					
66.	The average initial loading speed	10 launches using ADB on Wileyfox Swift	~2 minutes	1:57 2:01 1:59 1:59 2:05 1:56 1:57 1:58 2:01 1:59 Average: 1:59	pass
67.	The average initial loading speed	10 launches using ADB on Motorola Moto G (5)	Less than 2 minutes	4:40 4:37 4:41 4:42 4:40 4:40 4:41	Significantly slower on a phone with more powerful specifications. This is a failure but is possibly an issue with the device.

				4:39 4:37 4:38 Average: 4:40	
68.	The average initial loading speed	10 launches using ADB on Nvidia Shield Tablet	Less than 2 minutes	1:05 1:05 1:04 1:05 1:06 1:03 1:07 1:04 1:05 1:03 Average: 1:05	pass
RAM use					
69.	RAM use at idle	Application at idle	~250mb	242.93mb 242.91mb 242.92mb 242.95mb 242.93mb Average: 242.93mb	pass
70.	RAM use after a query	‘What is Android on mobile?’	~250mb	249.08mb 242.97mb 249.08mb 249.69mb 242.42mb Average: 246.65mb	pass
71.	RAM use after multiple queries	‘What is Android on mobile?’ ‘What is Android in robotics?’ ‘on mobile, there is an operating system called Android, what is it?’	~250mb	244.62mb 244.81mb 242.86mb 245.19mb 246.40mb Average: 244.78mb	pass
72.	RAM use at idle after multiple queries	‘What is Android on mobile?’ ‘What is Android in robotics?’ ‘on mobile, there is an operating system called Android, what is it?’ idle for 1 minute	~250mb	244.63mb 244.81mb 242.88mb 245.20mb 246.41mb Average: 244.79mb	pass

Logbook

Date	Type	Description
18/11/2016	Submission	Submitted project proposal
19/11/2016	Research	Researched artificial intelligence
20/11/2016	Research	Researched artificial intelligence
21/11/2016	Research	Researched artificial intelligence and middle ware
22/11/2016	Research	Researched middle ware
25/11/2016	Supervisor meeting	Met with supervisor to discuss research and proposal
26/11/2016	Research	Researched voice recognition
27/11/2016	Research	Researched voice recognition
28/11/2016	Research	Researched voice recognition middle ware
02/12/2016	Supervisor meeting	Met with supervisor and proposal is finalised, use of middle ware is discussed.
03/12/2016	Research	Researched Natural language processing
04/12/2016	Research	Researched Natural language processing
05/12/2016	Research	Researched Natural language processing
06/12/2016	Research	Researched Natural language processing
07/12/2016	Research	Researched Natural language processing
09/12/2016	Supervisor meeting	Met with supervisor
10/12/2016	Research	Natural language processing and middle ware
11/12/2016	Research	Natural language processing middle ware

12/12/2016	Prototyping	Began implementing voice recognition in Python
13/12/2016	Prototyping	Began implementing voice recognition in Python
14/12/2016	Prototyping	Test porting python voice recognition prototype to Android
15/12/2016	Supervisor meeting	Met with supervisor, discussed python porting issues
17/12/2016	Prototyping	Continue porting python voice recognition prototype to Android
18/12/2016	Prototyping	Continue porting python voice recognition prototype to Android
02/01/2017	Prototyping	Abandon python and begin java prototype
06/01/2017	Supervisor meeting	Met with supervisor discussed python porting issues and switch to java
07/01/2017	Research	Research Java voice recognition
08/01/2017	Prototyping	Implement voice recognition
13/01/2017	Supervisor meeting	Met with supervisor
14/01/2017	Prototyping	Implement voice recognition
15/01/2017	Prototyping	Implement voice recognition
21/01/2017	Prototyping	Test porting java prototype to Android
22/01/2017	Prototyping	Test porting java prototype to Android
23/01/2017	Prototyping	Test porting java prototype to Android
27/01/2017	Supervisor meeting	Met with supervisor, discussed interim report
28/01/2017	Writing	Began Interim report
29/01/2017	Writing	Continued Interim report
30/01/2017	Prototyping	Continued porting java prototype to Android
03/02/2017	Supervisor meeting	Met with supervisor, discussed interim report

04/02/2017	Research	Researched Natural language processing middle ware
05/02/2017	Research	Researched Natural language processing middle ware
10/02/2017	Supervisor meeting	Met with supervisor
12/02/2017	Prototyping	Test porting natural language middle ware to android
17/02/2017	Supervisor meeting	Met with supervisor, discussed interim report
18/02/2017	Writing	Continued Interim report
19/02/2017	Prototyping	Test porting natural language middle ware to android
21/02/2017	Prototyping	Optimising natural language processing for android
22/02/2017	Prototyping	Optimising natural language processing for android
23/02/2017	Writing	Finished Interim report
24/02/2017	Submission	Submitted Interim report and met with supervisor
27/02/2017	Prototyping	Begin processing text with natural language processing
28/02/2017	Prototyping	processing text with natural language processing
01/03/2017	Prototyping	Attempt merge android processing and voice recognition prototypes.
