Council Chatbots
Technology Landscape Review

1. Introduction

The Technology Landscape Review aims to assess and summarise the leading conversational Artificial Intelligence platforms, to share information regarding their advantages and disadvantages, and to provide advice to any potential council, or group of councils, wanting to investigate how to implement chatbot technology.

This document is one of a group of reports resulting from the discovery research project "Can chatbots and AI help solve service design problems?", in collaboration with 13 English councils.

All key project deliverables outline our findings in detail - please refer to our individual reports for more focused insights and information:

- ROI Analysis and Market Summary | April 2019 | Council Chatbots | Torchbox
- Example Shared Conversational AI Architecture | April 2019 | Council Chatbots | Torchbox
- User Research Summary Report | April 2019 | Council Chatbots | Torchbox
- Case Studies | April 2019 | Council Chatbots | Torchbox
- Project Summary Report | April 2019 | Council Chatbots | Torchbox

A blog has been published by the project lead, Neil Lawrence of Oxford City Council. To read articles covering each stage of the project please visit the blog:

- https://localdigitalchatbots.github.io

2. Executive summary

Systems based on Artificial Intelligence Markup Language (AIML) require a great deal of explicit configuration in a technical language. Intent- and entity-based systems that use machine learning are the first step towards conversational Artificial Intelligence (AI) having a basic understanding of natural language, which is needed before we start to try and train a system to interact with an organisation's users. Conversational AI is essential to what is often called 'chatbot technology', where a user interacts with a chatbot (the user interface connected to a conversational AI system) from a website or app in order to gain answers, complete an action or access a service.

Natural Language Understanding (NLU) in machines is still basic but obviously necessary for a system interacting with users in a conversational manner. As a result, it is necessary to limit the domain that the user-facing chatbot and its underlying system is trying to address, in order to achieve a good user outcome. To help a user understand what this domain is, we must be explicit about the purpose of any chatbot offered to the user.

We must also carefully select use-cases where the information and backend infrastructure sending to and receiving data from the conversational AI system are sufficient to create a satisfactory outcome for the user. For these reasons, use-case selection and user research is extremely important in creating a data model for the system to use.

This technology is not new; what is new, and is both driving and enabling the move towards conversational AI, is the following:

- the availability of large amounts of cloud hardware to run it cheaply
- increased user contact across multiple channels
- the open source online data available to train these technologies

- most personal communication now being conducted in messaging and social media channels, within which chatbots have better chances to reach near-human conversation parity compared to everyday life

All the major cloud and open source providers have adopted a very similar set of technologies for their conversational AI platforms, meaning they can all be trained from a very similar data model.

The best NLU performance among the conversational AI systems analysed for this report were IBM Watson and Rasa; however, the major platforms are closely keeping step with each other, and differences in features between the best platforms are small. Important considerations where differences lie are in the ability to generate answers from written documentation without explicit configuration effort for each scenario (long-tail solutions), and concerning conversation configuration; whether the platform uses deterministic rule-based configurations, or a machine-learnt, probabilistic and story-based configuration.

The major platform styles of use and configuration are attractive to different types of users. For instance, business users may prefer IBM, many developers show a preference for Microsoft, while data scientists may be more inclined towards Rasa. This may be a major factor in selecting a combined council platform, depending on how the system is intended to be developed and maintained.

The assessed platforms also offer cloud technology, which enables the cost-per-serve[1] to be low, typically ranging from £0.01 to £0.10. Where they fall in this range is largely dependant on the complexity of domain, the level of cloud isolation, together with any enterprise scale features required.

The software costs-per-serve are generally a small portion of the overall cost[2]. The costs of developing the initial model, the investment in ongoing maintenance and training, and integrations to a wide variety of differing

[1] The cost associated with a single contact or chat session
[2] For further detail on the costs of chatbot information, please see the Return on Investment analysis

back-end systems across each council, are likely to represent the greatest portion of project cost.

Collaborating as a group of councils will enable a centralised cloud system to have greater degree of data isolation on the cloud platforms, lead to economies of scale, and provide enterprise-class features.

The overall recommendation from this report for a conversational AI platform would be

IBM Watson Assistant Plus or Premium for a public cloud hosted system

or Rasa Stack for an open source privately cloud hosted system

Contents

1. Introduction **2**

2. Executive summary **3**

3. Reviewing the history of machine-human interaction and technology **9**

3.1. The challenge of natural language 9

3.2. Teaching the machine 13

4. Use case selection **25**

4.1. Have a focused purpose 25

4.2. Be sure you can service the understood need to solve the user's problem 26

4.3. Have a way of directing the user to more suitable help when the system cannot assist 27

5. Current common limitations of NLU in conversational AI systems **28**

5.1. Multiple or conditional utterances 28

5.2. Intentions with complex implicit hierarchies 29

5.3. The "long tail" of queries 32

6. Comparison of the accuracy of short tail conversational AI systems **34**

6.1. Intent classification scores 35

6.2. Named entity recognition scores – 1 x 54 entity set 36

6.3. Named entity recognition scores – average across 8 sets of up to 20 entities 36

7. The value of the data model for a business domain **38**

8. Other important aspects to consider in a conversational AI platform **39**

8.1. API and UIs 39

8.2. Context Object 39

8.3. Visual Conversation Flow Configuration 40

8.4. Conversation flow control 40

8.5. Pre-built channel integrations 41

8.6. Supported content types 42
8.7. Easy configuration of external web services and/or cloud functions 42
8.8. Pre-Trained System Entities 43
8.9. Pre-Trained user Intentions 44
8.10. Cluster and recommend user intentions based on existing chat logs 44
8.11. Analytics Dashboard 45
8.12. Advanced Analytics 45
9. Content Management 46
10. Costs 47
10.1. Charge types 47
10.2 Comparing charge types 48
10.3 Costs by platform 49
11. Feature matrix 52
12. High-level characterisation of each platform 53
12.1. Chatfuel 53
12.2. Botkit 53
12.3. LivePerson Maven 53
12.4. Amazon Lex 54
12.5. Microsoft Luis 54
12.6. Google Dialogflow 55
12.7. Rasa 55
12.8. IBM Watson 55
13. Recommendation 57

3. Reviewing the history of machine-human interaction and technology

3.1. The challenge of natural language

Ever since the first machines to automate tasks were created, we've grappled with two problems: first, how to build them to complete the tasks we need done; second, how to instruct them to do these tasks. Both are very complex, as the number of things that could possibly be asked of a machine is enormous, and the variety of ways someone could describe what they want is equally large.

Our attempts to make systems as intuitive and simple as possible for users has been to simplify both sides of this problem:

1. To build a very clear understanding what the machine can do

Then, even if the process to accomplish that is very complex:

2. Provide a very simple way for the user to activate that process

3.1.1. Simple tactile interaction

Initially, systems were entirely mechanically connected; the turning of a crank on a pump caused water to be raised from the well.[3] A direct physical connection existed between the process and the end result. These mental connections of direct cause and effect are often our most comforting and satisfying ways to interact with machines, and are ones we've replicated in our electronic devices.

A toggle light switch[4] is simple and satisfying, despite the myriad wires behind the wall. We have an object which responds with a positive action and always

[3] First depictions of piston pumps in Europe date back to c1450
[4] The toggle light switch was invented in 1897

causes a light to spring into life faithfully and repeatedly. We place them in predictable places in our houses; they are ubiquitous, and as such, rarely is anyone disappointed by a light switch, save for the absence of one where they expect one to be.

