Transforming SMS-Based AES: Leveraging Retrieval Augmented Generation, Agriculture 4.0, and User Suggestibility for a Global Approach

Jacqueline Dobreva-Skevington

Florin Ciucu

The University of Warwick

3rd year, 2023-24

1 Acknowledgements

I would like to thank my supervisor, Florin Ciucu, who provided weekly respites to my busy schedule and invaluable insights throughout the year. I would also like to thank my husband, Isaac Dobreva-Skevington. Your eternal support, as always, has been a bedrock for me. I am deeply indebted to you both.

2 Abstract

Agricultural Extension Services (AES) are key in agricultural information dissemination to smallholders, one of the blockers to improving agricultural yields in developing countries. Of existing types of AES, automated SMS implementations show the most promise, and many papers have proposed solutions. However, these typically Machine Learning (ML)-based solutions do not take advantage of Agriculture 4.0 principles, often cannot deal with unseen input, do not tackle the reasons why, even when given information, smallholders do not use it, and have not produced a unified, global solution that supports smallholders over the entire crop cycle for all crops, as well as rarely being both push and pull and supporting natural language input and output. This project presents a new solution, moving away from ML-based strategies and instead presents the first Large Language Model (LLM) Retrieval Augmented Generation (RAG) approach, as well as incorporating new Agriculture 4.0 principles, creating a solution tailored quantitively, to an individual smallholder level, via several APIs and data sources, rather than taking a blanket approach for whole countries. The paper also introduces a novel user suggestibility concept, taking inspiration from behavioural economics and interface mirroring, applied in the project to make responses tailored to each user qualitatively and therefore increase the chances information provided is acted upon. This has all been created as part of an easily extensible global solution, that supports smallholders over the whole crop cycle for all crops and has both push and pull capabilities. The user interface is entirely in natural language. The new LLM RAG approach has a comparable accuracy to ML systems (over 80%) and deals with unseen input, while overcoming traditional dataset issues. Testing has showing this to hold true over three sample countries, accounting for quantitative tailoring. Further evaluation of the applied user suggestibility concept has proven successful, with users 1.5 times more likely to action on the solution tailored qualitatively to them. This paper shows that it is possible to create a global, unified, and personalised solution, and invites further research into using RAG in agriculture, as well as short-text user characteristics and user mirroring to increase engagement and improve suggestibility.

2.1 Keywords

Agriculture, Natural Language Processing, Retrieval Augmented Generation, Large Language Models, Agricultural Extension Systems, User Characteristic Tailoring, User Suggestibility, SMS

3 Contents

1 Acknowledgements 116
2 Abstract 3
3 Contents 4
4 Glossary 5
5 Common Abbreviations 7
6 Motivation 8
7 Agricultural Extension Services 11
8 SMS-based AES 15
9 The Expert System 23
10 Quantitative User Adaptation 51
11 Qualitative User Adaptation 55
12 Message Handling 78
13 System Architecture 91
14 Project Management 96
15 Conclusions 101
16 Areas of Further Research 102
17 Legal, Social and Ethical Considerations 105
18 Licence 106
19 References 107
20 Appendix A 116

4 Glossary

Developing Country There is no clear definition of what a developing country is, and there is much contention about terminology and metrics used to divide countries into groups. For the purposes of this project, countries with a large gap between potential and actual agricultural yields will be termed as 'developing countries'.

Smallholding A small farm, usually family owned, with an average size of five hectares. While smallholdings are often associated with developing countries, they are prevalent worldwide, with the European Union calling them "the backbone of European farming" [1].

Smallholder A person who owns, or labours in, a smallholding.

Agricultural Extension System (AES) Services to improve agricultural information dissemination by providing advice, knowledge, and training to farmers, often supplied by government agencies, although independent groups and NGOs frequently develop their own [2]. Development of AES is financially supported by many organisations, such as the World Bank or the Food and Agriculture Organisation (FAO) [3]. AES improve techniques, increasing in yields and profits, and therefore improving social and educational standards.

Feature Phones Feature phones lie somewhere in between 'brick'/dumb phones and smartphones. They have more capabilities than classical phones, but less than smartphones and therefore may not have capability to install applications or have internet connectivity [4].

Agriculture 4.0 Connecting the agricultural sector to the Internet of Things (IoT), to perform real-time data-driven management of the farm, such as irrigating as and when needed, instead of everyday. Often termed as 'precision agriculture', it increases profits and improves sustainability [5].

Push-Pull Systems Push systems directly send information to a user. In the context of agriculture, this may be in the form of push notifications, warning a farmer about a weather event or pest. Pull systems on the other hand allow users to directly request information rather than it being pushed to them. In the terms of this project, an example of this is the user asking the system a question, and having the system respond. A push-pull system is one that incorporates both together.

Artificial Intelligence (AI) Technology that simulates human intelligence.

Machine Learning A branch of AI dealing with creating statistical models that allow the computer to learn, without being explicitly programmed.

Deep Learning A branch of AI dealing with using neural networks to teach computers to simulate human brains.

Large Language Model A Language Model trained on a large amount of data, capable of understanding and generating natural language, as well as other types of content.

Generative AI Deep Learning models that can generate content.

Retrieval Augmented Generation A technique for enhancing the accuracy of (usually text-based) Generative AI models by providing context content should be generated from.

Crop Cycle The process each crop goes through in a growing cycle.

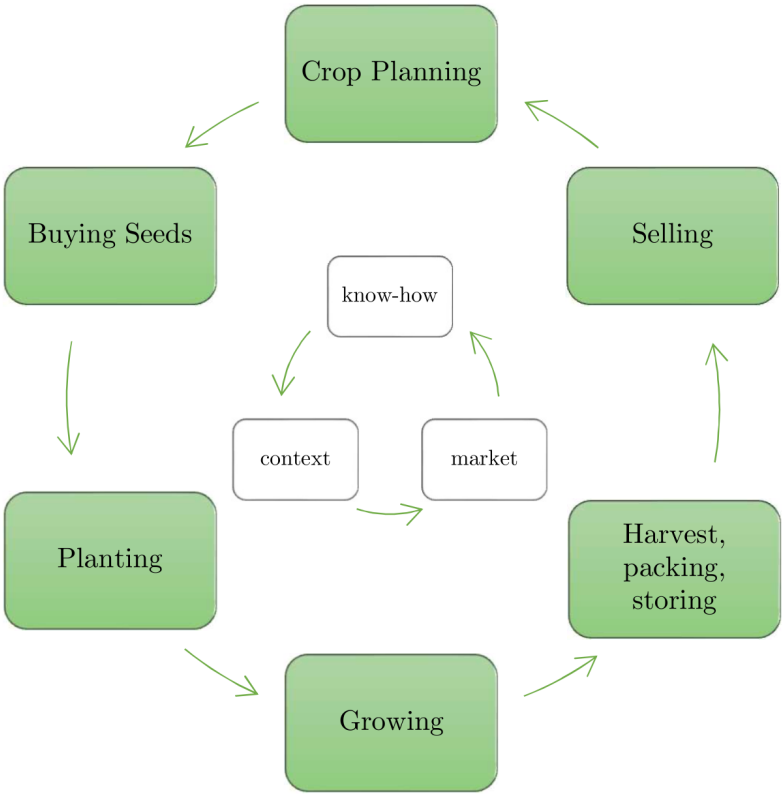


Figure 1 The agricultural crop cycle. Concept taken from [6]

5 Common Abbreviations

AES – Agricultural Extension System
SMS – Short Messaging Service
AEW – Agricultural Extension Workers
ADSS – Adaptive Decision Support System
FAO – Food Agricultural Organisation
PxD – Precision Development, a company
IVR – Interactive Voice Recognition
RAG – Retrieval Augmented Generation
LLM – Large Language Model
NLP – Natural Language Processing
NER – Named Entity Recognition

6 Motivation

90% of the world's 570 million farms are owned and operated by families, with smallholdings producing 46% of all human food supply globally [7]. However, without an increase in agricultural yields, agricultural output will soon not be able to cope with demands.

The world's population is growing at an unprecedented rate, with the Food and Agriculture Organisation of the United Nations (FAO) predicting that the population will rise to 9.1 million by 2050. The amount of arable land has also been declining, due to factors such as desertification, urbanisation, and climate change. In fact, if soil erosion in Africa continues, crop yields will half in 30-50 years' time.

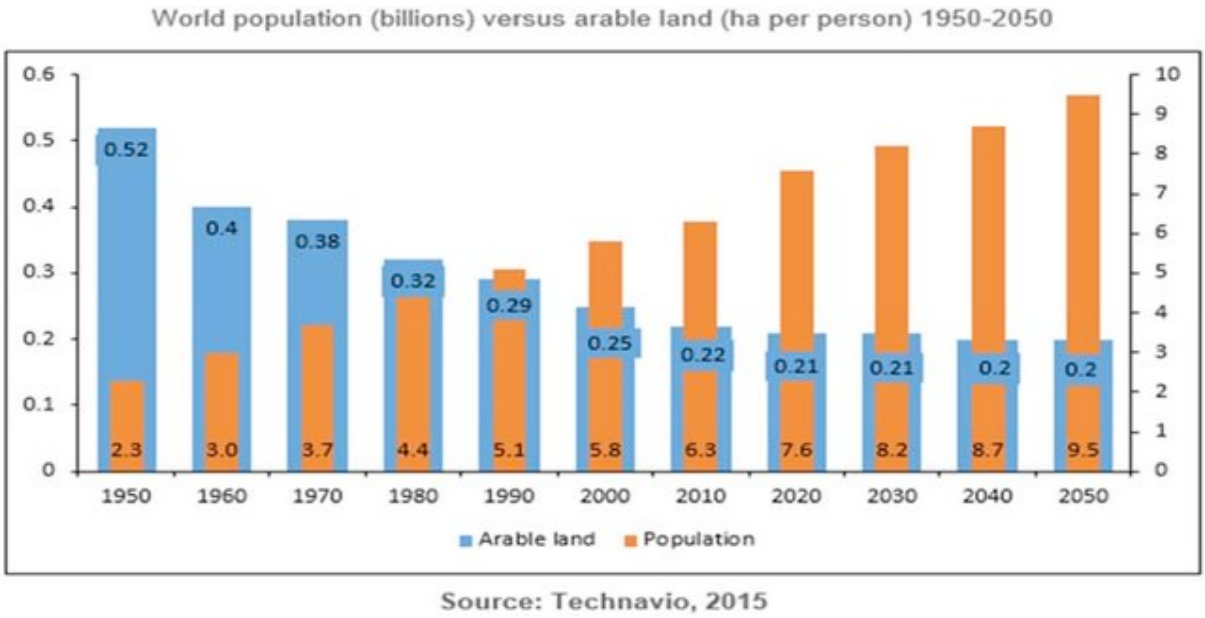


Figure 2 A graph displaying the rise in population, and the fall of arable land.

Growing demand and a decrease of land are not the only issues to contend with. The FAO report 'The future of food and agriculture' highlights other worrying global trends, such as the increase in prevalence of transboundary pests, which means that not only does agricultural productivity need to increase to meet growing demand and dwindling resources, but they also need to increase just to keep output the same [8].

Increasing agricultural yields does not just benefit the consumers, but smallholders too. The UN Sustainable Development Goal 2 states the need to "promote sustainable agriculture" [9] to "end hunger [and] achieve food security". Farmers in developing countries are some of the worlds hungriest people, trapped in a poverty cycle [10],

unable to both grow enough to sell and eat [11]. Increasing their yields alone is two to four times more effective in raising their incomes, and therefore their quality of life, compared to any other sector [12].

And yet, over the last 60 years, agricultural yields in developing countries have lagged, with yields in Low-income countries stagnating, in comparison to High and Middle-Income Countries who have increased yields by over 400% in some areas.