03/03/2017	Supervisor meeting	Met with supervisor
04/03/2017	Prototyping	Attempt merge android processing and voice recognition prototypes.
05/03/2017	Prototyping	Attempt to improve voice recognition
06/03/2017	Prototyping	Attempt to improve voice recognition
11/03/2017	Prototyping	Attempt training voice recognition
12/03/2017	Prototyping	Attempt training voice recognition
17/03/2017	Supervisor meeting	Met with supervisor, discussed voice recognition not being

		accurate
18/03/2017	Prototyping	Attempt implementing rigid syntax voice recognition
19/03/2017	Prototyping	Attempt implementing rigid syntax voice recognition
20/03/2017	Prototyping	Research android voice recognition
21/03/2017	Prototyping	Begin implementing Google voice recognition
25/03/2017	Prototyping	Continued implementing Google voice recognition
26/03/2017	Prototyping	Continued implementing natural language processing
28/03/2017	Prototyping	Continued implementing natural language processing
31/03/2017	Supervisor meeting	Met with supervisor and demonstrated current prototype
3/04/2017	Prototyping	Begin implementing information retrieval from internet
4/04/2017	Prototyping	Continued implementing information retrieval from internet
8/04/2017	Prototyping	Implemented voice output
9/04/2017	Prototyping	Continued implementing information retrieval from internet
10/04/2017	Prototyping	Continued implementing natural language processing
11/04/2017	Prototyping	optimised natural language processing
12/04/2017	Prototyping	optimised natural language processing
12/04/2017	Prototyping	optimised natural language processing
19/04/2017	Prototyping	Continued implementing natural language processing
20/04/2017	Prototyping	added more voice output
22/04/2017	Prototyping	Continued implementing information retrieval from internet
28/04/2017	Supervisor meeting	Met with supervisor and demonstrated current prototype, discussed Final report

29/04/2017	Prototyping	Begin implementing calculations
30/04/2017	Prototyping	Integrate calculations with voice processing
06/05/2017	Prototyping	Improved voice output
07/05/2017	Prototyping	Added jokes
08/05/2017	Prototyping	Improved information retrieved from internet
09/05/2017	Writing	Started final report
12/05/2017	Supervisor meeting	Met with supervisor discussed Final report contents
13/05/2017	Writing	Continued final report
14/05/2017	Research	Researched licenses and other info for final report
15/05/2017	Writing	Continued final report
19/05/2017	Supervisor meeting	Met with supervisor and discussed Final report layout
20/05/2017	Writing	Continued final report
21/05/2017	Writing	Continued final report
22/05/2017	Writing	Continued final report
23/05/2017	Writing	Continued final report
24/05/2017	Writing	Continued final report
25/05/2017	Writing	Continued final report

Code listing: MainActivity.java

```
package com.example.robert.opennlptest;

import android.content.Intent;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.speech.tts.TextToSpeech;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.content.Context;

import opennlp.tools.parser.AbstractBottomUpParser;
import opennlp.tools.parser.Parse;
import opennlp.tools.parser.ParserFactory;
import opennlp.tools.parser.ParserModel;
import opennlp.tools.parser.Parser;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTagger;
import opennlp.tools.postag.POSTaggerME;

import opennlp.tools.tokenize.Tokenizer;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;

import opennlp.tools.util.Span;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import java.util.Locale;

import java.net.*;

import org.w3c.dom.*;
import javax.xml.parsers.*;

public class MainActivity extends AppCompatActivity {

    TextToSpeech tts;

    static Tokenizer questionTokenizer;
    private static Parser questionParser = null;

    static{
        questionTokenizer = SetupTokenizer();
        questionParser = SetupParser();
    }
```

```
    String previousTitle = "";
    int requestNo = 0;
    RandomMessage randomMessage = new RandomMessage();

    private static Context mContext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mContext = this;

        setContentView(R.layout.activity_main);

        tts = new TextToSpeech(MainActivity.this, new TextToSpeech.OnInitListener(){
            @Override
            public void onInit(int i){
                System.out.print("Start");
            }
        });
        tts.setLanguage(Locale.UK);

    }

    public static Tokenizer SetupTokenizer()