In the 1920s we moved from the age of wired electricity, telegraph and telephone to the wireless radio. Suddenly our devices were not just talking through physical wires, they were driven by signals coming from hundreds of miles away through thin air. The ways this strange and new machine operated needed familiar and tactile devices to control it. The dial, the slider, the buttons came to be familiar controls. By pushing, turning and sliding, we could determine what controlled a complex remote system. Those controls still form the basis for a great deal of visual user interface (UI) design in computing today; the function of a visual radio button is instantly familiar.

3.1.2. Voice-activated interaction

With machines enabling us to talk to *humans* via a radio or a telephone, it prompts the question: why can't we talk to the *machine* in the same way?

The challenge is that human speech is remarkably complex and imprecise. Far from having just one function, the same word may have many functions depending on how and where it is used, and by whom. Our first solutions to this challenge came by trying to simplify the solution again - creating simple commands with hopefully simple expectations for complex tasks.

The earliest attempts at voice-activated systems were made around the 1920s with the advent of the wireless radio. "Radio Rex"[5] was a wooden dog house which, when a user said "Rex" loudly and clearly, would respond by having Rex spring forward and bark.

[5] How Radio Rex worked https://www.youtube.com/watch?v=AdUi_St-BdM

Radio Rex was a solution which simplified both sides of the control problem: it had one function and understood just one word. Or rather, it understood a noise: its "understanding" was limited to just recognising a reverberation at an approximate frequency.

This codification of language into commands with only one meaning is how we've tackled this problem in a variety of places where communication is limited or difficult. In military fields a countdown changed from five, to fiver, to avoid saying something that it is too similar to "Fire".

3.1.3. From automated response to conversational AI

We've probably all had the experience of being asked to say a particular word to an automated system on the phone. For instance, by asking us to say only "Yes" to proceed or "No" to stop, we are reducing the complexity to the equivalent of a verbal button. The system is created to assign a single possible meaning to the user's language depending on which word is said while the user is also restricted by the options the system presents to them and so increases the chance of a successful interaction.

Breaking out of an automated system to attempt conversational AI poses the challenge that, in natural - human - language, words often do not have a single meaning:

"**May**" can be modal verb, to express possibility, or ask permission, or express a wish.

Or, "**May**" can be a noun: the hawthorn blossom, or the fifth month of the year.

Or, "**May**" can be a proper noun, someone's name.

If we are looking to build a chatbot using conversational AI technogloy for a Doctor's surgery, we might well come across the sentence:

"**May** I have an appointment with Dr **May** in the first week of **May**?"

Using natural language tends to open up both sides of the control problem; the user's expectations may be different from the functionality supported, and whatever instruction the user gives might be highly ambiguous.

By encouraging a machine to interact more 'naturally' we have also raised user expectations. We're not asking the user to communicate in an unfamiliar specific programming language or using complex set of buttons; instead we're trying to mimic in a small way the way a human converses, and that invites comparison between real human conversation and the experience conversing with the machine, not just how easy our product is to use compared to our competitors.

We also use the way someone converses as a proxy for how they think. It is easy to avoid making yourself look foolish with a button, but with natural language we are inviting the user to subconsciously make a judgement not only on whether this process works or not, but about how 'smart' the system was that did it.

So, with these challenges, why attempt a conversational interface, a chatbot, at all? The desire to talk to machines in this way has never diminished. Despite all the advances in visual user experience, we see organisations and users continuing to want to reach out to converse with each other using natural language. Speaking is one of the most natural ways to instruct; it's the way we are taught to do from the earliest ages. As councils, if we can succeed in this area we will be more trusted by the public, and will be perceived as smarter than a button click could ever have been.

3.2. Teaching the machine

3.2.1. Keyword and synonym matching

In the 1960s, punch card programming, typewriters, and keyboards transformed how humans were communicating with each other and with machines. If we could now programme a computer by typing or using punch code, why couldn't it type back?

In 1966, Eliza was created as one of the first typed conversational systems that managed to create responses that seemed closer to natural language. Eliza replicated some of the aspects and questioning techniques of a client-centric psychiatrist who tries to understand, empathise, and get the patient to expand on their issue for themselves. This was once again designed to simplify both sides of the control problem:

a) The stylistic language helped give the user a very stereotyped expectation of what to expect from the conversation and

b) By reducing the sophistication of understanding and response required, Eliza needed to understand little of what the user said: only enough to combine what they said in a response to encourage the user to expand for themselves.

To do this, Eliza used keyword matching. It reacted only to certain words in a sentence and then used the parts of the sentence before or after that word (which it didn't understand or examine further) to play back to the user in the response. An example from the code would be:

"Doctor, **I dreamed** *of vampire cats*"

"Really, of vampire cats?

Only the fact that the two keywords matched, and the order of them was "**dreamed**" immediately following "**I**", are of significance to the system. The portion after dreamed is then played back in the answer.

Eliza's concept of words was very limited. It understood that certain words are synonyms of another word, and it understood simple patterns of where the words occurred in sentences. Where it was configured with synonyms, it would treat the presence of any of those synonyms in a certain place in a sentence as meaning the same thing.

It was configured to understand that

dreamt = dreamed

It would have reacted exactly as before if the user had said:

"I **dreamt** *of vampire cats*"

"Really, *of vampire cats*?

This is synonym matching.

Eliza understood about 92 words, had around 36 different classes of reactions, most with several variations, and had a little over 200 total possible responses in all. But otherwise, its understanding of words was very limited and it had no concept of a verb or a noun.

However, it proved that by making the purpose of the system clear and then carefully crafting a set of responses, that even with very limited technology, a persuasive system could be built. The total training script for Eliza is only about 400 lines long.[6]

3.2.2. Artificial Intelligence Mark-up Language

Moving beyond simple keyword matching, systems capable of more realistic interactions take a lot more training. This leads us to our next class of systems, based on Artificial Intelligence Markup Language (AIML).

[6] Eliza statistics are taken from Charles Hayden's faithful Java recreation of the original training file http://www.chayden.net/eliza/Eliza.html

AIML was an attempt to extend the original Eliza system into a codified mark-up language, and to then use this to train a much more sophisticated system: Alice. AIML allowed for more sophisticated patterns and response variations. It allowed Alice to better remember the user's position in a conversation and what they had said before. It provided programmers with better structures to configure the synonyms and patterns they needed.

Systems built in AIML can be extremely effective; Alice won the Loebner prize for the most human-like bot of the year three times in 2000, 2001 and 2004.[7] Fundamentally however Alice had no more understanding of the actual words than Eliza did. It still relied on human configuration of every pattern of words it could understand, and of every response that it could make. Given any specific domain of responses, and enough time to adapt to the expectations and wording of real users, a high-quality system can be created. Building such a system is a huge task - the last available version of Alice has 286,764 lines of AIML code.[8]

AIML is still used in award-winning systems. Mitsuku, which is based on an evolution of AIML, with added reasoning about objects and concepts, has won the Loebner prize four times, including most recently in 2018. Its writer, Steve Worswick, has been working on the Mitsuku personality since around 2005.[9]

The bots typically competing for the Loebner challenge are just trying to have the most natural conversation they can. They are trying to match a natural response to whatever you say and don't have a direction or purpose that they are trying to steer you towards. For instance, Mitsuku represents an 18-year-old girl from Leeds and will play games, or reason facts, or talk about what she likes or dislikes.

Generally, in business we aren't necessarily trying to have the most natural discussion. We instead have goals as services or pieces of information we want to steer the user towards. Business domains, backend integrations and views

[7] https://en.wikipedia.org/wiki/Loebner_Prize
[8] Alice v1.6 from the AIML Foundation https://github.com/fastcoding/aiml-en-us-foundation-alice.v1-6
[9] https://aidreams.co.uk/forum/index.php?page=Steve_Worswick_Interview_-_Loebner_2013_winner#.XK-Wr-hKiM8

about how tasks should be completed tend to vary widely between organisations. Creating systems for complex new business domains from AIML faces a number of challenges: configuration of AIML is not very intuitive - it feels much like the technical mark-up language it is, and the effort level to create a system in a new domain of understanding is large.