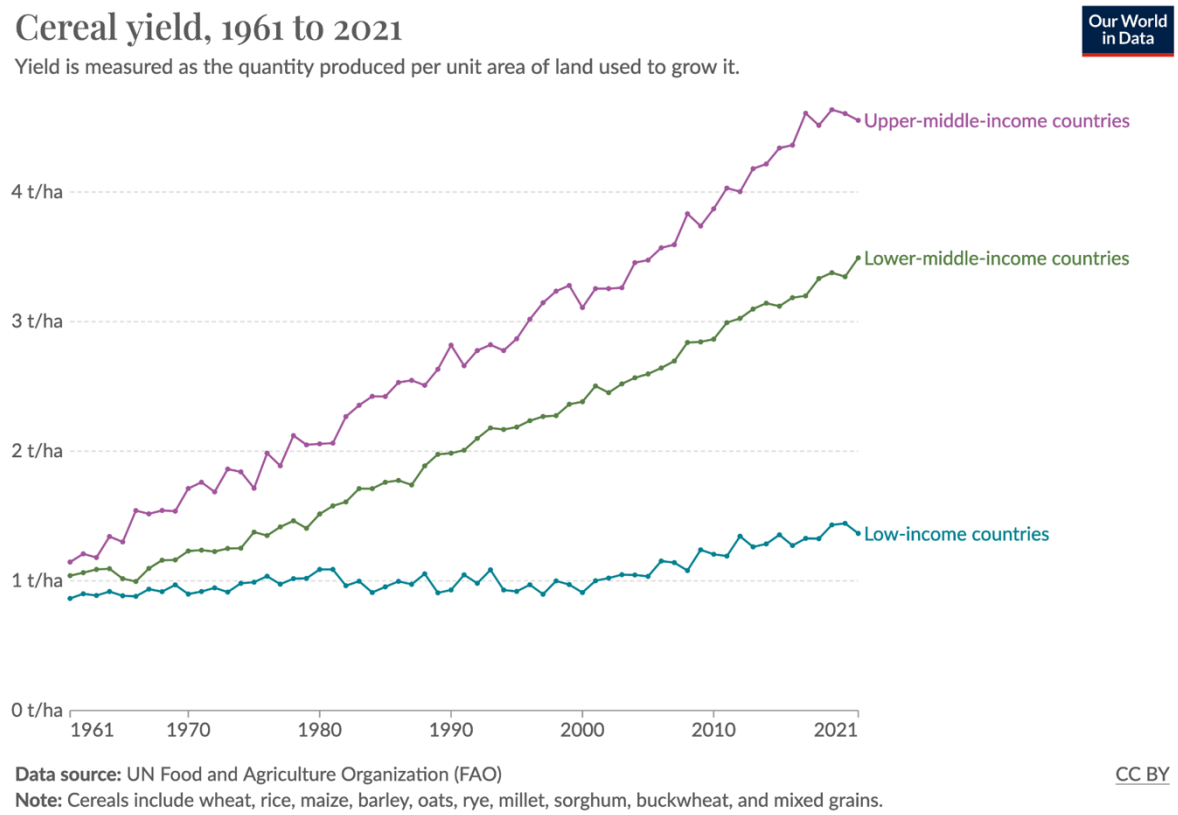


Figure 3 A graph displaying the differences in agricultural yield over the past 60 years.

Farm productivity in developed countries grows year on year, far surpassing the productivity of smallholdings in developing countries [13], but a multitude of complex factors supresses their ability to match the pace. A main issue is the difficulty of information dissemination in developing countries, along with the clustering of Agricultural 4.0 research in developed countries, the lack of any unified solution, and smallholders not implementing suggestions.

The goal of this project is to present the next stages of SMS-based AES, leveraging Retrieval Augmented Generation - a pioneering new approach - and produce the first ever global, unified solution while incorporating Agriculture 4.0, allowing smallholders

to have real time tailored data at their fingertips. The project also attempts to overcome barriers in smallholders not implementing suggestions by introducing a novel suggestibility application on qualitative tailoring based on user characteristics.

7 Agricultural Extension Services

Agricultural Extensions are services which educate smallholders about agriculture, leading to improved techniques, and therefore increased yields and profits [14]. This improves social and educational standards within the community, not only through the extra income, but also continued education as farmers share their successes and acquired knowledge through their community [15]. Forms of Agricultural Extension Services (AES) have been around for 2000 years, with modern types arising in the middle of the 19th century [16] each with their own unique approach.

This section will present the main types of AES, along with their unique benefits and drawbacks. Through doing this, we will be able to envision what an ideal AES could look like and decide on the most appropriate medium for implementing it.

7.1 Types of AES

7.1.1 Agricultural Extension Workers (AEW)

AEW are the traditional type of AES. Workers are educated in government-led training facilities or universities, and then travel to villages to spread information or run workshops, serving as knowledge brokers. This targeted human-facing approach is very effective. AEW can tailor information based on each individual farmer's needs and demonstrate techniques, as well as selling farmers the best seeds or equipment [17]. But at only one AEW to every 1000 farmers [18], the work they can do is limited.

Furthermore, in some regions AEW are extremely unregulated. There is no unanimous training standard or certification [19], leading to some AEW being unskilled and not having enough knowledge to respond to farmers' queries or demonstrate techniques correctly. Recent evidence shows that even AEW who have good knowledge find that they can't cope with growing needs of farmers to keep up with the rapidly changing farming landscape [20]. AEW have also been found selling farmers potentially dangerous or useless products to benefit themselves [21], putting the farmer out of pocket and potentially harming their yields, and those of the community.

Regulation isn't the only issue. Due to the remote locations of villages, there is also a lack of accountability, with AEW not visiting farmers they report they have. This lack of accountability means that AEW also are free to discriminate, with individual case studies showing discrimination on age, gender, and perceived health status – such as

refusing to visit smallholders who are rumoured to have HIV/AIDs [22].

7.1.2 Radio [23], TV [24], and Newspaper [25]

Radios often have dedicated AES channels for each village/region allowing smallholders to benefit from curated and tailored information. There may be special broadcasts that provide real time information, such as weather updates and what they mean for certain crops. However, this requires the farmer to be listening at that exact moment. Access to radio forms of AES also require smallholders to own a radio, and information given may not be trustworthy as anyone with the right equipment can set up a radio station. Radio is mostly a one-way communication form, potentially reducing engagement.

Newspapers tend to be a more reliable source than radio and smallholders can engage with content any point they like. Newspapers can also provide agricultural predictions until the next publishing date (such as weather forecasts) in an easy to access way. However, accessing this information often comes at an upfront cost via purchasing the newspaper, and the information may be less targeted, as newspapers are often done on a regional level.

TV has much more regulation than with radio and newspaper, so information is more reliable. Broadcasts can also be real time – which is valuable in agriculture where real time information can make the difference between a good and bad harvest. TV is also a visual form of media, allowing technique demonstrations to be broadcast for smallholders to watch. However, as with radio, access to a TV may be limited, and information communication is only one-way, so information may not be relevant or useful to the smallholder, especially as TV broadcasts tend to be done on a more regional/national level.

7.1.3 Mobile applications [26]

Mobile applications allow a smallholder to access information at any place, any time, based on what they need. However, it requires a smartphone, and a higher level of proficiency with technology than other mobile-based technologies. These mobile applications may only be relevant to one region/country, so smallholders in other locations may not be able to use them.

7.1.4 SMS-based AES

With ease of accessibility being the main contributor to a smallholder's satisfaction with an AES [27], SMS-based AES capitalise on the fact that most smallholders own a feature phone [28].

There are two main approaches currently in use:

1) AEW respond to farmers queries over SMS, therefore freeing up AEW from costly travel and allowing instant access for farmers [29]. However, this approach cannot scale easily.
2) Farmers communicate entirely with an automated system and may get passed on to an AEW in case of an unresolvable issue or unseen input [30].

Automated systems allow for real time data to be transmitted to smallholders. Sending messages is low cost [31], ensuring scalability, and messages have the potential for personalisation, which may reduce the marginalisation of certain groups of people typically side-lined by other forms of AES.

However, in their current form, automated systems are usually limited to Push systems (Government texting cocoa farmers that a certain pest is coming) or Pull systems (user goes through a series of options to get information they want). Pull systems also tend to not support natural language input or output, using prompts such as 'text R for rice'. There is already a general mistrust of technology in smallholder communities [32], meaning that they may not engage with such systems.

Current developed SMS-based AES also tend to focus on a specific group of farmers, in one region, doing one type of farming, for only one part of the crop cycle - very little work has been done to produce an integrated environment [22]. This limits the effectiveness of AES as there is no one place to go for all smallholder knowledge. Implementations also do not take advantage of the agriculture 4.0 research and development [30] – using real time data to inform farming decisions - that has allowed farmers in developed countries to increase their yields so much. This means SMS-based AES tend to be blanket wide approaches that do not tailor to a farmer's smallholding.

7.1.5 Evaluation

Through reviewing the different types of AES, we can see that many AES systems have a lack of extensibility and scalability and may be an inconvenience to the smallholder to access, as well as potentially only providing a one-way communication stream (a push system). Agriculture is also highly reliant on accurate real time data, however in many of the types of AES reviewed, smallholders may have difficulty accessing this information, and information provided may only be at a regional/national level and as such not applicable to the smallholder. However, some AES have unique benefits, such as the tailored advice a good AEW could give, and the accessibility of SMS-based AES.

Overall, the ideal AES would be extensible and scalable while providing tailoring based on the smallholder and offering instant access so the smallholder can utilise the system as and when they need it. It would also incorporate real time data insights applicable to the farmer's smallholding. There are only two forms of AES that could do this – mobile applications or automated SMS-based AES. Of these, only SMS-based AES is accessible by most smallholders, and therefore shows the greatest potential promise. Using an SMS-based AES also automatically produces a scalable, extensible solution with instant access to smallholders if it is fully automated. As such, this project focuses on SMS-based AES.

8 SMS-based AES

There are multiple methods that could be taken to develop an SMS-based AES solution. However, so far, a set of deficits in existing SMS-based AES have been identified:

- A lack of a pull and push system that a user can both ask questions to and receive notifications from.
- No systems which leverage Agriculture 4.0 principles, providing real time data.
- Not tailoring based on the smallholder, such as giving information specific to their land or crops.
- Using machine-like or unintuitive language which makes the system less credible and more difficult to use in the case of automated SMS-based AES systems.
- Fragmented solutions that cover only small aspects of the crop cycle for a small subsection of crops in only one or two countries.

This section presents existing solutions and background research on SMS-based AES, selected to provide the greatest coverage amongst these issues, and thereby allowing us to get a better picture of SMS-based AES. This will allow us to draw up the final set of objectives for this project at the end of this section.

8.1 Existing Solutions

Precision Development (PxD) is the leading SMS-based solution deployed in 10 countries, with 5.7 million users in 2021 [33]. Based on the information and images available on their official website, it can be inferred that each country runs on a separate deployed system, with some countries having multiple deployed systems to support different crops/crop cycle areas. These deployed solutions are not linked, so smallholders may have to switch through multiple systems to get an answer to their question.

They run most of their systems on 'Paddy' – their user communication framework – which supports messaging and Interactive Voice Control (IVR). Paddy is a collection of out-of-the-box flows that can be customised, providing an extensible and generic solution so new systems can be built and deployed with more speed. Organisations can get in contact with PxD and use Paddy in their own solutions as well, increasing the coverage of their solution. Below, the three flows provided through Paddy are discussed.

1) Message series: Users receive a fixed linear stream of messages and can only interact through 'Press A for more'. This is an example of a pull-system.
2) Menu: Users are given some options on a specific topic, such as 'text R for rice'. Selecting an option sends the user to flow 1 – message series.

Figure 4 PxDs menu flow [34].

3) Survey: Surveys allow PxD to perform data collection, such as asking for the user's location.

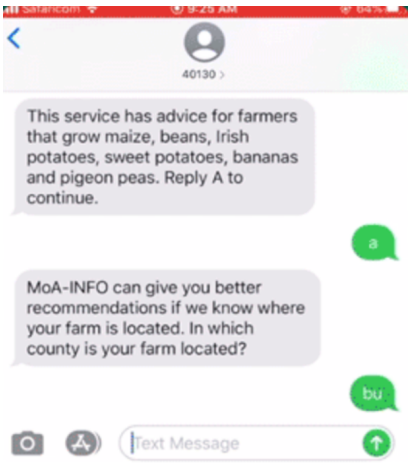


Figure 5 PxDs survey flow [35].

In Figure 4 and Figure 5 we can see that Paddy provides only limited natural language input and doesn't support the creation of a push system for real time notifications. Paddy does provide some tailoring, but, real time data is not incorporated, limiting its usefulness. Furthermore, PxD develop all their systems through combining permutations of the workflows show above, meaning solutions are hard-coded and have

limited extensibility. Some tools also only focus on a specific pest for a specific crop, meaning that the system is not a one stop for all, and farmers need to look elsewhere for any other information they need.