    {
        Tokenizer newTokenizer = null;
        InputStream inputStream = null;
        try{
            inputStream = App.getContext().getResources().getAssets().open("en-token.bin");
            final TokenizerModel tokenModel = new TokenizerModel(inputStream);
            inputStream.close();

            newTokenizer = new TokenizerME(tokenModel);
        } catch (final IOException ioe) {
            ioe.printStackTrace();
        }
        return newTokenizer;
    }

    public static POSTagger SetupPOSTagger()
    {
        POSTagger newPosTagger = null;
        InputStream inputStream = null;
        try {
            inputStream = App.getContext().getResources().getAssets().open("en-pos-
maxent.zip");
            final POSModel posModel = new POSModel(inputStream);
            newPosTagger = new POSTaggerME(posModel);
            inputStream.close();

        } catch (final IOException ioe) {
            ioe.printStackTrace();
        }
        return newPosTagger;
```

```
    }

    private static final int SPEECH_REQUEST_CODE = 0;

    private void displaySpeechRecognizer() {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        startActivityForResult(intent, SPEECH_REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
                                    Intent data) {
        if (requestCode == SPEECH_REQUEST_CODE && resultCode == RESULT_OK) {
            List<String> results = data.getStringArrayListExtra(
                    RecognizerIntent.EXTRA_RESULTS);
            String spokenText = results.get(0);

            String[] spokenwords = spokenText.split(" ", 2);

            if(spokenwords.length >= 2 && spokenwords[0].equals("calculate"))
            {
                StringCalc sc = new StringCalc();
                String ans = sc.Calc(spokenwords[1]);
                if(!ans.equals(""))
                {
                    tts.speak("The answer to "+spokenwords[1]+" is: " + ans,
TextToSpeech.QUEUE_FLUSH, null);
                }
                else
                {
                    tts.speak("Sorry, I couldn't work this out", TextToSpeech.QUEUE_FLUSH,
null);
                }
            }
            else if(spokenText.equals("tell me more"))
            {
                if(!previousTitle.equals(""))
                {
                    String result = SearchWiki(previousTitle, GenerateProp());
                }
                else
                {
                    tts.speak("I haven't told you anything yet!", TextToSpeech.QUEUE_FLUSH,
null);
                }

            }
            else if(spokenText.equals("tell me a joke"))
            {
                tts.speak(randomMessage.GetJoke(), TextToSpeech.QUEUE_FLUSH, null);
            }
            else
            {
```

```
                spokenText = spokenText  + ".";
                Parse myParse = parseSentence(spokenText, questionTokenizer);
                myParse.show();
                VoiceQuery vq = new VoiceQuery(myParse);
                if(!vq.IsError())
                {
                    Log.d("term", "onActivityResult: "+vq.GenerateTerm());
                    requestNo = 0;
                    String result = SearchWiki(vq.GenerateTerm(), GenerateProp());
                }
                else
                {
                    tts.speak("I'm a little confused by that, try phrasing it as a longer
question or a more complete sentence", TextToSpeech.QUEUE_FLUSH, null);
                }
            }
        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    public String GenerateProp()
    {
        String prop = "";

        if(requestNo == 0)
        {
            prop = "&exsentences=1";
            requestNo++;
        }
        else if(requestNo == 1)
        {
            prop = "&exintro=1";
            requestNo = 0;
            //previousMessage = "";
        }
        return prop;
    }

    public void SpeechOnClick(View v)
    {
        Uri notification = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
        Ringtone r = RingtoneManager.getRingtone(getApplicationContext(), notification);
        r.play();
        displaySpeechRecognizer();
        Log.d("wiki", "SpeechOnClick: ");
    }

    public void ShutUpOnClick(View v)
    {
        //this flushes the current speech queue, halting current speech
        tts.speak("shut up", TextToSpeech.QUEUE_FLUSH, null);
    }

    public String SearchWiki(String title, String prop)
```

```
    {