With AIML, the control problem remains large on both sides of the equation: the number of examples for how real users say things that we need to configure is very high. Simultaneously, the number of things that a user could expect us to say in a very open conversation is very high. The total configuration effort therefore is very high.

3.2.3. Intent and Entity Machine Learnt systems

So, we come to what is currently the most popular class of typical cloud-based systems. The need for such systems has again been driven and enabled by changing the way we use technology.

Our personal communication has changed radically in the last 15 years: the landscape has changed from most of our personal communication being done by telephone calls, to a tiny fraction now being done so. Most of our interpersonal communication is now done by SMS, messaging service or social media. However, organisations have largely been left behind in the speed of this change, busy adapting to a web-based economy while still being heavily reliant on phone contact. Organisations now face catching up with the explosion of messaging and social channels where users expect to talk to them.

At the same time our patience has diminished. The best digital customer experiences now happen in seconds. Even physical deliveries happen the same day we complete a transaction.[10] We are no longer tolerant of the complexity of the system fulfilling our needs. Our expectations of organisations have grown,

[10] Amazon Prime Now offers same-day deliveries in a variety of locations in the UK on a subrange of products

and we expect access not only on any channel, but near instantly on that channel. Simultaneously, organisations face the challenge of reducing costs.

For many organisations the only way to address all three areas (namely, the variety of channels, the desire for more rapid responses, and the associated costs of these) is to automate some of these interactions. Organisations have been increasingly looking to conversational AI to help serve that need. Using conversational AI can help to provide instant responses to certain tasks across multiple new channels, and to free up human time for those circumstances in which a computer couldn't reasonably be expected to replace human conversation.

Social and technological changes also mean we have several new factors which help us build these new types of systems.

We now have large volumes of well written, structured natural language on almost every subject available online. Wikipedia is around 3 billion words in English alone. We have an enormous, curated examples of how humans write and speak.

Users are now conversing on messaging systems where a human is reduced to a small avatar and a small passage of text. This is a simplified environment for robots, within which it is much easier for a bot to attempt to be closer to humans in behaviour.

Lastly, the cloud provides vast amounts of processing power to try and analyse this data to learn how users refer to objects in the world and how they typically express themselves.

The world's largest cloud players, such as Amazon, Google, IBM, and Microsoft, have all invested heavily in conversational AI platforms. They regard it as a pivotal application with which to try to persuade organisations to decide to move to cloud hosted systems, due to the difficulty of organisations building and scaling these systems for themselves with private processing power.

Each of these big cloud players have developed systems that use NLU systems to tackle the problem in similar ways.[11] Rather than rely on human-configured lists of synonyms or patterns, they have analysed these large repositories of online information to build a more fundamental understanding of the language upon which business users can train and configure new business systems.

By analysing every pair of words within these large corpuses, these systems can understand that because the month of March and the month of December appear in very similar places in sentences next to very similar words, that they are a similar type of word. These systems can understand based on the words preceding what word might typically come next. The same technology underlies the auto-correction on your phone, by understanding which words typically follow other words. Similarly, if given a sentence with a word removed, the system can suggest words that might likely go in that space. So:

> "I've cut my **** and I'm on my way to hospital"

The system might suggest, leg, hand, head or arm, and be able to rank these in order of likelihood based on their understanding of analysing that very large corpus of data.

These systems don't understand words as we do; they have a limited grasp of grammar and (generally) no external concept of the objects outside of the text. However, they understand the similarity between words and some of the differences in meaning of words when they appear in different parts of the sentence or paired next to other words.

In this way they us address natural language by understanding (to some extent) the difference between:

> Is doctor **May** in the house?

and

[11] Typically a Support-vector machine classifier for intent detection

May I have a biscuit?

and

I hear **May** is lovely in Portugal"

Before we even start to train and configure our system, it already has some understanding of the fundamental language, and types of words within it, upon which to build through training. Most of the systems have contributed to this pre-built recognition of commonly needed entities, such as such as dates, people, organisations, and other typical important parts of speech. Some will also use these techniques to deal with different inflections of words, preventing us from having to deal with different manual configurations of all the variations, by using stemming or lemmatisation. In this way, such systems understand that **sing**, **sang** and **sung** all belong to the same root verb.

With millions of examples of utterances available in the base language, these systems are then able to ask us to provide far fewer utterances that a user might use to express themselves in our specific business domain. Typically, these are examples of what a user might say to express what their intention is in the moment of the conversation, and within that utterance, to mark what the significant entities are involved that they want to do that thing with.

So, for our imaginary doctor booking system, we might give examples of a user's intention to:

"Book an appointment"

I need to come and see **Dr May** urgently

Are there any appointments available **today**?

Can Mr Kamath look at my knee any time this **week**?

"Find opening times."

When is the surgery open **today**?

On **Saturdays** what time do you close?

Can you give me your opening times please?

Where we've noted that

Today, **week**, and **Saturday**

are ***dates*** that the answers of both intentions need to be specific about,

Dr May, and **Mr Kamath**

are ***practitioners*** at the surgery that an appointment needs to be with.

Instead of asking us to configure a rigid pattern of words that a user's utterance must fit exactly to in AIML, any utterance a user says can be compared against the base language model and the examples that we previously gave the system. The system can then tell us which set of examples the utterance is *most similar to*, even if the pattern of words is very different, or the actual words are different from the ones in the example but are ones that the base language model understands to be similar.

So, if our actual user comes in and says

"I must have an appointment on **Monday**"

The system will be able to say that, even though both intentions mention days in, this intention is much more likely to be "booking an appointment" than "finding opening times".

This is intent classification.

The system will also know that even though Monday wasn't configured in the examples, that this is the date for the appointment and that the user has not specified a particular doctor with whom to make an appointment.

This is “named entity recognition” (NER) or entity extraction.

These systems make all their decisions about what a user means through a combination of these two objects, intents and entities, that are trained by annotated example. Every user utterance must be understood by training these two objects. Even simple confirmations or greeting intentions need to be given domain specific examples to ensure they work with the other intentions within the set.

Intention or intent training example

An intention or intent is something that a user intends to accomplish through a specific utterance. This is evaluated by the overall combination of all the words and the position of the words in an utterance against the different examples for each intention in the training.

The closest matching intent will then be returned, together with a system confidence level. Intentions are often given a leading # to mark them.

For example, a training set might be:

#Book_Appointment

“I want to see **Dr Kamath** on **Tuesday**”

“I need an appointment tomorrow”

“Can I book an appointment **next week** with the **nurse** for a blood test.”

#Affirmative_Yes

“Yep”

“Yas”

“Affirmative”

#Emergency

"Help I need an ambulance"

"Emergency"

"Help me please I need a doctor"

A domain of knowledge is described by the total number of intentions within it and the examples making up those intentions.

Modern systems require between 5-15 examples for each intention to have a basic level of training. 30 example intentions is typical for initial alphas, with 100+ required for production launches, and maybe as many as 1000+ after optimisation once in production.

Entities training example

Entities are things which a user intends to accomplish that action *with* or *on*. These are evaluated by looking for particular words from a training set (synonym based), or words that are like particular words in a training set and used in similar positions (contextually trained).