Overall, this is a good system that helps millions of people, and the start of a generic solution through their development of Paddy. However, the lack of natural language input, the need for a separate deployment for each part of the crop cycle, for each crop, for each country, limits its effectiveness and reach, as well as the lack of real time-data utilisation and no push functionality to deliver time critical messages.

8.2 Research

8.2.1 ML-Based Approaches

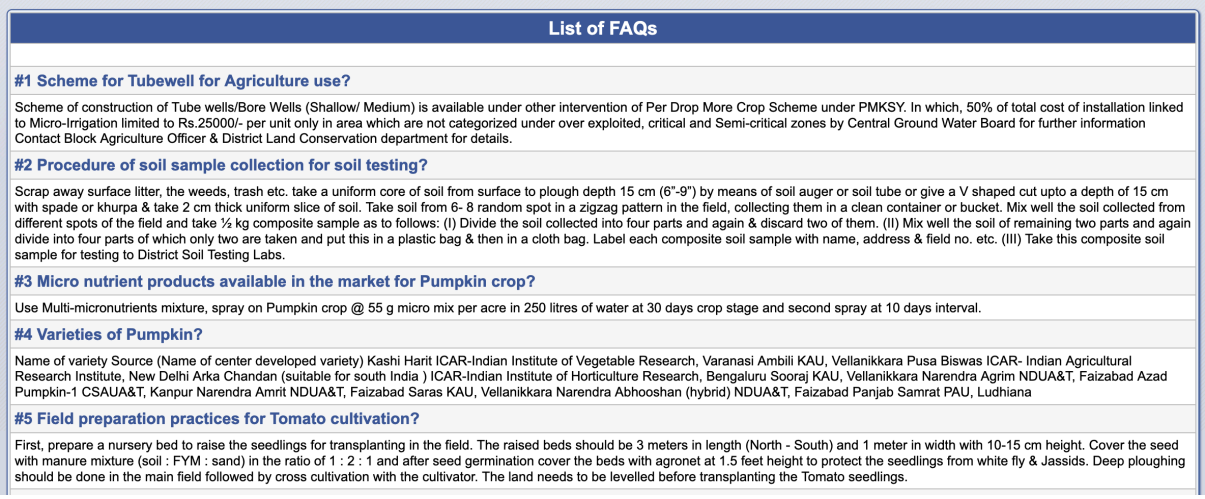
Most of the research specifically focusing on SMS-based AES implementations has been concentrated on ML focused approaches. Below are two representative exemplary solutions compared side by side.

	AgriBot	Intelligent Chatbot
Implementation	Question answer dataset. Sequence to sequence pattern. Generative model.	Question answer dataset. Cosine similarity. API for weather and market prices.
Positives	Can respond to unseen input.	High accuracy. Incorporates real-time data.
Negatives	Limited to India. Only English. Not Natural Language. Not whole crop cycle.	Limited to India English and Hindi. No unseen input – refer to AEW.

Table 1 Comparing two example ML-based approaches.

Both require a very specific format of data – a question answer dataset - meaning there is a knowledge acquisition bottleneck, with data requiring a significant amount of pre-

processing and often needing an expert in the loop to review answers. Both use the Kisan Call Centre (KCC) Dataset - a dataset of common questions and answers that Agricultural Extension Workers (AEW) who work in the Kisan Call Centre use. When call workers encounter a question not in the KCC dataset, they add their answer in, creating a comprehensive dataset for farming in India.



List of FAQs

#1 Scheme for Tubewell for Agriculture use?

Scheme of construction of Tube wells/Bore Wells (Shallow/ Medium) is available under other intervention of Per Drop More Crop Scheme under PMKSY. In which, 50% of total cost of installation linked to Micro-Irrigation limited to Rs.25000/- per unit only in area which are not categorized under over exploited, critical and Semi-critical zones by Central Ground Water Board for further information Contact Block Agriculture Officer & District Land Conservation department for details.

#2 Procedure of soil sample collection for soil testing?

Scrap away surface litter, the weeds, trash etc. take a uniform core of soil from surface to plough depth 15 cm (6"-9") by means of soil auger or soil tube or give a V shaped cut upto a depth of 15 cm with spade or khurpa & take 2 cm thick uniform slice of soil. Take soil from 6- 8 random spot in a zigzag pattern in the field, collecting them in a clean container or bucket. Mix well the soil collected from different spots of the field and take ½ kg composite sample as to follows: (I) Divide the soil collected into four parts and again & discard two of them. (II) Mix well the soil of remaining two parts and again divide into four parts of which only two are taken and put this in a plastic bag & then in a cloth bag. Label each composite soil sample with name, address & field no. etc. (III) Take this composite soil sample for testing to District Soil Testing Labs.

#3 Micro nutrient products available in the market for Pumpkin crop?

Use Multi-micronutrients mixture, spray on Pumpkin crop @ 55 g micro mix per acre in 250 litres of water at 30 days crop stage and second spray at 10 days interval.

#4 Varieties of Pumpkin?

Name of variety Source (Name of center developed variety) Kashi Harit ICAR-Indian Institute of Vegetable Research, Varanasi Ambili KAU, Vellanikkara Pusa Biswas ICAR- Indian Agricultural Research Institute, New Delhi Arka Chandan (suitable for south India) ICAR-Indian Institute of Horticulture Research, Bengaluru Sooraj KAU, Vellanikkara Narendra Agrim NDUA&T, Faizabad Azad Pumpkin-1 CSAUA&T, Kanpur Narendra Amrit NDUA&T, Faizabad Saras KAU, Vellanikkara Narendra Abhooshan (hybrid) NDUA&T, Faizabad Panjab Samrat PAU, Ludhiana

#5 Field preparation practices for Tomato cultivation?

First, prepare a nursery bed to raise the seedlings for transplanting in the field. The raised beds should be 3 meters in length (North - South) and 1 meter in width with 10-15 cm height. Cover the seed with manure mixture (soil : FYM : sand) in the ratio of 1 : 2 : 1 and after seed germination cover the beds with agronet at 1.5 feet height to protect the seedlings from white fly & Jassids. Deep ploughing should be done in the main field followed by cross cultivation with the cultivator. The land needs to be levelled before transplanting the Tomato seedlings.

Figure 6 An example of Questions and Answers in the KCC dataset.

However, answers in the KCC dataset are not updated to follow new best practices and do not incorporate real time data. They are also only applicable for India, and answers are generic, providing no tailoring on a regional/smallholding level. Therefore, using a concrete dataset like this is not a feasible approach to creating a global unified solution, as a Q&A dataset will need to be found for each region globally, and each one will need to be pre-processed a different way.

Both papers use a different approach to determine the best fit answer to the question asked. For instance, if asked "What kinds of pumpkin can I plant?" the system should correlate that with the question "#4 Varieties of Pumpkin" in Figure 6 and respond accordingly. On top of this best fit matcher, the Agribot paper applies a generative model to provide basic responses to unseen input or questions the sequence-to-sequence pattern matcher cannot produce a match for. On the other hand, the other paper 'Intelligent Chatbot' simply refers unseen input to AEWs – meaning that the solution presented isn't scalable and has many of the same drawbacks as non-automated SMS-based AES and AEW.

However, 'Intelligent Chatbot' does use some Agriculture 4.0 principles through incorporating real time weather and market data, providing more tailored and time-specific information. Both systems, however, are only applicable to one country,

support limited languages, and do not provide coverage for the whole crop cycle. While the second paper does not provide images of their system in action, the first paper does, showing non-natural language input and output, making the system less intuitive reducing its credibility for smallholders.

Figure 7 Agribot system in use. Note the lack of natural language input and output. We can see it does provide some real time data, but it doesn't say what that means for the farmer.

8.2.2 LLM-based Approaches

Recently, there has been an increase of interest in Large Language Models (LLMs), which has been reflected in AES. However, as this is a recent development, less than 10 studies have been done in this area, with most being specific case studies – such as 'Leveraging Large Language Models for Improving Extension Services in Nigeria' [36]. Only two provide general concrete approaches, and both were published after the progress report (and therefore after the project's design was chosen, and implementation had begun). It is worth noting that due to the fast-growing nature of this field, more studies may have been published after the submission of this dissertation.

8.2.2.1 'Large Language Models and Agricultural Extension Services', Tzachor et al. [37]

This paper uses OpenAI for its LLM base, and then fine-tunes it (incorporates additional knowledge into the LLM itself) to provide responses to agricultural queries. The authors found that this was a manual process requiring experts and was therefore time-consuming and not easily extensible.

The paper also states that a "response would ideally consider the farmer's location (relevant for soil, weather and market conditions)", alluding to Agriculture 4.0 principles, however it goes on to say that "we are still far from such capabilities". The paper also mentions that "incorporating the conversational history between the chatbot and each farmer can further enhance the chatbots ability to generate tailored recommendations". It also suggests future systems could "adapt to the individuals specific needs, preferences and context". This project will address all these statements.

8.2.2.2 'RAG vs Fine-tuning: Pipelines, Trade-offs, and a Case Study on Agriculture', Balaguer et al. [38]

This paper compares the accuracy of Retrieval Augmented Generation (RAG) and fine-tuned models when applied to agriculture.

RAG is a context augmentation pattern. When a user asks a query, data pertaining to that query is retrieved and then added to the question as context. The prompt and the context are then fed to the LLM, which uses the context to formulate a response back to the user.

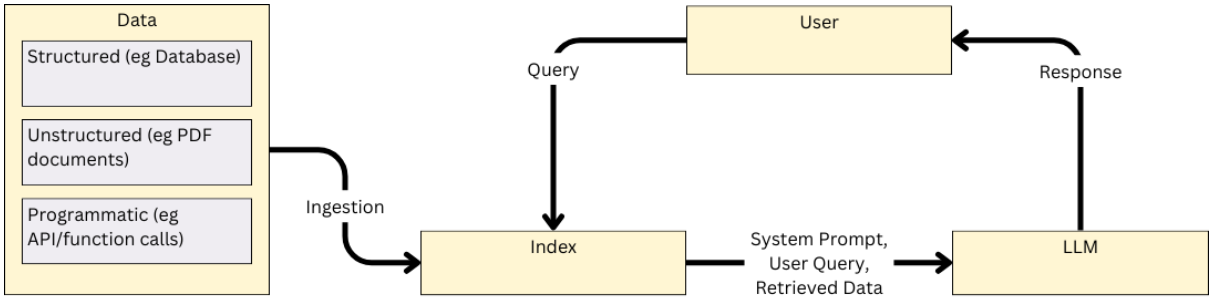


Figure 8 A diagram showing the key concepts of RAG.

The paper found that using GPT-4 as their LLM base gave the greatest accuracy (outlined in red on Table 2 overleaf). The table shows the accuracy of the model when asked an agricultural question without RAG on the left, with whether the model has also been fine-tuned indicated by a tick. We can see that the non-fine-tuned GPT-4 without a RAG was middling in its performance of 75%±3%, but the fine-tuned GPT-4 was well above any models without RAG at 81%±5%.

Model	Fine-tuned	Accuracy	+RAG
Llama-2-chat 13B		76% ±2%	75% ±2%
Vicuna		72% ±2%	79% ±2%
GPT-4		75% ±3%	80% ±4%
Llama2 13B	✓	68% ±3%	77% ±2%
GPT-4	✓	81%±5%	86% ±2%

Table 2 Accuracy of base and fine-tuned models with and without RAG.

It's interesting to note that they found that the accuracy of a solely fine-tuned LLM was comparable to the RAG implementation (highlighted on the table). Of course, fine-tuned models with RAG applied on top provided the best accuracy (86%±2%), but it is worth remembering the takeaways of the paper by Tzachor et al. – fine-tuning LLMs is an expensive and manual process, and the end model is only as good as its dataset. RAG also allows for context aware conversations, something Tzachor et al. notes would be beneficial.

Another interesting thing raised by the paper is the concept of tailoring – "what if we could provide location specific insights to a farmer?". They tailor their model to three countries – Brazil, USA and India, using agricultural documents, showing that some tailoring can be done, in contrast to Tzachor et al.'s statement "we are still far from such capabilities". They also find that the succinctness of a RAG-only GPT model outperforms any other model and combination of RAG and fine-tuning – important for an SMS-based service.