        String result = "No Result";

        XMLGetter xg =(XMLGetter) new XMLGetter(new XMLGetter.AsyncResponse() {
            @Override
            public void processFinish(String title, Document output) {
                Log.d("wiki", "onPostExecute: done even more ");
                Log.d("wiki message", "processFinish:  started");
                Log.d("wiki message", "processFinish: " + (output == null));
                String message = "Sorry, somethings gone wrong";
                if(output == null)
                {
                    message = "I can't seem to access my data right now, try checking the
network.";

                }
                else
if(output.getElementsByTagName("page").item(0).getAttributes().item(3).getNodeName().equals("
missing"))
                {
                    message = "I can't find anything on "+title;
                    if(output.getElementsByTagName("p").getLength() > 0)
                    {
                        String snip =
output.getElementsByTagName("p").item(0).getAttributes().getNamedItem("snippet").getNodeValue
();
                        String strippedText = snip.replaceAll("(?s)<[^>]*>(\\s*<[^>]*>)*", "
");
                        message = randomMessage.GetNotSure() + strippedText;
                        previousTitle =
output.getElementsByTagName("p").item(0).getAttributes().getNamedItem("title").getNodeValue()
;
                        //previousMessage = "";
                        Log.d("wiki", "processFinish: adds search title to previous");
                    }
                }
                else
                {
                    try {
                        message =
output.getElementsByTagName("extract").item(0).getTextContent();
                        previousTitle =
output.getElementsByTagName("page").item(0).getAttributes().getNamedItem("title").getNodeValu
e();
                        for (int i = 0; i < output.getElementsByTagName("cl").getLength(); i+
+) {
                            if
(output.getElementsByTagName("cl").item(i).getAttributes().item(1).getTextContent().equals("C
ategory:All article disambiguation pages")) {
                                message = "please be more specific, " + title + " is
ambiguous.";
                                break;
                            }
                        }
                    }
                    catch(Exception e)
                    {
                        message = "I'm sorry, I don't recognise that!";
```

```
                    }
                }

                tts.speak(message, TextToSpeech.QUEUE_FLUSH, null);
            }
        }).execute(title, prop);

        return result;
    }

    private static Parse parseSentence(final String text, Tokenizer tokenizer) {
        final Parse myParse = new Parse(text, new Span(0, text.length()),
AbstractBottomUpParser.INC_NODE,1, 0);

        final Span[] spans = tokenizer.tokenizePos(text);

        for (int idx=0; idx < spans.length; idx++) {
            final Span span = spans[idx];
            myParse.insert(new Parse(text, span, AbstractBottomUpParser.TOK_NODE, 0, idx));
        }

        Parse newParse = parse(myParse);
        return newParse;
    }

    private static Parse parse(final Parse p) {
        return questionParser.parse(p);
    }

    private static Parser SetupParser() {
        if (questionParser == null) {
            InputStream inputStream = null;
            try {
                inputStream = App.getContext().getResources().getAssets().open("en-parser-
chunking.bin");
                final ParserModel parseModel = new ParserModel(inputStream);
                inputStream.close();

                questionParser = ParserFactory.create(parseModel);
            } catch (final IOException ioe) {
                ioe.printStackTrace();
            }
        }
        return questionParser;
    }
}

class XMLGetter extends AsyncTask<String,Integer,Void> {

    Document doc;
    String title;

    public interface AsyncResponse {
        void processFinish(String title, Document output);
    }
```

```
    public AsyncResponse delegate = null;

    public XMLGetter(AsyncResponse delegate)
    {
        this.delegate = delegate;
    }

    protected Void doInBackground(String...params){
        title = params[0];
        String urlformat = title.replace(" ","+");
        URL url;
        String prop = params[1];

        try {
            url = new URL("https://en.wikipedia.org/w/api.php?