These are often given a leading @ to mark them, and they are typically annotated within the intention examples. So here marked in bold above are examples of entities and their positions in user utterances for

@practitioner = "Dr Kamath", "Nurse"

@date = "next week", "Tuesday"

User utterance intent classification and extraction example

If a user said

"Are there any appointments with Dr Luke available a week Wednesday"

The system would compare this utterance to the domain training sets and its base understanding of the natural language, and give the most likely intention classification from the three intentions it understands in this domain, along with

any entities it is able to extract related to that intention. The most likely classification for this utterance is

#Book_Appointment

With entities of

@date: “next Wednesday”

@practitioner:“Dr Luke”

In a rule-based system, a response can then be configured which does something similar to

Condition:

If intention = #Book_Appointment

and @date is present

and @practioner is present

Action: Book appointment for @practitioner on @date

Say: “I’ve booked you in with @practitioner on @date”

Some systems have ways of reducing the configuration effort by prompting the user for entities that are needed for a given intention, in case the user has not mentioned them, or the system has not detected them. This is typically referred to as “slot filling” - namely, identifying the necessary entities to complete a slot necessary to perform and intended action.

Otherwise typical conversational AI systems require a condition to be matched for every combination of intention, entity and conversational context that it is wished for the system to address.

3.2.4. The benefits and limitations of intent and entity systems

Intent and entity systems greatly reduce the number of examples we need to provide to adequately train the system and they have helped greatly with one side of the control problem: understanding the user.

However, given a very open conversation, the number of things that users might expect us to respond with is very high. The other side of the control problem is largely unaddressed by these techniques: we still must build a satisfactory response for the things that the user wants to achieve, and a reasonable expectation of what those are in order to create a good user experience.

As a result, use case selection and user research are extremely important in this field.

4. Use case selection

The current sophistication of conversational AI requires us to still select carefully the cases we try and address. While modern intent and entity classification systems help greatly with reducing the configuration level to understand the user, generally they do little to reduce the effort in configuring how to respond to the user. To create a satisfactory outcome, many answers must be known by the system or complex actions performed across a wide variety of backend systems. These answers are normally quite different between organisations, and the systems that need actions taken on vary in use, purpose and technology.

So, while modern intent and entity systems make it quicker to train the natural language for a domain, the effort to configure the response to any user intention is largely the same as for an AIML based system.

To improve the user interaction with any conversational AI system, a number of tactics need to be employed.

4.1. Have a focused purpose

To reduce the total number of things a user can expect of the system to respond to, we need to be very clear on what the system is meant to achieve. If the system is clearly trying to address a specific area like “Recycling queries” or “Road problem reporting”, and the chatbot is clearly positioned as such to the end user, then the system can focus on a narrower set of possible utterances.

Working with narrower domains of knowledge produces a better conversational model and fewer ambiguities of meaning. Furthermore, the variety of off-topic or ancillary functions needing to be supported can be greatly reduced, as we are more able to signpost the user back to the core functionality.

Narrowing the domain and clearly labelling the chatbot's intended use, and what it can be expected to do (or not do) is crucial for both a satisfactory user experience and also ensuring we can build a system within a reasonable amount of time and effort.

4.2. Be sure you can service the understood need to solve the user's problem

Within a limited domain each answer still needs to be satisfactory to the end user. If a user wants to know opening times, then all that is required is a configuration of the opening times in the system response. However, if the user wants to book an appointment, the system will need access to the appointment booking system and permission to make changes to provide the user with a satisfactory outcome.

One way to consider the suitability of a task for conversational AI automation is to try to think of what information and system a human user would need to solve the problem. A conversational AI system knows only the documents it's given and the systems it's connected to. It can't turn around to ask a colleague, it can't use some wider experience beyond its training, and it's not free to speculate or guess. If a human user could only use the information within certain set documents, and only specific systems, even if they understood the user perfectly, could they deliver a positive result for the user?

If a human couldn't solve the problem within those limitations, then the system will not be able to either, and this use-case is almost certain to create a poor user experience and should be avoided.

4.3. Have a way of directing the user to more suitable help when the system cannot assist

Conversational AI systems typically rely very heavily on the information contained in backend systems, and the typical processes demanded or suggested by an organisation. In any domain there will be situations where the standard process does not help the user, where the situation is more complex than the organisation typically expects, or where the user has specific needs that the system can't address. In these cases, it's important that the conversational AI system has a route to pass the user onto a channel that can assist when it can't.

When selecting a use-case, picking a domain of knowledge where a reasonably high proportion of the user intentions can be handled by the system is important for maintaining the overall user experience. This analysis formed a key step in our methodology of considering the most appropriate use-cases for a chatbot within council services.[12] A chatbot where most user intentions can't be handled is likely to be regarded as highly unsatisfactory, even if it can handle a minority of user intentions very well.

For these reasons, to build a system in a reasonable amount of time and effort, selecting an appropriate use-case is as equally important as selecting an appropriate technology.

[12] Refer to the Return on Investment analysis for further details of use case considerations within the context of council chatbots

5. Current common limitations of NLU in conversational AI systems

All the major cloud players have selected an intent and entity based model for their systems. While accuracy levels tend to differ somewhat depending on the volume of training, the domain of knowledge, and the style of use, they are largely similar in underlying technologies and share similar strengths and weaknesses.

5.1. Multiple or conditional utterances

Currently these systems tend to find user utterances containing multiple intentions or conditional intentions difficult to classify.

It should be noted that humans also find these situations difficult to reliably classify. While there are a number of practical ways good conversational design can help these systems to deal with these situations, these systems don't generally have good out-of-the-box solutions for dealing with this sort of ambiguity.

5.1.1. Conditional intention utterance example

A conditional utterance is where the overall intention is dependent only on the other part of the utterance occuring:

> "If the doctor doesn't see me this week, I'm going to report her to the BMA!"

With typical domain training these systems are likely to be confused between "Book an appointment" or "Deal with a complaint".

Good conversational design can provide for standard ways to deal with the priority of different intentions and disambiguating between two very likely intentions.

5.1.2. Multiple intention utterance example

A multiple intention utterance is where a user has a number of things they want to accomplish and lists them all in one utterance.

> "I want to book an appointment for my blood test, then find out when you close today because I need to collect my prescription, oh and I need another appointment for a scan next week?"

In this example, with typical training conversational AI systems are likely to be confused between intentions:

> Book an appointment for a blood test or a scan? Is it this week or next?
>
> Give opening hours?
>
> Advise on prescription collection?
>
> Or all of these?
>
> and if so, in which order?

Good conversational design can encourage users to speak in a way which helps the system. It can also select the best intention with which to start helping the user, and then make a placeholder to come back to a secondary item from the list.

5.2. Intentions with complex implicit hierarchies

These systems classify the likelihood of an utterance having one meaning or another just based on the words and their position. They don't make underlying judgements on the relative importance of one classification over another in making their decision. So, if we train the system on example utterances for intentions like:

Book an Appointment:

"I need an appointment; now what time is a doctor available?"

(and other similar examples)

and

Emergency Request:

"Help I need a doctor urgently!"

(With other examples containing of "help" or "emergency" frequently)

The system might struggle to classify between the two intentions when a user says:

"I need a doctor now! No time for an appointment."

This contains many of the same words and arrangements of words as both intentions training examples. It also has additional words which cause confusion between the two examples, and it lacks some of the key words which we have trained to show the importance of the emergency request intention, like "Help" or "Emergency".

In this case, a human understands implicitly that the impact of misclassifying the utterance as the non-emergency situation is high, and if needing to make a choice between them will probably treat it as an emergency situation even if the words are more like the non-emergency example.

Conversational AI systems don't by default possess the concept of the possible impacts of the classifications, and so if we wish the system to consider these factors we must explicitly overlay them in our configuration. For instance, given the choice above between an emergency or a non-emergency classification, a

system could be trained to always treat close classification decisions as an emergency until it is clear this isn't the case.

These implicit hierarchies of intents are often strongly felt within organisations but weakly articulated. As the number of different user intentions within a domain grows, the similarity between each intent becomes greater, and this implicit priority becomes more significant and harder to manage.