8.3 Objectives

Given the conclusions of looking at the general types of AES in the first section, as well as the deployed solutions and research presented in this one, various limitations of existing systems have been identified. These include a lack of natural language input and output, systems not being both push and pull, not including real time data (and therefore not leveraging agriculture 4.0, and not tailoring to teach specific smallholder.

In terms of implementation, several challenges have also been identified. With ML based approaches, standardised datasets need to be used, meaning significant effort is required to procure and pre-processing data. As existing datasets are only applicable to a nation or region basis, to create a global of solution thousands of datasets would need to be found and processed, meaning the solution is not easily extensible. Dealing with unseen input is also a significant issue with Machine Learning approaches.

With LLM based approaches, finetuning requires significant effort to set up, with Tzachor et al stating that tailoring is not currently feasible. However, RAG provided the most promising approach, potentially allowing for tailoring and real-time data to be incorporated, as well as including conversation context.

PxD demonstrated a solution that could be deployed in multiple different countries for different parts of the crop cycle. However, with each individual section requiring its own deployment, and solutions being limited to different combinations of the same flows, it would be ideal to find a more generic solution.

Several strengths have also identified – for instance, the human touch with AEW, with their tailoring and natural language conversing. Adaptive Decision Support Systems (ADSS), a development in Decision Support Systems which aim to support businesses in making decisions [39] has also introduced the idea of tailoring based on the user, not just on their location (and therefore local weather, market prices etc), but also based on their user themselves, giving them answers adapted to know much a user knows about a specific topic [40]. This would be incredibly useful within the context of an SMS-based AES, as it would ensure smallholders always get information presented to them at a level they understand best, which may in turn mean more smallholders implement the suggestions presented by the system.

Bringing this all together, the overall objectives for the project are:

- Making a system that is both pull and push, allowing the user to not only interact with a system and ask it questions, but also receive notifications on important information, such as a pest arriving in the area.
- Making a system that supports natural language input and output to overcome 'tech mistrust' and therefore ensure that suggestions are implemented.
- Tailoring based on the user, both quantitively (Agriculture-4.0) and qualitatively (ADSS information based on user knowledge) – improving user suggestibility and ensuring suggestions are implemented, something which hasn't been done within the context of agriculture before.
- Incorporating real-time information (Agriculture-4.0) to ensure farmers receive important up-to-date information that affects their crops.
- Making a one stop extensible solution that supports all crops, over the whole crop cycle, globally, so farmers don't need to find multiple different sources of information of varying degrees of accuracy to be able to run their smallholding.
- Overcoming the need for a specific dataset format – 'knowledge acquisition bottleneck'.

9 The Expert System

An Expert System's goal is to provide a user with a decision. The correctness of this decision should be close to, or better than, that of a real expert [39]. As that would be ideal to emulate with our project, the question answering part of the system has been called the Expert System.

In this section we will go through the design and implementation of our expert system, keeping in line with the identified objectives. Of these objectives, the following will be discussed in this section:

- Making a push-pull system
- Making a system that supports natural language input and output.
- Incorporating real-time information (Agriculture-4.0).
- Making a one stop extensible solution that supports all crops, over the whole crop cycle.
- Overcoming the need for a specific format of dataset.

After we have discussed implementation, the Expert System will be evaluated to determine whether the above objectives have been met successfully.

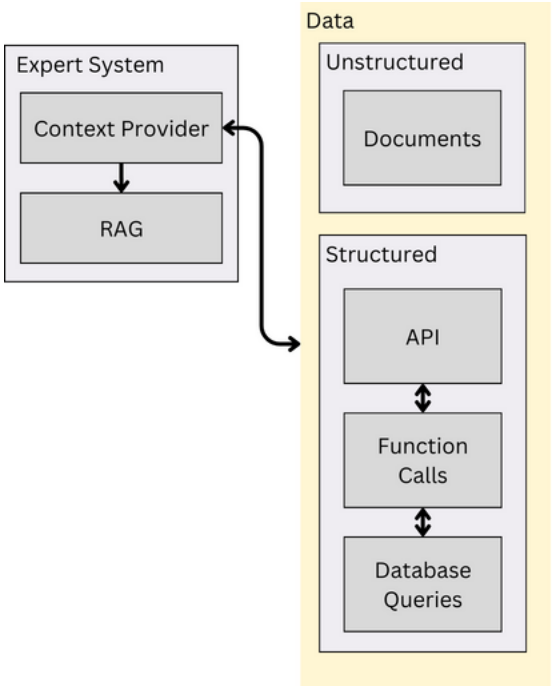


Figure 9 The architecture that will be discussed in this section.

9.1 Initial Design

This project presents a novel approach to SMS-based AES, by utilising Retrieval Augmented Generation (RAG). Typical ML approaches have many drawbacks, as previously discussed, and fine-tuned LLMs are hard to keep up to date and tailor [37], both of which are key objectives of the project.

RAGs' ability to use structured, unstructured, and programmatic forms of data means the 'knowledge acquisition bottleneck' can be overcome as data doesn't need to be in a specific format or extensively pre-processed, making the development of SMS-based AES easier and more extensible, as well as meeting one the project's objectives. Programmatic forms of data also mean that real-time data can be fed to the LLM, ensuring smallholders receive up-to-date critical information tailored to their needs.

This also meets another objective and it at contrast to some of Tzachor et al.'s conclusions. Furthermore, RAG can be made context aware by incorporating a users' conversational data, meaning users can ask for clarifying information, or give some more themselves. The ability to do this means that answers can be more accurate and can have even more of a natural language feel. The benefits of this were also mentioned by Tzachor et al earlier.

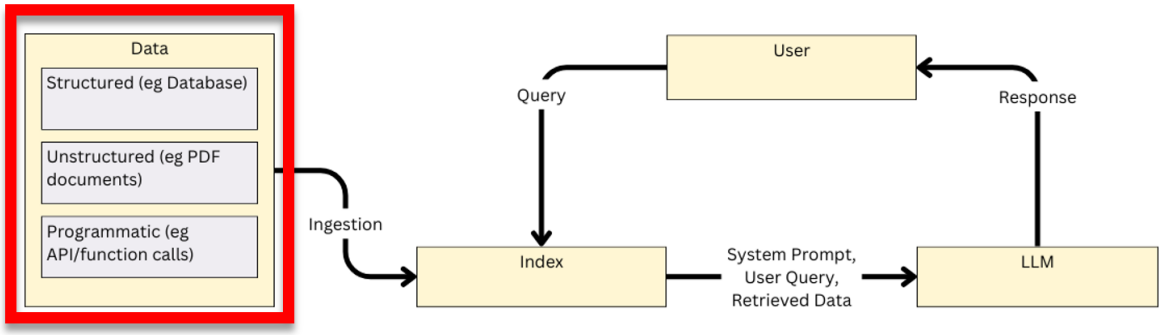


Figure 10 The key concepts of RAG with its flexible data types highlighted.

9.1.1 Initial Problems

9.1.1.1 Translation

To meet the objective "making a one stop extensible solution", the system needs to be suitable for users globally and therefore needs to be able to take input, and produce output, in multiple languages. There are two possible solutions to this:

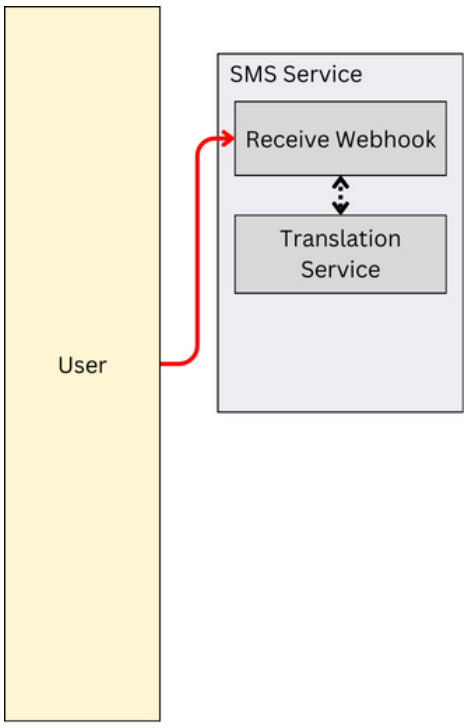


Figure 11 The user interfacing with the receive SMS webhook, which may interface with the translation service as needed.

1) Feed the message as-is to the LLM. The LLM will then produce an output in the same language due to multilingual capabilities. Messages will also be stored in the original language in the database.
2) Receive the message and translate into English. Pass the message to the LLM in English and have the LLM produce an English response. Translate the answer into the original language and send it to the user. All messages are stored in English in the Database.

The second solution makes the overall system design simpler, and despite many LLMs having multilingual capabilities, most of their base training data is in English, and their accuracy decreases

when asked to produce output or respond to a question in a different language. OpenAI (the LLM used in the system due to the findings by Balaguer et al.) has the same training data biases as other LLMs capabilities - 93% of its unlabelled base data is in English [41].

9.1.1.2 Data

As highlighted in Figure 10, RAG provides great flexibility with data. However, there are still some difficulties with gathering data sources for the RAG. There is no data source that is a one-stop call for agriculture, supporting the whole crop cycle, for all crops, globally. Any existing data sources only provide a subset of necessary information – for instance, commodities APIs only provide global prices for a select few crops, and these crops tend to only be provided in groupings, such as 'dry beans' which may be less useful for a farmer who needs to know the price of soybeans.

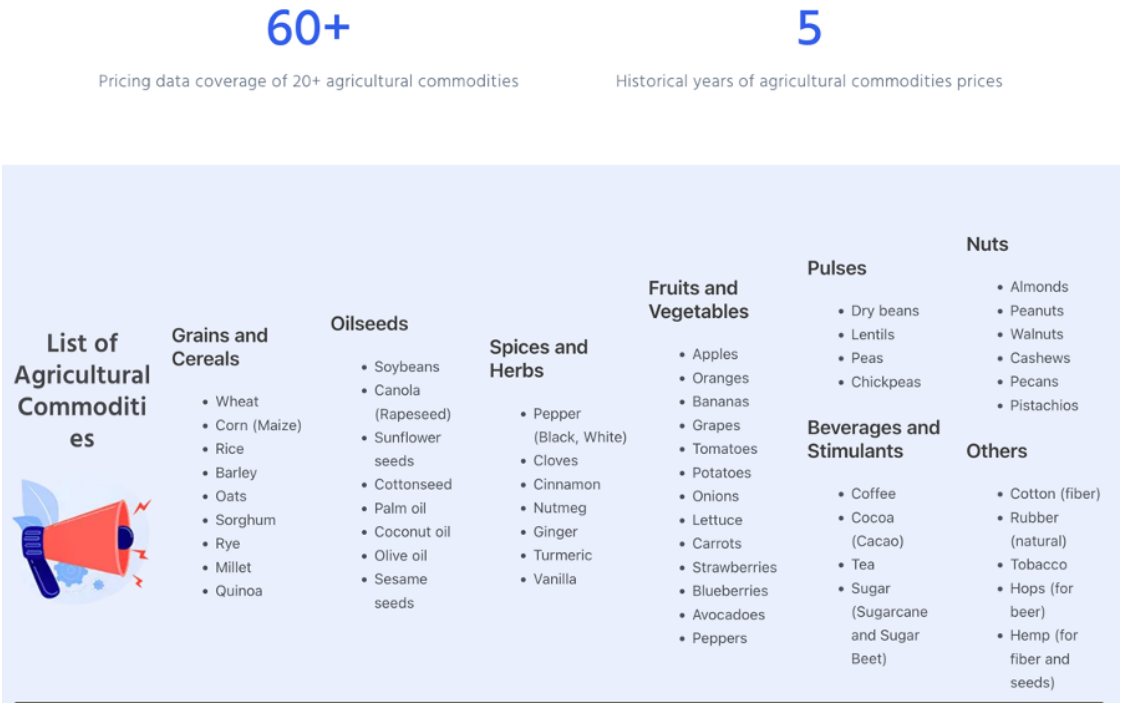


Figure 12 Commoditic API website. This API provides the largest range of commodity pricing, but many crops are missing [42].

Any data sources found also need to be extensible by location specific APIs, such as weather, to be able to tailor to each specific user on a granular per-smallholder level, to meet the agriculture-4.0 objective. This provides a further challenge with data collection. However, these challenges provide the opportunity to create many datasets, all of which will be published after the project as standalone RESTful APIs, allowing others to benefit from this work as well.