action=query&format=xml&prop=extracts
%7Ccategories&titles="+urlformat+"&redirects=1"+prop+"&explaintext=1&list=search&utf8=1&srsea
rch="+urlformat);

            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            dbFactory.setNamespaceAware(true);
            doc = dbFactory.newDocumentBuilder().parse(url.openStream());
            Log.d("wiki", "onPostExecute: doing");
        }catch (Exception e) {
            Log.d("wiki", "onPostExecute: " + e.getMessage());
        }

        return null;
    }

    protected void onPostExecute(Void result){
        Log.d("wiki", "onPostExecute: done");
        delegate.processFinish(title, doc);
    }

}
```

Code lisitng: RandomMessage.java

```
package com.example.robert.opennlptest;

import java.util.Random;

/**
 * Created by robert on 18/05/17.
 */

public class RandomMessage {
    private String[] notsure = new String[]{
            "I'm not sure, but this could be it: ",
            "I could be mistaken, but here it is: ",
            "This is probably right: ",
            "I think this is correct: "};

    private String[] joke = new String[]{
            "Two hats are sitting on a hat rack. One says, you stay here, I'll go on a
head.",
            "Parallel lines have so much in common but it’s a shame they’ll never meet.",
            "I would tell you a UDP joke, but you might not get it.",
            "How many computer programmers does it take to change a light bulb? None, that's
a hardware problem.",
            "As a computer program, Sometimes i feel the call of the void. And I know I've
reached the point of no return.",
            "why did the functions stop talking. they had too many arguments!",
            "why don't bachelors like git? because they are afraid to commit.",
            "Yo Momma's so fat. She can't handle files over 4 gigabytes."

    };

    public String GetNotSure()
    {
        return RandIndex(notsure);
    }

    public String GetJoke()
    {
        return RandIndex(joke);
    }

    private String RandIndex(String[] array)
    {
        int rnd = new Random().nextInt(array.length);
        return array[rnd];
    }
}
```

Code listing: StringCalc.java

```
package com.example.robert.opennlptest;

import android.text.TextUtils;
import android.util.Log;

import java.math.BigDecimal;
import java.util.Arrays;
/**
 * Created by robert on 17/05/17.
 */
public class StringCalc {
    public String Calc(String sum)
    {
        Log.d("calc", "Calc: " + sum);
        String[] words = sum.split("\\s+");
        Log.d("calc", "Calc: " + Arrays.toString(words));
        boolean error = false;

        for(int i = 0; i<words.length;i++)
        {

            if(TextUtils.isDigitsOnly(words[i].replace(".","")))
            {
                Log.d("calc", "Calc: digits: "+ words[i]);
            }
            else if(words[i].equals("times")||words[i].equals("X")||words[i].equals("x")||
words[i].equals("*"))
            {
                Log.d("calc", "Calc: *: "+ words[i]);
                words[i] = "*";

            }
            else if(words[i].equals("divide")||words[i].equals("divided")||
words[i].equals("/")||words[i].equals("÷"))
            {
                Log.d("calc", "Calc: /: "+ words[i]);
                words[i] = "/";

            }
            else if(words[i].equals("plus")||words[i].equals("add")||words[i].equals("+"))
            {
                Log.d("calc", "Calc: +: "+ words[i]);
                words[i] = "+";

            }
            else if(words[i].equals("minus")||words[i].equals("subtract")||
words[i].equals("-")||words[i].equals("takeaway")||words[i].equals("take"))
            {
                Log.d("calc", "Calc: -: "+ words[i]);
                words[i] = "-";
```

```
            }
            else if(words[i].equals("million"))
            {
                Log.d("calc", "Calc: 000000: "+ words[i]);
                words[i] = "000000";
            }
            else if(words[i].equals("billion"))
            {
                Log.d("calc", "Calc: 000000000: "+ words[i]);
                words[i] = "000000000";
            }
            else if(words[i].equals("away")||words[i].equals("by")||words[i].equals(" "))
            {
                Log.d("calc", "Calc: waste: "+ words[i]);
                words[i] = "";
            }
            else
            {
                Log.d("calc", "Calc: error: "+ words[i]);
                error = true;