In these cases, often humans within the same organisation will start to disagree over the correct reaction or interpretation of a user utterance which limits the overall accuracy the model can achieve.

The ability both of humans to agree in interpretation, and of the system's limited ability to consider factors other than language, often become limiting factors on the overall accuracy of the system. This tends to put an upper limit on the number of intentions it's practical to try and train within any one domain using this sort of technology.

5.3. The “long tail” of queries

Despite great improvements in the understanding of language, and the number of pre-trained entities and user intentions available, the effort in configuring an intent and entity based AI system for any given domain is high. That effort rises non-linearly as the number of user intentions that the system is trained to recognise within a domain increases.

As a rule of thumb, most systems built with these tools support 10-50 user intentions within a knowledge domain. Domains of knowledge with 50-500 intentions are often built in large production systems with a much greater amount of effort.

Systems exceeding 500 intentions are built with these technologies, but there tends to be a practical limit somewhere above this where the cost/benefit equation becomes unviable. This exact point varies with domain and approach, but somewhere in the range 600-1200 intentions would typically become impracticable.

Unfortunately, the total number of possible “intentions” a user could have for a given interaction with an organisation can typically be much higher than this practical limit. Some of those intentions will happen thousands or millions of times a year, and some will happen only once or twice. It is not currently practical to train a conversational AI system explicitly for those situations which only happen infrequently. The queries that happen regularly and are key targets for intent and entity conversational AI systems and are often termed the “short tail”. The queries which happen infrequently are termed the “long tail” and require a different sort of technology to serve them.

Long tail targeted systems are not trained with a specific list of actions they can carry out; instead they are trained on strategies for analysing the user input, searching a corpus of information for a possible answer, and then presenting a list of possible documents for the user to read further, alongside a snippet of evidence that indicates why it’s been shown. If the answer to the user’s problem

is somewhere within the documentation then there is a chance the system will present it, even if it has never had an example of when to present this data before. The correct answer might not perhaps be in the first or second position, but it might be in the list.

Hence the costs for training a long tail system tend to be initially higher, but scale in a much more sustainable way. Instead of training and integrating each new possible behaviour, any new answer added to the corpus of information will become immediately available to be found. Well-worded, infrequent requests are as likely to be matched as frequent requests if they closely resemble a document within the corpus

The most familiar interaction most users will have with a long tail problem is a search like Google. We're used to trying a search term, scanning 5-10 things we might want to examine further and if not satisfied altering our search and trying again.

That feels very different however to talking to a conversational bot that will try and clarify what we want to do and then help us directly achieve it. So not only is the underlying system and technology different, the presentation of the information in the conversation also needs to be different.

One of the major features being actively incorporated or worked on by the major cloud players is the ability to build hybrid systems, which have short-tail behaviour to help users complete frequent tasks and provide high certainty answers, but then have long-tail like behaviour to provide multiple possible answers generated automatically from supporting documentation to infrequent tasks.

6. Comparison of the accuracy of short tail conversational AI systems

From the requirements identified throughout this research project, the most appropriate type of system to consider would be based on intent and entity based machine learning. This review was asked to compare eight systems. The following have been selected, based off their initial appropriateness for consideration:

- Chatfuel
- Botkit
- LivePerson Maven
- Amazon Lex
- Microsoft LUIS
- Google DialogFlow
- Rasa
- IBM Watson

Chatfuel is the only keyword-based system within this review, and can be extended by connecting it to Google Dialogflow. Botkit has no inbuilt NLU but similarly can be connected to any of the major intent and entity based platforms and defaults to using Microsoft LUIS.

Heriot Watt University conducted a thorough study (Liu et al 2019) of four of the major conversational AI platforms from this list[13] from IBM Watson, Rasa, Microsoft Luis, and Google DialogFlow. The study used a crowd sourced set of more than 64 intents and 11,000 utterances across a variety of common virtual

[13] Xingkun Liu, Arash Eshghi, Pawel Swietojanski and Verena Rieser. "Benchmarking Natural Language Understanding Services for building Conversational Agents." Tenth International Workshop on Spoken Dialogue Systems Technology (IWSDS) 2019.

agent scenarios, such as instructing a machine to make a calendar reminder, or play a piece of music, as a way of training and testing each system.

IBM Watson was found to have leading intent recognition scores for both precision, recall and combined F1 score (explained below).

6.1. Intent classification scores

This test examines how well the systems differentiate between different user intentions within a domain.

System	Precision	Recall	Combined F1 Score
IBM Watson	0.884	0.881	0.882
Google DialogFlow	0.870	0.859	0.864
Rasa	0.863	0.863	0.863
Microsoft LUIS	0.855	0.855	0.855

Precision measures the proportion of intentions that are correctly classified. A high precision number means that the system returned few incorrect classifications for the number of correct classifications it returned.

Recall measures how many correct intentions are classified compared to the total number of intentions that could have been classified. A high recall number means that the system correctly classified most of the intentions.

The **combined F1 score** combines these two numbers into an overall figure[14]

For all the measures, higher is better, up to a maximum of 1.

Between these major platforms it can be seen that the scores for intention classification vary by about only about 3-4% from the best to the worst performer.

[14] F1 score is the harmonic mean of precision and recall

A separate test in the study looked at named entity recognition using contextual training on the same systems using the same 11,000 utterance training set annotated for 54 types of entities. IBM Watson was not included in the contextual training tests, but run in synonym only mode which does not produce comparable results.

6.2. Named entity recognition scores – 1 x 54 entity set

Using the same data set as the intention test, this test examines how well the systems can extract the name entities relevant to that intention from what the user has said.

System	Precision	Recall	Combined F1 Score
Microsoft LUIS	0.837	0.725	0.777
Rasa	0.859	0.684	0.768
Google DialogFlow	0.782	0.709	0.743

To provide a result for the IBM Watson contextual recognition training, the original corpus of 11,000 annotated user utterances was re-used to split the 54 entity set into a series of tests for each of the scenarios, with a maximum of 20 entity types in each scenario.[15] This was rerun for Google DialogFlow and IBM Watson to give a two system comparison on this somewhat simplified dataset.

6.3. Named entity recognition scores – average across 8 sets of up to 20 entities

System	Precision	Recall	Combined F1 Score
IBM Watson	0.8773	0.9117	0.8935
Google DialogFlow	0.7203	0.8921	0.7943

[15] 20 entity types is the maximum number supported by IBM Watson on the standard plan. Higher limits are available on the Plus and Premium plans.

From these early results IBM Watson seems to be a very good performer for Named Entity Recognition, and we look forward to the publication of the revised test set across all four systems.

7. The value of the data model for a business domain

These modern intent and entity based systems under examination are trained in similar ways, with a set of classified user utterances annotated for significant entities. For any given business domain, these utterances need to be matched to a set of actions and information that is useful for the user. Creating this initial model, then testing and evolving it with real users in a live situation, requires intensive effort. However, when created and optimised, it can be relatively easily used to train any of the major systems.

The methods for testing the accuracy of these models are well established. A created model for a user domain can be stored as a training set and a test set, alongside measure precision and recall scores, as seen in the accuracy tests in the previous section. This can be stored as an open source asset in its own right and provides a useful standard for any organisation tackling this domain of knowledge, regardless of the technology chosen for implementation.

Selecting the right use cases, and creating a quality model that serves that use case, will allow organisations to select their preferred technology and channel

8. Other important aspects to consider in a conversational AI platform

8.1. API and UIs

Ideally a conversational AI platform should have strong User Interface (UI) tools to help train and update the system, and have full Application Programming Interfaces (APIs) support to allow development to be automated as part of a DevOps pipeline.