9.2 Creating the Knowledge Base

9.2.1 Packing and Storing

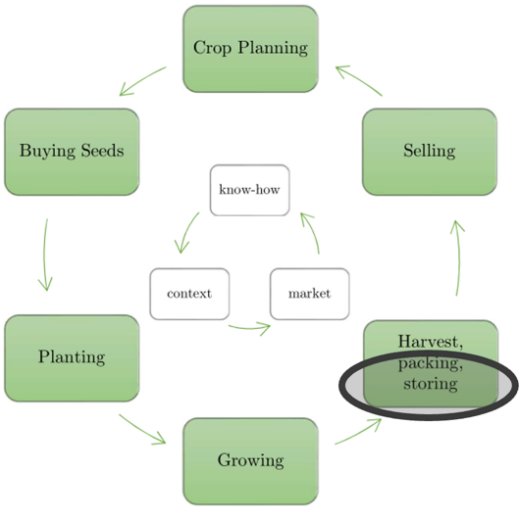


Figure 13 The crop cycle, with the discussed aspect highlighted.

Before crops are sold, they need to be potentially stored. An example of this is 'clamping' summer maincrop potatoes by placing them in clamps of soil and hay so they can be stored throughout winter and sold as needed. When ready for sale, crops then need to be packed. While there is no global hub of information on how each plant is packed and stored, plants can be divided into broad categories that have similar packing and storing needs. The IR-4 project divides plants into 25 main groups, each with subgroups, providing a list of example crops in each group and subgroup [43], which can be used by the RAG to infer which subgroup a given crop belongs to.

As other parts of the crop cycle rely on crop groups as well, it is best to store them as static data to the database, rather than providing the pdf document with crop group divisions directly to the RAG. By storing the crop groups in the database, they can be quickly queried and potentially used by programmatic function calls. First, the pdf provided by IR-4 can be processed into a Tab Separated Value (TSV) file for ease of parsing (Figure 14).

```
1. ROOT AND TUBER VEGETABLES    "Arracacha; arrowroot; artichoke, Chinese; artichoke, J
1A. Root vegetables subgroup    "Beet, garden; beet, sugar; burdock, edible; carrot; ce
1B. Root vegetables (except sugar beet) subgroup    "Beet, garden; burdock, edible; car
1C. Tuberous and corm vegetables subgroup   "Arracacha; arrowroot; artichoke, Chinese;
1D. Tuberous and corm vegetables (except potato) subgroup   "Arracacha; arrowroot; arti
2. LEAVES OF ROOT AND TUBER VEGETABLES (HUMAN FOOD OR ANIMAL FEED) GROUP    "Beet, garc
3-07. BULB VEGETABLE GROUP  "Chive, fresh leaves; chive, Chinese, fresh leaves; daylily
"3-07A. Onion, bulb, subgroup"  "Daylily, bulb; fritillaria, bulb; garlic, bulb; garlic
```

Figure 14 A TSV of the IR-4 Crop Groups

The crop groups can then be stored using a relational structure in the database via entities. *CropGroup* entities have a letter and number, and may contain several *Crop* entities, which are stored in a separate table, ensuring they can be searched quickly and effectively to determine which crop is in which group, while at the same time providing a mechanism for the system to fetch a list of all the crop groups. Using these general groups, a text file with long term (dry) and short term (fresh) storage solutions

can be created, along with each crop groups' packing instructions. This text file can then be referenced by the RAG.

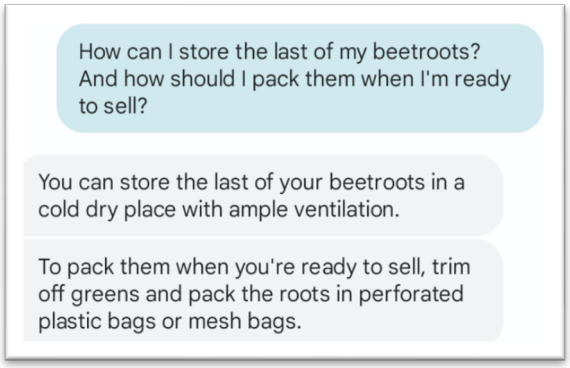


Figure 15 Packing and storing in action – real Expert System output. Beet is part of the Root and Tuber Vegetables group, all of which have these similar packing and storing needs. The RAG uses the crop group table to determine what group beets are in, and then references this to the packing and storing PDF.

9.2.2 Buying Seeds

To buy seeds, farmers need to know how many they need to buy, and how much it will cost them. There is no global index of seed prices, so the first step of being able to tell farmers how much it will cost them is taking UK seed prices and using Purchasing Power Parity (PPP) to estimate what those seed prices will be in other countries.

```
/**
 * In memory map for the purchasing power parity (reduces query time and not very big so
 * doesn't need to be stored in DB)
 * Map<Pair<CountryName, TheYearConcerned>, ThePPP>
 *     For all years (about 20)
 * */
val PPPs: Map<Pair<String, Int>, Float> by lazy {
    val file = File(this::class.java.classLoader.getResource( name: "PPP.txt")!!.toURI())
    val lines = file.readLines()
    lines.flatMap { it: String
        val segs = it.split( ...delimiters: ",")
        val country= segs[0]
        (0 ≤ .. ≤ 18).map{ it: Int
            val year = 2010 + it
            val value = segs[it+1].toFloat()
            (country to year) to value ^map
        } ^flatMap
    }.toMap() ^lazy
}
```

Figure 16 Lazy loading PPPs from the database to increase access speed during message processing.

Now we have a way of determining what the cost of something will be in any given country if provided with the British price using PPP, we need a source of British seed prices. In this case a document [44] from Tamar Organics was used, as it contains a comprehensive list of seeds, while also providing all the data in an easy to parse Excel spreadsheet, unlike other sellers. If a seed doesn't exist in their roster, the selling price of a crop can be used instead, obtaining which is discussed in section 9.2.4.

However, the price of seed itself isn't useful unless the farmer also knows how much to purchase. Below is a standard formula [45] used to calculate seed rate – the amount of seeds that a smallholder should sow per unit area.

$$SR_{bc}\ (g\ ha^{-1}) = \frac{\frac{DPP}{ha} \times ISW\ (g) \times 3}{PP \times GP \times 4}$$

$$Where\ SR_{bc}\ (g\ ha^{-1})\ is\ Seed\ Rate\ (broadcast\ or\ continuous\ sowing)\ in\ grams\ per\ hectare$$

$$\frac{DPP}{ha}\ is\ the\ Desired\ Plant\ Population\ per\ hectare$$

$$ISW(g)\ is\ the\ Individual\ Seed\ Weight\ in\ grams$$

$$PP\ is\ the\ Purity\ Percentage$$

$$GP\ is\ the\ Germination\ Percentage$$

The desired plant population and purity percentage can be specified by smallholders according to their needs, but as a default the purity percentage has been set to 85%, accounting for possible impurities in the seed such as chaff and sand. The desired plant population can also be calculated according to the type of crop group the crop is in, and whether seeds are dibbled or broadcasted. This leaves individual seed weight and germination percentage still unset in the formula, and neither can be interpreted or calculated. This means we required another dataset.

The Seed Information Database (SID) is run by Royal Botanic Gardens Kew and Society for Ecological Restoration. It currently holds 54,453 records on germination and 88,902 records on seed data and is under a Creative Commons CC BY 2.0 license [46]. However, it has no CSV or dataset export, and requires searching by the scientific name, and then selecting the correct entry.

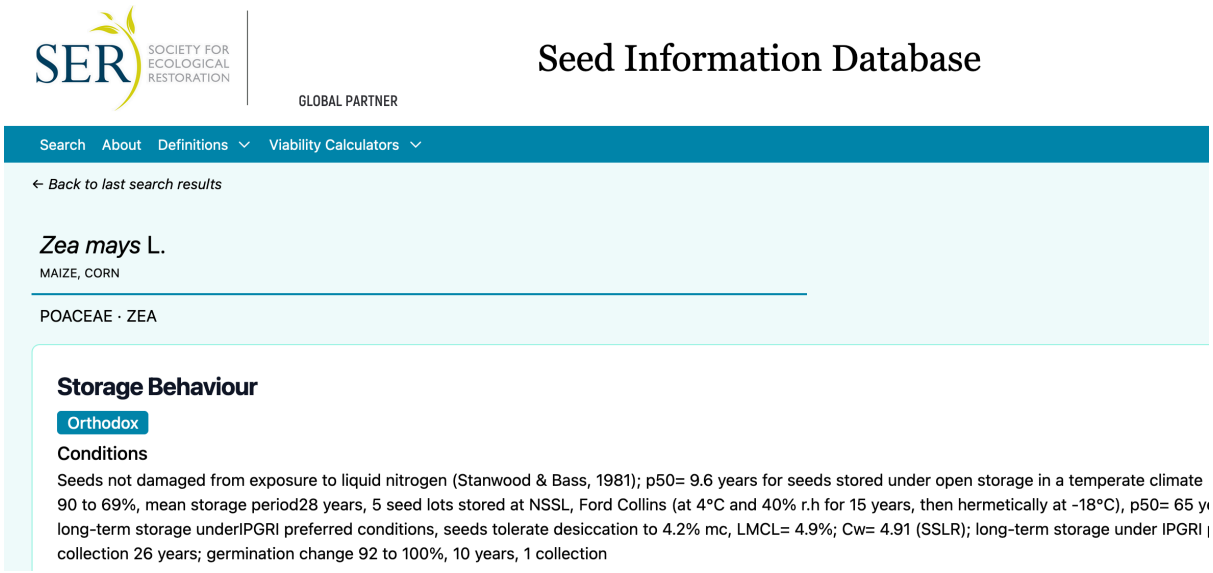


Figure 17 A search for Zea mays - commonly known as maize or corn [47]

Thankfully, all their requests contain their API key in the request header, meaning that the dataset can be queried legally due to the Creative Commons CC By 2.0 license.

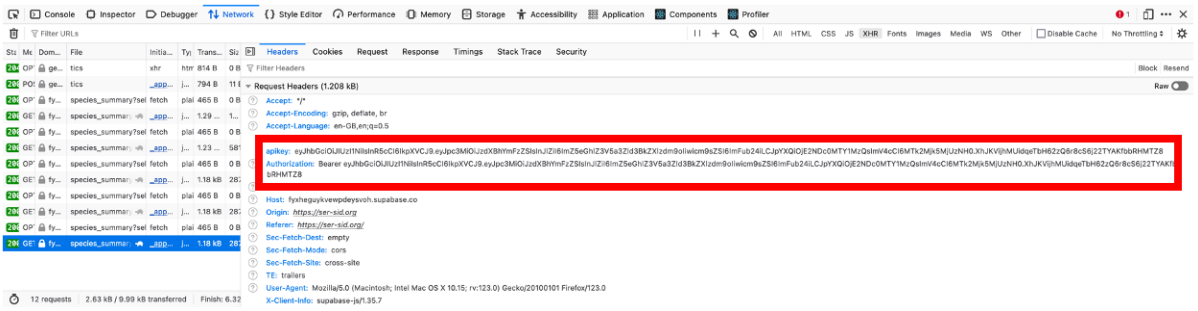


Figure 18 Inspecting the request made when searching for 'Zea Mays'. Note the API key and authorisation provided in the request header.

The request can be sent with different URL query parameters, along with a small amount of SQL injection for wildcard searching, to request everything and store it in the database, ensuring data does not need to be fetched every time. Under the permissions granted by the Creative Commons CC BY 2.0 license, this scraped dataset will be made into a RESTful API for other researchers to use.

With this data, which contains the individual seed weight and germination percentage for each crop, the seed rate can now be calculated from the formula. However, the formula is ideal and relies on ideal conditions, despite factors such as soil type and climate being known to affect the seed rate [48]. However, no formulas have been produced factoring this in, so instead, the seed rate calculated using the formula is

taken as a base, and the optimal condition for each crop is found (using ECOCROP, discussed in section 9.2.3). The ideal amount is then linearly increased using the assumption that the further away from these optimal conditions we go, as more crops fail, and crops produce less, producing a more accurate, realistic estimate for the smallholder. Even though this means initial outlay for smallholders may be greater, they will then have more successful produce to sell, recuperating costs.