                break;
            }
            System.out.print("["+words[i]+"]");
            Log.d("calc", "Calc: " + error);
        }
        if(!error)
        {
            String sumformat = "";

            for(int i = 0;i<words.length;i++)
            {
                sumformat += words[i];
            }
            Log.d("calc", "Calc: " + sumformat);
            try {
                BigDecimal result = null;
                Expression expression = new Expression(sumformat);
                result = expression.eval();
                System.out.println(result);
                return result.toPlainString();
            }catch (Exception e){
                Log.d("calc", "Calc: ARITHMETIC ERROR");
                return "";
            }
        }
        else
        {
            Log.d("calc", "Calc: ERROR");
            return "";
        }
    }

}
```

Code listing: VoiceQuery.java

```
package com.example.robert.opennlptest;

import android.util.Log;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import opennlp.tools.parser.Parse;

public class VoiceQuery {

	private List<Parse> questions = new ArrayList<Parse>();
	private List<Parse> wh = new ArrayList<Parse>();
	private List<Parse> verbphrases = new ArrayList<Parse>();
	private List<Parse> nounphrases = new ArrayList<Parse>();
	private Parse primaryNP = null;
	private boolean error = false;

	public VoiceQuery(Parse p)
	{
		TreeTraverse(p);
	}

	public VoiceQuery(List<Parse> questions,List<Parse> wh, List<Parse>
verbphrases,List<Parse> nounphrases, Parse primaryNP)
	{
		this.questions = questions;
		this.wh = wh;
		this.verbphrases = verbphrases;
		this.nounphrases = nounphrases;
		this.primaryNP = primaryNP;
	}

	public List<Parse> GetQuestions()
	{
		return this.questions;
	}
	public List<Parse> GetWH()
	{
		return this.wh;
	}
	public List<Parse> GetVerbPhrases()
	{
		return this.verbphrases;
	}
	public List<Parse> GetNounPhrases()
	{
		return this.nounphrases;
```

```
	}
	public Parse GetPrimaryNP() {return this.primaryNP; }
	public boolean IsError() { return this.error; }

	public void TreeTraverse(Parse p)
	{
		try {

			questions = new ArrayList<Parse>();
			wh = new ArrayList<Parse>();
			verbphrases = new ArrayList<Parse>();
			nounphrases = new ArrayList<Parse>();
			primaryNP = null;

			Queue q = new LinkedList();
			q.add(p);
			//Traverse looking for basic chunks
			while (!q.isEmpty()) {
				Parse parse = (Parse) q.remove();
				for (int i = 0; i < parse.getChildCount(); i++) {
					Parse child = parse.getChildren()[i];
					System.out.println(child.toString() + " " +
child.getType());
					if (child.getType().equals("SQ") ||
child.getType().equals("SBARQ") || child.getType().equals("SBAR")) {
						questions.add(child);
					}
					if (child.getType().equals("WRB") ||
child.getType().equals("WP")) {
						wh.add(child);
					}
					if (child.getParent().getType().equals("VP") &&
(child.getType().equals("VB") || child.getType().equals("VBD") ||
child.getType().equals("VBN"))) {
						verbphrases.add(child);
					}
					if (child.getType().equals("NP")) {
						boolean rootNP = true;
						for (int j = 0; j < child.getChildCount(); j++)
{
							if (child.getChildren()
[j].getType().equals("NP")) {
								rootNP = false;
							}
						}
						if (rootNP) {
							System.out.println("Adding np: " +
child.toString());
							nounphrases.add(child);
						}

					}
					q.add(child);
				}
				System.out.println();
			}
```

```
                        //Traverse final question chunk looking for most significant noun
phrase
                        Queue questionq = new LinkedList();
                        if (questions.size() > 0) {
                                questionq.add(questions.get(0));
                        } else {
                                questionq.add(p);
                        }

                        List<Parse> np = new ArrayList<Parse>();

                        while (!questionq.isEmpty()) {
                                Parse parse = (Parse) questionq.remove();
                                for (int i = 0; i < parse.getChildCount(); i++) {
                                        Parse child = parse.getChildren()[i];
                                        System.out.println("questions: " + child.toString() + "
" + child.getType());
                                        if (child.getType().equals("NP")) {