Some systems, such as Chatfuel, have a heavy focus on configuration via the UI, and provide little in the way of API support. This makes them very quick and easy for non-technical users to create flows but are ultimately limited in being able to programmatically create and maintain a wide variety of bot functions.

Other systems like Rasa NLU/Core have a heavy focus on configuration by file and via API. This makes them less approachable for non-technical users to assist in training and iterating the bot.

The best systems support all features through both the UI and the APIs.

8.2. Context Object

As well as intent and entities, a context object allows the system to keep track of objects discussed within the conversation, other information about the users situation, and where the conversation is up to. The design of this context object is crucial to building a conversation which feels natural and allows the user to refer to items earlier in the conversation, and to switch the subject during the conversation. Any conversational AI platform needs to support a flexible context object which is tracked and logged throughout the conversation and preferably can support complex object types.

8.3. Visual Conversation Flow Configuration

As well as understanding the user's intention, and having a response to aim for, it's important to be able to trace how the flow of the conversation will progress to that goal.

Some tools provide ways to configure visually the conversation flow. This can be very useful to help quickly design and understand conversational flows and involve less technical members in their design.

Other platforms rely only on configuration file or code-based creation of rule or story based flows

8.4. Conversation flow control

There are two major methods for controlling the flow of a conversation:

8.4.1. Rule based flow configuration

By considering the position in the conversation flow, the state of the context object, and the user's last utterance in terms of intents and extracted entities, a set of rules should define how the bot responds. These rules can be configured in a rule engine, or in code, and may have a visual tool to help design them

The advantages of this method are that it is straightforward to create predictable functionality quickly, and it works well with agile development methods to incrementally build new features with associated tests. In complex systems, these rule sets can become large and difficult to manage. In scenarios where there is little existing data showing how users interact with this sort of conversation, this is normally the most practical way to quickly create a system that users can then start to react to, and to update the model over time. Initial systems tend to have coarse sets of rules and need to be iterated after user reactions; rule sets quickly need to grow to create natural behaviour. If a large volume of data is available describing how users already talk in this business situation, creating the bot rule set from can be effort intensive and lead to a complex initial rule set.

All the platforms aside from Rasa fall into this category, though they differ in their exact implementations of these sort of rules systems

8.4.2. Machine Learnt Story Based

Instead of explicitly configuring rules which determine the flow at each conversational turn, "stories" are used to give example conversations (both good and bad), and a probabilistic model is machine learnt to control the flow of the conversation. This can create elaborate functionality from large volumes of existing data, and is designed to more maintainable in the long run. However, the volume of stories required for training can quickly become large, and creating fine grained control of certain behaviour or investigating odd behaviour can be more difficult. Some configuration like slot filling (ensuring that all the entities for an intent are present, and prompting the user for any that are missing) may need many example stories to create something that would be much more quickly done with a reusable rule policy. To combat this, systems may use a hybrid approach of configured rules for forms or slot filling, combined with a machine learnt story approach.

Rasa is the only platform in the review which currently uses probabilistic machine learnt stories.

8.4.3. In the near future

Most of the major platforms are designing new ways in which the weight of configuration of rules or stories necessary to configure common conversational structures can be reduced. The next iterations of cloud platforms are likely to automatically iterate and learn details such as the best order or possibly phrasing to ask multiple required questions for a desired goal.

8.5. Pre-built channel integrations

Having a conversational platform that supports your target channel out-of-the-box can substantially speeds-up delivery of a solution. Pre-built integrations also tend to help ensure that the internal message model within the

conversational platform matches easily the message model of the external systems.

Other pre-built integrations that may be useful are long tail solutions, Robotic Process Automation (RPA) solutions (which automate manual human tasks on legacy backend systems), Customer Relationship Management (CRM) solutions (that may contain useful information about a customer and their prior interactions), or ticket-tracking systems (which may allow a bot to start manual processes involving both human and automated systems or get updates on issues already in flight).

8.6. Supported content types

Whilst the focus of a conversational AI platform is understanding pure text, human usage of messaging systems tends to involve a wide variety of other content, such as buttons, options, videos, emojis, gifs, URLs and images. Having a conversational platform which supports the configuration of these into its internal model can substantially improve the user experience and avoid the need to design complex objects externally. It also helps ensure consistent behaviour across multiple channels that the platform supports

If possible, conversational AI systems should also understand these inputs via the inbound channel. Typically, this is limited only to emojis, buttons and options. But visual image recognition can be integrated in most of the major cloud platforms in order to understand images which can be very useful in certain scenarios - for instance, to read a number plate or verify a document. In the future, automatic processing of images into usable data for the conversation may become an important differentiating feature.

8.7. Easy configuration of external web services and/or cloud functions

Bot responses can be enhanced by integrating information from the user with information from internal or external web services.

Common external APIs such as Maps, Weather, News or Public transport can greatly enhance a bot's ability to understand the general situation and respond in a tailored way to the user. Internal organisation APIs may allow the bot to understand a customer's status, orders or account balance, and conduct a wide variety of actions on behalf of the user

Ways to easily configure the call of a web service from within the conversation configuration are very useful features within these platforms.

This data may be returned from the web service in complex formats that need sophisticated manipulation, or a response may need to be constructed across information from a variety of APIs. Alternatively, the user may say something that is in a complex format, for instance a list of items with possible duplications. As well as the ability to call an external webservice, it is also very useful to have the option to break out of the conversational configuration language and do a small amount of traditional programmatic processing of data to transform the information in context to support the next response. The ability to call by simple configuration a serverless cloud function to perform this in your language of choice is similarly very helpful to building bot conversations.

8.8. Pre-Trained System Entities

Whilst a project could train any entity type from scratch, there are some entity types which are re-used across many scenarios, such as dates, people, numbers, currencies or places. Users may also say these items in a wide variety of complex ways such as,

> "Can I see **May** a **week next Tuesday**?" (@person: Dr May, @date: 2019-04-16)
>
> "Up my order to **10k** this week **Dave"** (@number: 10,000, person: Dave)

In which case the entity requires enrichment and processing to format the data into the most useful way for a bot response.

The best systems have a variety of sophisticated pre-trained entity types which deal with complex variations and have been well optimised to spot different sort of examples within user utterances without further custom training.

8.9. Pre-Trained user Intentions

Similarly, a user could train any intents from scratch, but as user interactions with bots become more common, there is an establishing variety of functionalities that users expect from bots. The best platforms provide a catalogue of pre-trained or "built in" content which the organisation can use to accelerate the training of its own scenarios, or use via configuration along-side their own training.

8.10. Cluster and recommend user intentions based on existing chat logs

Another feature that can speed development of a solution is the ability to ingest large volumes of existing chat data, and cluster the user utterances into suggestions for user intentions ready for training. Google via Chatbase provides this functionality and IBM's Watson Assistant provides this natively.

8.11. Analytics Dashboard

To aid in continuously improving the system once initially launched, the conversational tools should provide a dashboard of the user conversations. Typical functions include number of conversations and number of user utterances per conversation. Better systems will give statistics on responses that have weak or strong training, or good bad user reactions, or give visualisations of the distribution of routes through the system that users are following.

8.12. Advanced Analytics

Continuous improvement of a complex system when live often involves much more complex reporting and analysis than an analytics dashboard provides. The best systems provide their logs in an accessible way that can be provided to a more advanced analytical platform. This would let users create and track their own metrics for different conversational flows success rates, produce industry standard measurements such as precision and recall, and produce advanced visualisations of things such as conversational flows and confusion matrices (large matrices which show which intentions are most often confused with each other).

9. Content Management

All of these systems require configuring responses for the user. These will be templated responses in the system's internal format, with variables to be filled from the conversation, which will then be rendered by the appropriate channel.