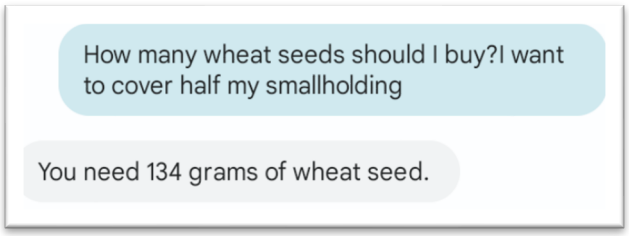


Figure 19 Real expert system output relating to buying seeds.

In Figure 19, we can see the seed rate formula being used in action. Here the smallholder has told the system before how large their smallholding is, and using information provided from the SID dataset, as well as the seed rate formula and linear increases, the expert system has determined that to cover half their smallholding, they will need to purchase 134 grams of wheat seed.

9.2.3 Planning, Growing and Harvesting

Initially, Trefle API was used to obtain general knowledge about plants for planning, growing and harvesting [49]. However, even with common crops, such as wheat, most fields are returned empty. This means important information such as row spacing, days to harvest and sowing conditions cannot be provided to smallholders. In an initial attempt to improve the data, we can query all sub-species and then amalgamate the results, with preference over the first entry (as more common species come first).

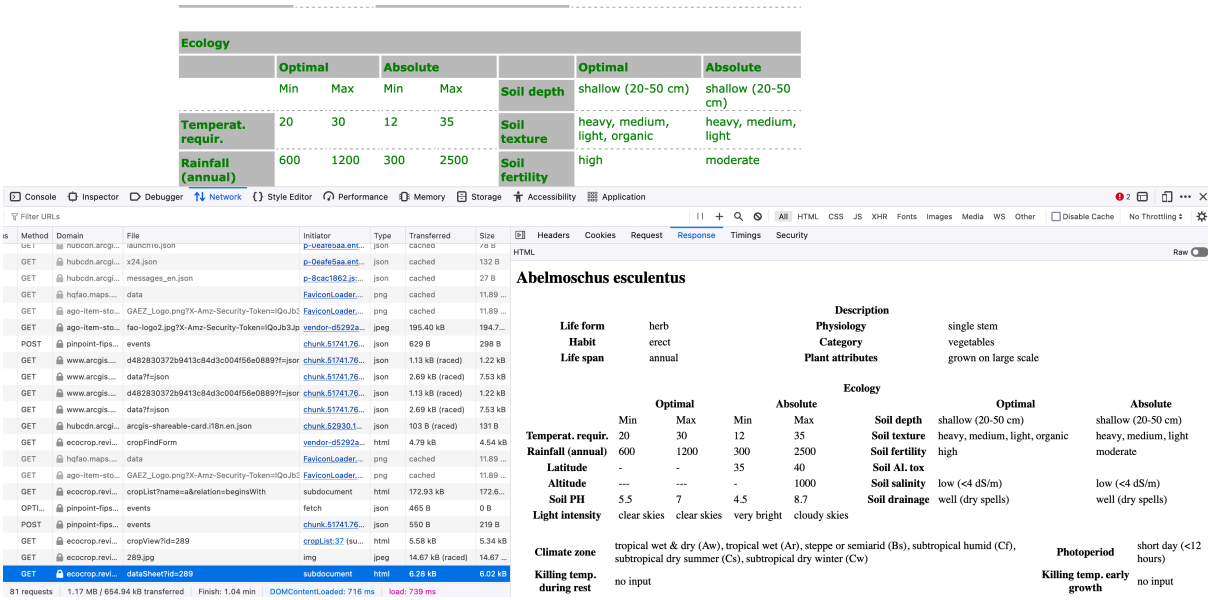
```
Jacqueline Dobreva-Skevington
final inline fun <reified T> defaultNoArgConstructor(): T{
    return T::class.constructors.first { it.parameters.isEmpty() }.call()
}


Jacqueline Dobreva-Skevington
final inline fun <reified T> List<T>.coalesce(): T{
    val newObj = defaultNoArgConstructor<T>()!!
    newObj::class.memberProperties.mapNotNull { it as? KMutableProperty<*> }.forEach {kProp->
        val firstNotNullOrNull = this.firstNotNullOfOrNull { kProp.getter.call(it) }
        kProp.setter.call(newObj, firstNotNullOrNull)
    }
    return newObj
}
```

Figure 20 Amalgamating Trefle results of all subspecies.

Figure 20shows how reflection is used to coalesce all the empty values into one object with all the information. First, the fields of the entity are looped over by name. The corresponding field is then fetched from the list of results returned by Trefle, and the first non-empty value is chosen. Finally, this is set on a blank version of the object, which is returned to the rest of the system.

However, on performing this amalgamation on wheat, only two extra fields are retrieved, leaving important fields such as *days_to_harvest* still null. As such, Trefle API can only be used as a summary grounding point for crops, and we need a more in-depth information source for further querying.

For this, the ECOCROP dataset can be used [50]. The ECOCROP dataset provides plant descriptors and environmental descriptors for crops that can be used to determine their ideal environment, and as such can inform the smallholder about a crops planning, growing and harvesting conditions. This data is once again under a Creative Commons CC BY 2.0 license, but the data is not available to download explicitly via CSV and does not provide an API interface.



Ecology

	Optimal		Absolute			Optimal	Absolute
	Min	Max	Min	Max	Soil depth	shallow (20-50 cm)	shallow (20-50 cm)
Temperat. requir.	20	30	12	35	Soil texture	heavy, medium, light, organic	heavy, medium, light
Rainfall (annual)	600	1200	300	2500	Soil fertility	high	moderate

Abelmoschus esculentus

Description

Life form	herb	Physiology	single stem
Habit	erect	Category	vegetables
Life span	annual	Plant attributes	grown on large scale

Ecology

	Optimal		Absolute			Optimal	Absolute
	Min	Max	Min	Max	Soil depth	shallow (20-50 cm)	shallow (20-50 cm)
Temperat. requir.	20	30	12	35	Soil texture	heavy, medium, light, organic	heavy, medium, light
Rainfall (annual)	600	1200	300	2500	Soil fertility	high	moderate
Latitude	-	-	35	40	Soil Al. tox		
Altitude	---	---	-	1000	Soil salinity	low (<4 dS/m)	low (<4 dS/m)
Soil PH	5.5	7	4.5	8.7	Soil drainage	well (dry spells)	well (dry spells)
Light intensity	clear skies	clear skies	very bright	cloudy skies			

Climate zone	tropical wet & dry (Aw), tropical wet (Ar), steppe or semiarid (Bs), subtropical humid (Cf), subtropical dry summer (Cs), subtropical dry winter (Cw)	Photoperiod	short day (<12 hours)
Killing temp. during rest	no input	Killing temp. early growth	no input

Figure 21 The ECOCROP website. As we can see, the response is provided in the form of an HTML file.

Instead, JSOUP can be used to extract the information. JSOUP is a Java library for data extraction and parsing for HTML files [51] – which can be used here as the response provided by the ECOCROP API requests is in HTML format, as can be seen in Figure 21. A two-step process is necessary for data collection. First, the API is called with combinations of the first letters of crops until all the IDs of the results are gathered, which can be used to get the URLs for all plants. These URLs can then be supplied to JSOUP, which parses the data and stores it in the database. This means we can avoid having to collect the data from the remote location each time it is required.

As with SID, under the permissions granted by the Creative Commons BY 2.0 license for the ECOCROP data, the dataset will also be made available via a RESTful API after the completion of the project for other users to benefit from as well.

```
private fun parsePage(document: Document): EcocropData {
    val content = document.getElementById( id: "content")!!
    val scientificName = getScientificName(content)
    val descriptionTableChildren = content.findTableByTitle( title: "description") Element
        .getElementsByTag( tagName: "tr") Elements
        .let { it.subList(1, it.size) } MutableList<Element>
        .flatMap { it: Element
            it.children()
        }
    val lifeform = descriptionTableChildren[1].text()
    val physiology = descriptionTableChildren[3].text()
    val habit = descriptionTableChildren[5].text()
    val category = descriptionTableChildren[7].text()
    val lifespan = descriptionTableChildren[9].text()
    val plantAttributes = descriptionTableChildren[11].text()

    val ecologyTableChildren = content.findTableByTitle( title: "ecology") Element
        .getElementsByTag( tagName: "tr") Elements
        .let { it.subList(2, it.size) } MutableList<Element>
        .flatMap { it: Element
            it.children()
        }
```

Figure 22 – JSOUP parsing of the ECOCROP webpage.

With the data from ECOCROP, generic information can be given to smallholders about each crop. However, wheat sown in Wales for instance, may have very different requirements to wheat sown in Algeria, meaning this data is not useful for smallholders, and doesn't meet our objective on tailoring information to a granular per-smallholding level. To be able to determine this information, we can attempt to find planning, growing and harvesting conditions in each country, and how that impacts each crop, and then correlate that to ECOCROP. Thankfully, this is unnecessary as many countries have similar requirements.

The standard grouping of countries for crops is via hardiness zones, based on dividing the world into 13 groups with similar minimum temperature boundaries [52]. However, hardiness zones do not consider other affective conditions, unlike Köppen climate zones which considers both temperature and precipitation, including when peaks and minimums of both are [53]. This makes Köppen climate zones a much more accurate representation of conditions for plants.

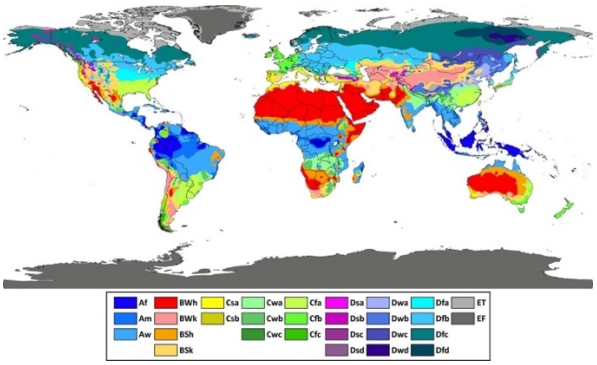


Figure 23 A visual representation of the Köppen Climate Zones [53].

The 5 main categories of the Climate system are: tropical climates, dry climates, temperate climates, continental climates, and polar climates, with further subgroups indicating aspects about specific weather and precipitation trends [53]. Köppen climate zones are also used by ECOCROP to denote the ideal conditions of plants, meaning it can also easily be corresponded to data we already have in the database.

Within each zone (for instance, Cfb for the UK), there are also other variations which may affect the conditions for plants. For instance, the clay soil in the West Midlands may take longer to warm up than sandier soils near the coast but may also retain heat and moisture for longer. There are also further weather distinctions. For instance, it may be forecasted to rain for a week in Llanelli and therefore may not be advisable to plant onion sets, but may be dry in Coventry, meaning onions can be planted, despite both cities having similar soil, and both being in the same Köppen Climate zone. As such, both Weather and Soil APIs are also integrated into the system.

Using the data provided by the soil, weather and climate APIs, and given a smallholding's exact location, we can correlate back to the ECOCROP ideal planting conditions and inform smallholders about crop planning, growing and harvesting requirements specific to their needs. This ensures that tailoring can be done on a per smallholding granularity, meeting the objective on tailoring and using real-time data by incorporating Agriculture-4.0 principles, in contrast to Tzachor et al stating that we cannot provide such tailoring yet.

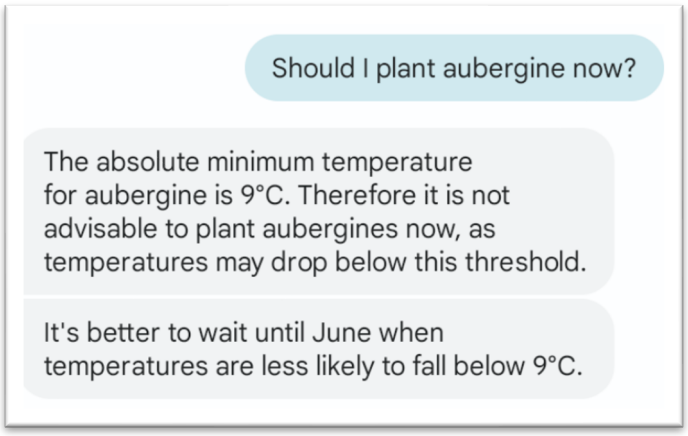


Figure 24 System output demonstrating use of both the Weather API and ECOCROP data.