                                                boolean rootNP = true;
                                                for (int j = 0; j < child.getChildCount(); j++)
{
                                                        if (child.getChildren()
[j].getType().equals("NP")) {
                                                                rootNP = false;
                                                        }
                                                }
                                                if (rootNP) {
                                                        System.out.println("Adding np: " +
child.toString());
                                                        np.add(child);
                                                }

                                        }
                                        questionq.add(child);
                                }
                        }
                        if (!np.isEmpty()) {
                                if (np.get(0).getChildren()[0].getType().equals("PRP") ||
np.get(0).getChildren()[0].getType().equals("PRP$")) {
                                        for(int i=0;i<nounphrases.size();i++)
                                        {
                                                if(!nounphrases.get(i).getChildren()
[0].getType().equals("EX") && nounphrases.get(i).getChildCount() <= 1)
                                            {
                                                        primaryNP = nounphrases.get(i);
                                      nounphrases.removeAll(Collections.singleton(primaryNP));
                                                        break;
                                                }
                                        }

                                } else {
                                        primaryNP = np.get(0);
                                        nounphrases.removeAll(Collections.singleton(primaryNP));
                                }
                        }
```

```
            for(int i = 0; i<nounphrases.size();i++)
            {
                if((nounphrases.get(i).getChildren()[0].getType().equals("DT") ||
nounphrases.get(i).getChildren()[0].getType().equals("EX") ||
nounphrases.get(i).getChildren()[0].getType().equals("PRP") ||
nounphrases.get(i).getChildren()[0].getType().equals("PRP$")) &&
nounphrases.get(i).getChildCount() <= 1)
                {
                    nounphrases.remove(i);
                    i--;
                }
                else if (nounphrases.get(i).getChildren()[0].getType().equals("DT") ||
nounphrases.get(i).getChildren()[0].getType().equals("PRP$"))
                {
                    nounphrases.get(i).remove(0);
                }

            }



                        for (int i = 0; i < questions.size(); i++) {
                                System.out.println(questions.get(i));
                        }
                        for (int i = 0; i < wh.size(); i++) {
                                System.out.println(wh.get(i));
                        }
                        for (int i = 0; i < verbphrases.size(); i++) {
                                System.out.println(verbphrases.get(i));
                        }
                        for (int i = 0; i < nounphrases.size(); i++) {
                                System.out.println(nounphrases.get(i));
                        }
                        if (primaryNP != null && primaryNP.getChildren()
[0].getType().equals("DT")) {
                                primaryNP.remove(0);
                        }
                        System.out.println("primary term: "+this.primaryNP.toString());
                }
                catch (Exception e)
                {
                        Log.d("parsing", "TreeTraverse: " + e.getMessage());
                        error = true;
                }
        }

        public String GenerateTerm()
    {
         String term = "";
         if(!error)
         {
             term = this.GetPrimaryNP().toString();
             for(int i = 0; i < this.nounphrases.size(); i++)
             {
                 term += " "+this.nounphrases.get(i).toString();
             }
```

```
        }
        else
        {
            term = "error"; //This should never really be relied upon, errors should be
handled at search time
        }
        Log.d("term", "GenerateTerm: " + term);
        return term;
    }

}
```