While the internal format may differ from system to system, all share the fact that they don't explicitly call this a content management function, and typically it is a weakly supported ancillary function to the core training of the system.

If a system is to be used across multiple languages, or across multiple organisations based on a shared investment in training, it's strongly recommended to replace the internal content management processes in the chosen conversational AI system with an external content management system. Having an external content management system allows the chatbot's responses to the user to be presented in the best format for the user's language, organisation and device format, and to be consistent across many different channels.

Modern "headless" content management systems (a CMS where the content models and content are updated by APIs as well as via the UI) can be quickly adapted to work with chatbot utterance,s and can help ensure that the answer provided across channels is consistent.

Full text search of chatbot responses, version control, editorial control and duplication or similarity identification, are all desirable functions when considering an external CMS.

10. Costs

Most providers offer a free tier suitable for experimentation or personal use, but these in general would not be suitable for a production solution.

Listed below are the costs as publicly published for the paid-for plans suitable for enterprise use in a shared public cloud environment.

Most of these providers also offer enhanced versions of the service with enterprise-focused features, such as with segregated or private cloud installations, multiple high availability regions, and support for large numbers of environments.

These providers will have enterprise facing sales organisations and negotiate on price based on requirements and volume. Depending on the organisation's other cloud requirements, there may be advantages for negotiating for NLU services alongside a wider cloud package.

The costs for paying for the cloud hosting and cloud NLU solutions are generally small compared to the costs in services and maintenance to create a domain specific bot that they then have to continuously maintain, and supervise the machine learning in production.

Cloud based providers charge in a variety of ways, detailed below:

- per API call
- per conversation or per daily active user
- per active monthly user

10.1. Charge types

10.1.1. Per API call

A charge is made for the response to each user utterance.

10.1.2 Per Conversation or per daily active user

A charge is made for a single conversation, regardless of the number of utterances within it. Each system will define how a conversation is started and ended; some consider it to be any contact within a day, so a single charge is made for a user establishing a conversation on a day regardless of how many user utterances are analysed within that day.

10.1.3 Per active monthly user

A charge is made for each user who contacts the organisation any time within the month. The charge covers any number of conversations containing any number of analysed utterances for that user. Typically, these subscriptions operate in tiers, for instance up to X,000 monthly active users.

This can make it difficult to compare prices across providers, and different billing schemes may be more or less advantageous for organisations with different contact profiles or conversation lengths.

10.2 Comparing charge types

To aid comparison, the following "rules of thumb" are useful:

A typical bot conversation contains 8-10 user utterances.

A typical user contacts the organisation one or two times a year, or alternatively, the monthly contact volume is approximately 10% of the total user volume.

So, for a system with 100,000 users, 10,000 monthly subscriber contacts could be expected, which would generate around 100,000 API calls/month.

For an organisation purchasing and hosting a professional bot system, together with associated cloud hosting for monitoring, analytics or integrations, supporting multiple environments in shared public cloud, with an agreed

support contract and SLAs for a single use case, should cost in the region of £2-10k a year.

For a larger organisation with requirements to create and maintain a wide variety of internal and external bots, with much higher usage requirements, more enterprise-like requirements, for instance requiring greater levels of data isolation or multi region high availability, budgeting should start from around £100k a year.

Organisations requiring fully private solutions, having very specific data requirements, or having very high usage and performance requirements, should budget from £250k to multi-million a year.

10.3 Costs by platform

10.3.1. Chatfuel

Chatfuel defines reachable users and subscribers in a different way than is typical with other conversational AI systems, as it focuses on managing a total reachable audience, not the monthly active number of users: reachable users who have not contacted the organisation this month, but whom the organisation could still reach out and contact, would still be charged as a subscriber to the channel.

Chatfuel is billed by the total number of subscribers to the channel. For 10,000 subscribers with the Premium support package the charges are $385/month[16], or approximately $5,000/year.

10.3.2. Botkit

Botkit is an open source platform built on node.js for building bots connecting other NLP to other Channels. As such, it doesn't have a charge for usage, it just needs cloud infrastructure to be purchased to host the app.

[16] https://chatfuel.com/pricing.html

10.3.3 Amazon Lex

Amazon Lex is charged by blocks of 1,000 API calls at $0.75[17] per thousand. Amazon Lex suggests use of Amazon Lambda for making additional processing based on Lex enrichment, which requires separate charges but Amazon Lambda has a generous free tier.

10.3.4. LivePerson Maven

LivePerson typically charges by monthly active users. LivePerson is a full messaging management platform, providing the human user management functions suitable for organisations with large contact centres, together with the NLU functions for bots using external providers or LivePerson maven, and the channel integrations.

The per active monthly user charges are therefore generally substantially more expensive than an NLU platform which provides only the bot functionality.

LivePerson does not publicly publish pricing information.[18]

10.3.5. Rasa

Rasa is provided as: either an open source Rasa stack, including the NLU engine; Rasa core, providing the basics for creating and managing conversations; and Rasa platform which provides enterprise level functionality for building and managing bots with full support.

Rasa Stack does not incur a charge, and Rasa does not publicly publish pricing for the Rasa platform[19].

[17] https://aws.amazon.com/lex/pricing/
[18] https://www.liveperson.com/pricing/
[19] https://rasa.com/products/pricing/

10.3.6. Microsoft Luis

Microsoft is charged by blocks of 1,000 API calls at £1.118[20] per thousand. Microsoft provides the ability to run LUIS within a private container which is paid for separately. Long tail like functionality is provided through the Microsoft QnA maker which is charged for separately and requires three Microsoft Azure instances and a £7.50[21] payment/month for the management portal and APIs.

10.3.7. Google Dialogflow

Google Dialogflow Enterprise Essentials is charged at $2 / 1000 API calls with short tail functionality only, or $4 / 1000 for the Enterprise Plus edition which also executes the knowledge connector long tail functionality search for each API execution.[22]

10.3.8. IBM Watson

IBM Watson Assistant standard plan is charged per individual API call at £0.00167 per API call, so 1,000 API calls would be £1.67[23] on the Standard Plan with short tail only functionality. IBM provide a Plus plan including long tail functionality and multi environment support for small and medium enterprise above this which is billed by monthly active users, as well as a Premium plan above this for large enterprises that is also billed in the same way.

IBM does not publicly publish Plus and Premium plan pricing.

---

20 https://azure.microsoft.com/en-gb/pricing/details/cognitive-services/language-understanding-intelligent-services/

21 https://azure.microsoft.com/en-gb/pricing/details/cognitive-services/qna-maker/