In Figure 24 above, we can see the use of the Weather API, as well as ECOCROP data. Behind the scenes, the response is tailored specifically to Westwood Heath in Coventry, and is using data specific to that location.

Being able to determine the exact conditions in each area, as well as having access to optimal plant conditions from the Trefle and ECOCROP datasets, we can now refer to the buying seed formula in the previous section. It was mentioned that the formula was ideal and assumed optimal conditions. With this new data, a simple linear equation is set up, increasing the amount of seed that is needed the further away from its optimal conditions the system moves, thereby ensuring farmers can get the harvest they need.

9.2.4 Selling

The final part of the crop cycle is selling. A smallholder may want to know the price they may be able to sell their crop at when they are planning for their next growing season, so they know whether it is economically justifiable to plant the crop. They also need to be informed about future selling data at harvest/storage time, so they can determine when the best time to sell their crop would be. Therefore, given a dataset, we need to be able to extrapolate potential selling prices accordingly.

To calculate the price of selling the FAOSTAT crop price dataset is used as it provides a comprehensive list of the past prices of crops, by country. This is then used to estimate the prices which a user could sell their crops at, using a weighted linear average, with most recent data having a higher weighting to account for recent financial trends which may continue over the course of a crop cycle, such as inflation.

$$C(t_p) - C(t_n) = \frac{\sum_{i=1}^{n}\left(\frac{C(t_0) - C(t_i)}{t_0 - t_i} * (t_p - t_i)^{-k}\right)}{\sum_{i=1}^{n}(t_p - t_i)^{-k}}$$

$$Where\ C(t_n) = \ The\ cost\ of\ the\ crop\ at\ time\ n$$

An example of this formula in action is shown in Figure 25 for the predicted price of wheat in the UK. In the case where costing for a certain country does not exist in the dataset, the purchasing power parity discussed in the 'Buying Seeds' section can be applied to the data of a country we do know to determine the possible selling price in the country we do not.

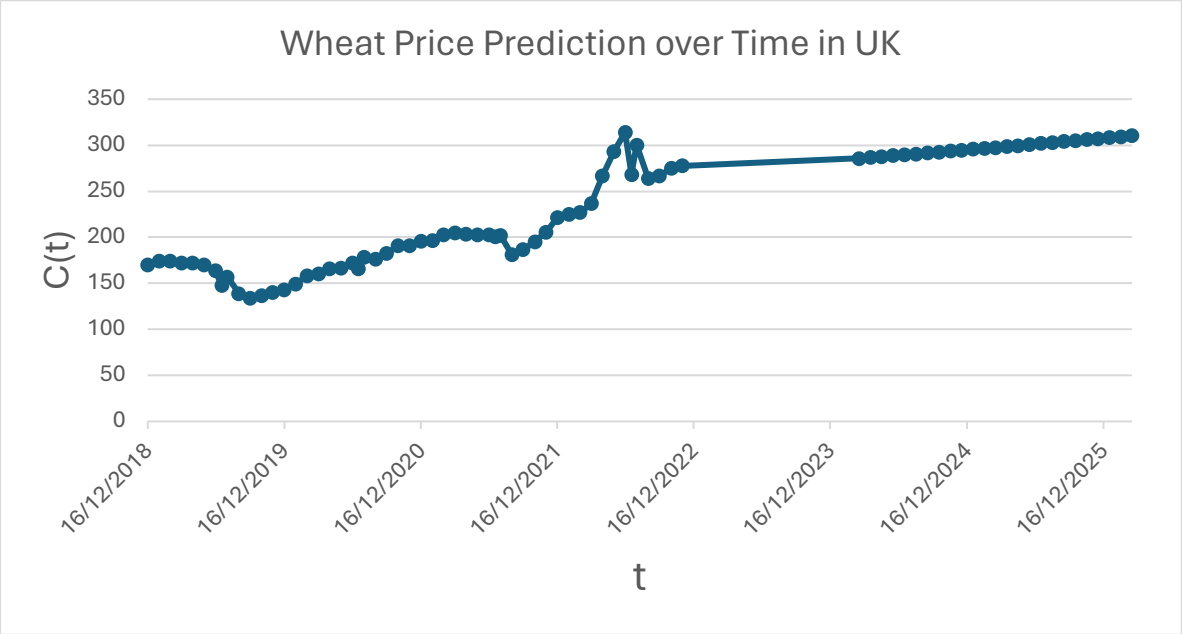


Figure 25 – Results of the weighted average – FAOSTAT data ends in 2022.

In application within the system, selling predictions are used to provide the smallholder with information both when asked a direct question regarding crop prices, and to add extra context to information to allow the smallholder to make an informed decision regarding their queries.

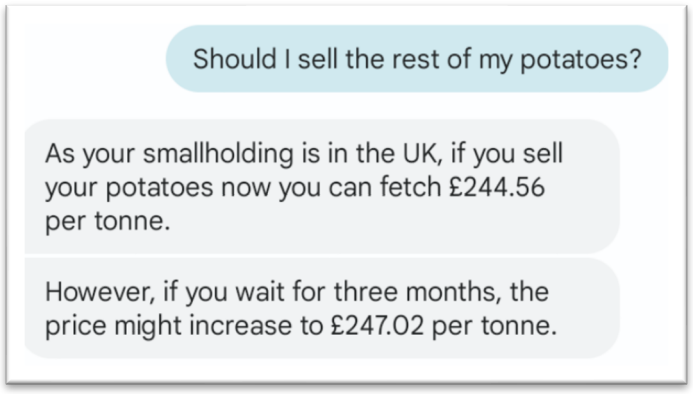


Figure 26 Using the selling data to provide price predictions to smallholders.

For instance, in Figure 26 we can see the expert system providing detailed information on the selling price of potatoes. The system offers the smallholder two pieces of information – one, based on current prices, and the other based on the local maximum the price is expected to go to within the crop cycle, allowing the user to make an informed decision.

9.3 RAG

Now we have all the data for the Expert System, it can be put together to form the context for the RAG. LlamaIndex was used [54] as the library with which to implement the RAG, due to its well-established user base and documentation. While other popular LLM libraries such as LangChain exist, LlamaIndex is specialised for RAG applications, and reduces the need for boilerplate code [55], thus speeding up implementation.

9.3.1 Kotlin-Python Interoperability

However, the project so far has been developed in Kotlin, the choice of which has been explained further in the 'Overall System Architecture' section later. LlamaIndex on the other hand, is a Python library, meaning the project is now dual-language and it is necessary to have interoperability between the two.

Interoperability for the project was set up using file-based communication for a lightweight solution. Starting from the Kotlin side, a specific method is linked to its Python counterpart using custom annotations. When called, this function first begins by packaging arguments passed to the function into a JSON format, which is then written as a file to a shared storage location. After this, the Kotlin process can start a Python child process and wait for it to complete.

```kotlin
@Service
class AgriculturalQuestionExtraction : PythonClass() {
    Jacqueline Dobreva-Skevington
    @PythonFunction( functionName: "getQuestions",  scriptName: "AgriculturalQuestionExtraction.py")
    fun getQuestions(userMessage: String):List<String?> {
        return execute(::getQuestions, userMessage)
    }
}
```

Figure 27 The Custom *@PythonFunction* annotation created for Kotlin Python interoperability.

During this call process, arguments are also supplied to the shared storage location. Once the Python process is launched, it deserialises the JSON function arguments, and tries to find a function which has been pre-registered for interoperability. This function can then be called, and the result returned in a serialised form to the shared file location.

```kotlin
kotlinInterop.registerFunction( name: "test_answer_relevance", test_answer_relevance)
```

Figure 28 Registering functions for Kotlin-Python interoperability in the Python file.

Meanwhile, the Kotlin process polls a shared output file to allow the programmer to assess what is happening in the Python file (such as print statements) in real-time. This polling finishes once the Python process terminates, and results are read back from the shared location, and deserialised using the Jackson library for Kotlin. Finally, any errors can be reported, and the result of the function can be returned to the main Kotlin program.

Using this file-based approach means there is a very low overhead when launching multiple processes, and it also allows for multiple Python processes to be launched concurrently, by using randomly allocated folder locations. This is imperative in an application where messages from the end users must arrive be processed in a parallel manner to cope with load.

This approach to Kotlin Python interoperability will be made available as a Gradle plugin after this project to allow other users to have a lightweight solution to dual-language projects as well.

9.3.2 Implementation

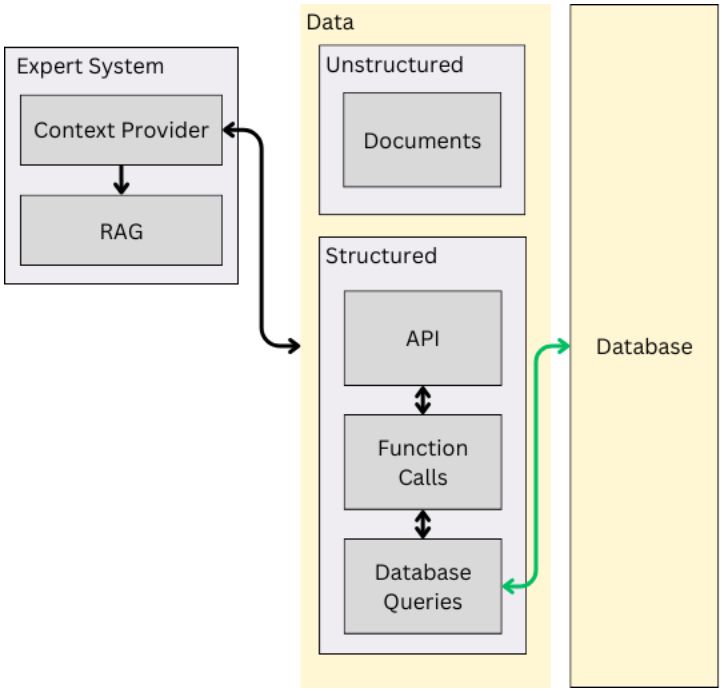


Figure 29 The Expert System in context.

Originally, LlamaIndex was going to be passed all the datasets collected in the sections above, which it would then use to generate the answer for a given question. For normal applications with a static dataset this would be fine. However, a requirement of the system is having real time data, and after some tests with a data dump, it was discovered that this is a very slow approach given that the data needs to be indexed every time a query comes for LlamaIndex to be up to date.

Similarly, in a production environment, the system may need to handle several thousand messages per minute. For the RAG to have the user's message history as context when responding to queries, the benefits of which were discussed by Tzachor et al, LlamaIndex will also need to be passed the updated message history of all the users whenever an expert answer was required. This was also discovered to hold up the entire system during stress testing, and consequently is an unfeasible approach.

Therefore, to include real-time dynamic data, a more complex methodology to providing context needs to be employed – a set of 'Context' providers need to be implemented that decide which subset of data to send to the RAG for any given message. The RAG can then index this data efficiently and use it to respond.

The first of these providers is the '*UserMessageContextProvider*' which allows the RAG to take advantage of the user's message history with the system. It constructs data for the RAG using a set format, for example, one datapoint may read 'User said <x> on date <y>'. This allows the RAG to differentiate between User and System messages and includes knowledge of when the message was sent. After this, a final data point can be added to inform the RAG that it is playing the role of the system. To show how this affects the responses, a simple example can be formulated with the message stream overleaf.

- User: “I would like to know how to plant tomatoes”
- System: <answer

---later---

- User: “Should I plant them now?”

Without the prior message history context, the conversion would continue with:

- System: “Empty Response”

However, with the context added, the conversion could instead go:

- System: “Yes, it is a good time to plant tomatoes now.”

As the system can now determine that the “them” in the user’s message refers to tomatoes, allowing for a continuous, natural feeling message stream, and thereby improving the usability of the system.

```
@Component
class UserMessagesContextProvider: InitialContextProvider {

    Jacqueline Dobreva-Skevington *
    fun Message.toContext(index: Int): String?{
        return when(type){
            MessageType.INCOMING -> "User sent message $index: " +
                    "\"$message\" at ${createdAt.format(DateTimeFormatter.ISO_DATE_TIME)}"
            MessageType.OUTGOING -> "System sent message $index: " +
                    "\"$message\" at ${createdAt.format(DateTimeFormatter.ISO_DATE_TIME)}"
            else -> null
        }
    }
    Jacqueline Dobreva-Skevington
    override fun contextForMessage(message: String, user: User): List<String> {
        return user.messages.sortedBy { it.createdAt }.mapIndexedNotNull { index, m ->
            m.toContext(index)
        } + listOf("You are System")
    }
}
```

Figure 30 The User Message Context Provider.