22 https://cloud.google.com/dialogflow-enterprise/pricing

23 https://www.ibm.com/cloud/watson-assistant/pricing/

11. Feature matrix

Area	Feature	Chatfuel	Botkit	LivePerson Maven	Amazon Lex	Microsoft LUIS (with QnA)	Rasa Stack (OOS only)	Google DialogFlow (Enterprise Plus)	IBM Watson (Plus)
NLU	Intent Recognition	Keyword	NA	ML Example Based	ML Example Based	ML Example Based	ML Example Based	ML Example Based	ML Example Based
	Entity extraction	Keyword	NA	Synonym	Automated Expanded Synonym	Contextual NER and Synonym	Contextual NER and Synonym	Contextual NER, Synonyms, Automated Expansion	Contextual NER, Pattern, or Synonym
	API support	Basic Dashboard and Bot Status	NA	Partial	All UI functions	All UI functions	All UI functions	All UI functions	All UI functions
	Long Tail Integration	No	No	Yes - Knowledge Base Search	No	Yes - MS QnA	No	Yes - Knowledge Search	Yes - Watson Discovery
Conversation Flow	Visual Flow Tool	Yes	No	Yes	No	No	No	No	Yes
	Flow configuration method		Botkit Studio - Script authoring tool	Visually Configured Rules	Code level rule configuration	Code level rule config via Dialogs SDK	Machine Learnt Stories	Configured SubIntent Rules	Visually Configured Rules
	Context object supporting complex objects	Yes	Conversational Threading	No	Yes	Yes	Yes	Yes	Yes
Build acceleration	Pre-trained Entities	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Pre-trained/Built-in user intentions	No	No	Pre-built bots for common LP scenarios	Yes	Yes	No	Yes	Yes
	Cluster existing data to suggest intents	No	No	No	No	Yes	No	Via Chatbase	Yes
Integration	Out of the box supported channels	Facebook	Web, Apps, Facebook Messenger, Slack, Twilio, Cisco Spark, Microsoft Teams	IVR, Apple Business Chat, SMS, Web, Mobile Apps, Facebook, Google RCS, WhatsApp	Facebook, Slack, Kik, Twilio	Cortana, Direct Line, Email, Facebook, GroupMe, Kik, LINE, Microsoft Teams, Skype, Slack, Telegram	Facebook, Cisco Webex Teams, Slack, Mattermost, Telegram, Twilio, RocketChat, MS Bot Framework, SocketIO, REST channels	Google Assistant, Facebook Messenger, Slack, Dialogflow Web Demo, Kik, Line, Skype, Cisco Spark, Telegram, Cisco Tropo, Twilio, Twilio Programable Chat, Twitter, Viber	Facebook, Web, Slack
	Support webservice and/or cloud function calls	No	No	Hosted java script functions	Yes - AWS Lambda Cloud Functions	Yes	No	Yes	Yes - IBM Cloud Functions and Webhooks
	Out of the box response types	Text, Image, Audio, Video, RSS, Google Sheets	NA	Text, Image, Audio File, Video, List Picker, Quick Reply	Text, Images, Option Cards	Text, Images, Video, Audio, Files, Buttons	Text, Images	Text, Image, Card, Quick Replies	Options, Images, Text, Pause
Analytics	Analytics Dashboard in Tool	Yes	No	Yes	Yes	Yes	No	Yes	Yes
	Advanced Analytics	NA	Via Middleware Plugins	No	No	No	No	Via Chatbase	Via Watson Studio

12. High-level characterisation of each platform

12.1. Chatfuel

Chatfuel is a popular Facebook messenger based automated response system. It has a focus on outbound messaging and marketing. It uses keyword based NLU but can be extended by connecting it to Google Dialogflow. It is focused on the Facebook channel and is a simple and quick tool for business users to create large scale automated outbound messaging bots and simple inbound message responses. It is cost effective and well-targeted at maximising the messenger features that Facebook provides.

12.2. Botkit

Botkit is a connectivity platform for connecting a wide variety of different channels to a large number of natural language engines. It formalises a message pipeline for transforming inputs from a wide variety of channels to a normalised message format, then passing it to a variety of NLU systems for evaluation and then handles executing the response back to the specified channel. As such, it doesn't have any inbuilt NLU, but can use any of the major platforms. It currently defaults to use LUIS NLU and the Howdy team that built Botkit have recently joined Microsoft. It is currently provided and maintained as an open source framework.

12.3. LivePerson Maven

LivePerson released LivePerson Maven their propriety Bot Tooling and NLU engine in December 2018. Prior to that, LivePerson had a partnership IBM Watson for NLU understanding and bot tooling, and still supports connecting to IBM Watson, Google Dialog flow, or other custom bot connectors. At this time there are no publications of the technology underlying the NLU, nor any available independent benchmarks for its accuracy. However, from examination of the documentation and behaviour of the tooling, it appears to be based on a

similar intent-and entity based training method as with the IBM Watson or Google Dialog Flow connectors provided.

LivePerson is a messaging focused platform and integrates with some of the latest messaging standards such as Apple Business Chat, Google RCS, Facebook Messenger and WhatsApp. It has strong history of managing large scale human webchat and messenger call centres. It would not be typical to try and use LivePerson Maven without utilising the rest of the LivePerson suite. In that respect it has a much larger system footprint than the other conversational AI platforms in the review providing extensive human messaging management, IVR integration, Provided UIs, channel integrations, as well as NLU and Bot functionality.

In an environment with an extensive existing LivePerson webchat, or messaging deployment, LivePerson Maven represents a straightforward next step into starting to automate tasks with conversational AI.

12.4. Amazon Lex

Amazon Lex is the text version of the Amazon Alexa system. It provides an easy to configure UI for building command and control style bots with easy automatic slot filling. More sophisticated configurations rely on its tight integration with Amazon Lambda functions, and onward into other AWS services such as Elastic Search. Amazon Lex is very cost effective and suits well organisations with a large existing footprint of AWS developments already.

12.5. Microsoft Luis

Microsoft LUIS has a heavy developer focus, providing coded configuration of conversational flows. It supports a wide variety of channels and content types, and the base NLU is supported with a strong set of developer SDKs in the Microsoft Bot Framework. It has a strong viewpoint for bot development and provides long tail document ingestion and integration via the Microsoft QnA services.

12.6. Google Dialogflow

Google Dialogflow is the natural language engine which allows users to build apps or actions for google assistant on Android phones or via google smart speakers. It is also available to be used to build chat or messaging systems using text only. It has an easy to use training UI for intents and entities, but uses a series of context based sub intents to configure conversational flows. It supports long tail functionality by linking knowledge connector search like behaviour into Dialog Flow systems. It supports more advanced analytic functions including rich flow visualisations and recommendation of user intentions from existing logs via the Chatbase virtual agent modeller.

12.7. Rasa

Rasa was launched in December 2016. It is included as it provides similar functionality to the major proprietary cloud platforms but is available in an open source format which can be run locally without an internet connection. It is configured primarily by configuration files and has a data science focus and feel to it. It provides the greatest ability to customise the underpinning natural language processing for a bot created with it. It is the only system in this review to use Machine Learnt Stories to configure conversational flow.

However, these features mean that it requires the most work to setup, host and scale, and is probably the least accessible to less technical users.

Rasa have launched a paid version of the Rasa platform which provides more advanced hosting, DevOps and Tooling around the core open source product.

12.8. IBM Watson

IBM Watson is designed for a “no-code” persona and is one of the most non-technical-user friendly platforms whilst having some of the most powerful underlying features including intent suggestion from chat logs and built in long tail integration It scored top in the accuracy benchmarks for this report. IBM has some of the longest experience in this field and provides pre-built content

focused on enterprise rather than consumer use cases. It tends to be at the higher end of the price bracket. IBM provides long tail integration via Watson Discovery and advanced analytics via Watson Studio including pre-built analytics workbooks for accuracy measurement and confusion matric generation.

13. Recommendation

Based on this review, this document would recommend these systems in this order. Specifically, any of the first four (IBM, Rasa, Google or Microsoft) are the strongest competing choices capable of building very powerful systems for councils wishing to implement a chatbot solution:

1. IBM Watson
2. Rasa
3. Google Dialog Flow
4. Microsoft Luis
5. Amazon Lex
6. LivePerson Maven
7. Botkit
8. Chatfuel

A lot of the organisational fit for a platform depends on the style of user targeted. If the project will be very data science led, Rasa might be the primary choice. If developer led, LUIS might be best. Or, if supporting both business and technical users was a priority, IBM might be the best choice.

Similarly, the order might be altered for an organisation already with substantial deployments on one of the cloud platforms already, making an IBM, Google, Microsoft, LivePerson or Amazon solution a natural fit.

The only system which can be deployed without internet connectivity and can form part of a fully open source deployment would be Rasa. It should be noted though that using any of the systems, the data model and responses can be fully open sourced.