Along with this, data about the user’s situation are also provided to the RAG, for example the location and size of their smallholding, so that it can apply this to message generation. Once again, a good example of the usefulness of this is during a conversion regarding how many seeds to plant:

- User: “I would like to know how to plant tomatoes”
- System: <answer>
- User: “How many seeds should I plant?”

If the user smallholding details have not been provided, the conversion will go:

- System: “You should plant 0.07 kilograms of seeds per hectare”

However, if the user’s smallholding data, specifying they have 3 hectares of land, is added, the RAG will provide a more tailored answer:

- System: “You should plant 0.21kg of tomato seeds to fill your smallholding”

Once again, we can see the effect of including the users message history in the context – the RAG is aware of what kind of seeds the user is referring to by using the message context.

```
@Component
class UserSmallholdingContextProvider: InitialContextProvider {

    Jacqueline Dobreva-Skevington *
    override fun contextForMessage(message: String, user: User): List<String> {
        return user.userSmallholdingInfo.flatMap { it: UserSmallholding
            listOfNotNull(
                it.location_country?.let { "in the country $it" },
                it.location_city?.let { "in the city $it" },
                it.smallholdingSize?.let { "${it} hectares" },
                it.cashCrop?.name?.lowercase()?.let { "used to grow $it" }
            ).map { it: String
                "The user's smallholding is $it."
            }
        } + listOf("The user's smallholding information should be used to scale any results.")
    }
}
```

Figure 31 The User Smallholding Context provider.

As identified, one major issue with conversing with a user is that they will often use non-specific pronouns such as ‘they/them’ when referring to previous messages. However, it is pointless to send all the data to the RAG, as only the last two messages, or only the smallholding size rather than also location may be relevant. If we blanket send all the data to the RAG, we once again get to the point where the RAG has a lot of data to index and go through, when much of it may be pointless.
To get round this, the system marks some of the context providers as special ‘Initial’ context providers. This includes all the user’s smallholding information and message

history which can be provided as a preliminary context to the RAG. The RAG then uses this to generate a generic response, not using any of the data collected in the previous section. As it only has to deal with the message history and user smallholding information, the RAG is still quick at indexing the data and forming a response. This response is then concatenated with the original user's question to provide extra information about the question itself – like the smallholding size and the crop the message is referring to, and gets passed on the next context provider, which now can use that information we have about all the crops to answer the question specifically.

To see this in action, let us use the above example again:

- User: "I would like to know how to plant tomatoes"
- System: <answer>
- User: "How many seeds should I plant?"

As can be seen, the user's last message provides no clue by itself as to which crop they are referring to. The system first sends the initial context to the RAG, which replies "Just plant a few tomato seeds to start on your 3 hectares". This is then added to the message as context: <Context: "Just plant a few tomato seeds to start on your 3 hectares", UserMessage: "How many seeds should I plant?"> From this, the relevant information is provided to the next context provider, and the specific data for the question can be loaded in from real-time contexts, allowing the RAG to output a specific answer quickly, while considering all available context.

Using the context provided from the context providers, LlamaIndex can be called via the Kotlin-Python interoperability interface, which loads the context as an array of documents. This is then indexed using LlamaIndex's VectorStoreIndex allowing for efficient querying via the query engine.

```
def getAnswer(userMessage, inputDocumentData):
    # load the documents and create the index
    documents = [Document(text=s) for s in inputDocumentData]
    index = VectorStoreIndex.from_documents(documents)

    # query the index
    query_engine = index.as_query_engine()
    response = query_engine.query(userMessage)

    return str(response)
```

Figure 32 Tying it all together – the LlamaIndex part of the RAG.

9.4 Notifications

So far, we have been implementing a pull-based system, but one of the objectives is also incorporating a push aspect to it. Notifications are an effective way to remind smallholders to interact with the system, as well as providing them with vital, timely information. For instance, we can send smallholders event-driven notifications based on weather events that may affect their main crop.

This section goes through the implementation of both event driven and temporal notifications.

9.4.1 Implementation

Two types of notification were implemented during the project. 'Scheduled' notifications are generated in advance of their dispatch date, for example, when it is assessed that a user has planted corn, a notification for when they should harvest it is generated. These notifications include some brief details about the user and the crop that it relates to, and the time it should be sent, along with a prompt for the message pipeline to process. When the time to send the notification arrives, the system receives this constructed incoming message as if it was sent by the user, starting the process of generating the response and then providing the user with information.

The second type of notification is 'triggered' notifications. These are generated during an unexpected event, such as drought, and inform the user on how best to deal with the unforeseen circumstances. For example, in the drought case, the notification trigger service will identify the drought based on the data provided from the integrated weather API, and then provide the information to the user based on a pre-defined set of steps.

One base 'Event' interface can be implemented, and then specific events, such as a *DroughtEvent*, which implement the Event interface can be injected as a list into the notification triggering service. This means that when a new triggering event is added, there is no need to add it to a list of events to processed, as this happens automatically.

```
interface NotificationTriggerEvent {

    Jacqueline Dobreva-Skevington
    fun notificationBody(crops: List<Crop>, topics: List<Topic>): String
    val executionType: ExecutionType
    val delayDays: Int

    Jacqueline Dobreva-Skevington
    fun shouldTriggerNotification(time: LocalDateTime, location: Location): Boolean

    Jacqueline Dobreva-Skevington *
    fun getNotificationOrNull(
        time: LocalDateTime,
        location: Location,
        crops: List<Crop>,
        topics: List<Topic>
    ): String?{
        return if(shouldTriggerNotification(time, location)) notificationBody(crops, topics)
        else null
    }

    Jacqueline Dobreva-Skevington *
    fun execute(time: LocalDateTime,
                location: Location,
                crops: List<Crop>,
                topics: List<Topic>
    ): List<TriggeredNotification>{
        val possibleCropsAndTopics = executionType.filterCropsAndTopics(crops, topics)
        val possibleNotification = this.getNotificationOrNull(
            time, location, possibleCropsAndTopics.keys.toList(),
            possibleCropsAndTopics.values.toList()) ?: return listOf()
        return possibleCropsAndTopics.map { it: Map.Entry<Crop, Topic>
            TriggeredNotification(
                time.plusDays(delayDays.toLong()),
                it.key,
                it.value,
                possibleNotification
            )
        }
    }
}
```

Figure 33 Interface Events such as DroughtEvent implement which allow for triggered notifications.

9.5 Evaluation

9.5.1 DeepEval [56]

To evaluate the RAG quantitatively, DeepEval was used, an LLM evaluation framework. Below is a short example list of some of the questions used to test the system. The questions were selected to provide a broad coverage of the system functionality.

1) "How many tomato seeds should I plant?"
2) "What temperature is it best to plant aubergines at?"
3) "When should I sell apples?"
4) "How much could I buy orange seeds for?"
5) "When should I sow wheat?"
6) "Is now a good time to harvest my onions?"
7) "How much can I sell my apples for in a month?"
8) "What is the weather like in two days? Should I be outside planting?"
9) "Should I plant garlic when it is cold?"
10) "How long will my olives take to mature?"
11)

The Expert System was tested using the following evaluators provided by DeepEval:

- Answer Relevancy, which measures the quality of the RAG by checking how relevant the output is in relation to the input query.
- Toxicity, which makes sure the RAG isn't applying any toxic opinions to the answer such as dismissive statements or mockery.
- Bias, which checks whether responses from the RAG shows any opinions which are based on protected characteristics, such as racial, political or gender-based bias.

These evaluators were selected as they provided the greatest spread of testing both the technical proficiency of the output, but also whether the RAG is suitable to be used by actual smallholders – the RAG could be as accurate as possible, but if it is perpetuating misinformative stereotypes, or being unpleasant to interact with, then it is still unusable.

Each question was passed through the expert system to generate a response. The question-and-answer pairs were then provided to DeepEval, to evaluate the responses on the three metrics discussed above.
Overall, Toxicity and Bias passed by 100% on all results, and Answer Relevancy passed on 87.5% responses. Types of questions it failed on were ones such as "When should I sell apples?" as the answer generated by the system provided information on when apples should be ready to harvest alongside the market prices, and DeepEval determined that the crop cycle information was irrelevant to the response. Another example was the system providing weather information to the question “When should I plant cucumbers”, and DeepEval determining that knowing temperature conditions was irrelevant. All answers DeepEval produced a fail on were of such nature and none of the answers were factually incorrect or didn’t answer the question, but rather DeepEval determined they provided more information than necessary.

The RAG aims to provide farmers with explanations to provide a more transparent question-answering process, as well as educating farmers about the topic they have asked about. For instance, instead of answering “no” to the question “can I plant cucumbers now”, answering “no, cucumbers must be planted after the last frost date. The last frost date is predicted to be two weeks from now, and you can plant cucumber then”, as this facilitates the learning of the farmer.

However, despite this, the answer relevance of 87.5% is greater than the 80%±4% RAG accuracy Balagher et al. achieved on their paper comparing LLM and RAG in an agricultural context, so we can conclude overall the Expert System performs well on the quantitative metrics provided by DeepEval.

9.5.2 A Global System

To evaluate more subjectively whether the system is applicable globally, five family members in Bulgaria were asked to send the system questions once they’d shared their location with it. They were asked to send nine questions: three in January, three in February and three in March, about things that were going on in their gardens at that point to emulate a real user over a quarter of a year. Similarly, three family members in England were asked to do the same. In the middle of March, once they had submitted all 9 questions and read through the answers, they were asked to read through the conversational history again and answer the questions presented overleaf.

1) Did you feel like the answers received considered the time of year? Yes/No
2) Did you feel like the answers received were appropriate for your country? Yes/No
3) Did you feel like the answers received were appropriate for your city? Yes/No
4) Did you feel like the answers received considered the weather at the time? Yes/No
5) How accurate did you feel the answers were on a scale of 1-10?

Yes and no answers were normalised to 1 and 0 respectively, and then averaged and multiplied by 100 to produce a percentage of agreement. The responses were as follows:

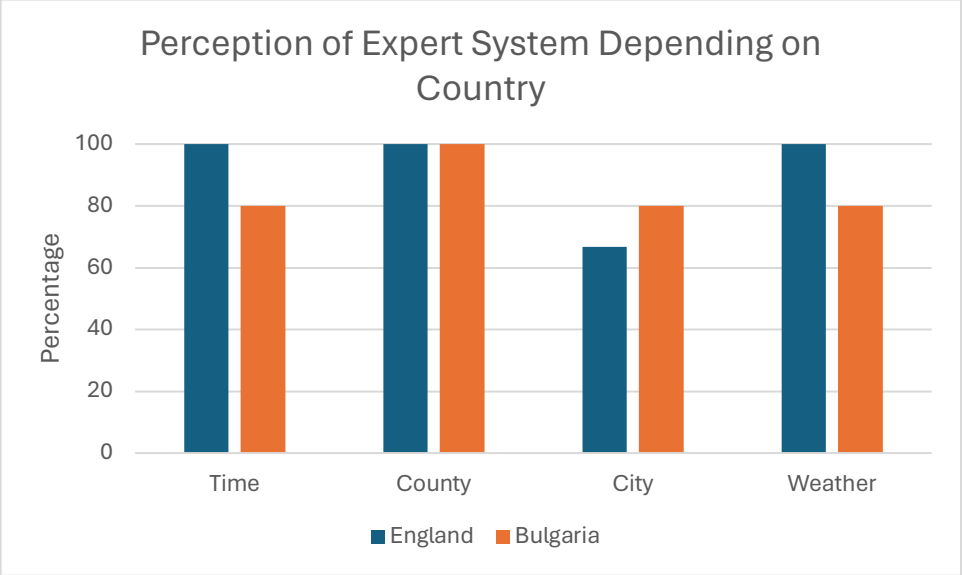


Figure 34 Comparing the expert system across two countries.

In terms of the final question, accuracy, the Bulgarian participants gave the system an accuracy score of 90%, whereas the English participants rated the accuracy at 86.7%. Overall, from the graph and accuracy scores, both English and Bulgarian participants had similar experiences of the system, thereby suggesting that the system is applicable globally. However, due to participant size limitations, no further evaluation could be done with locals in different countries to see whether the system is globally applicable in a more rigorous study. However, the initial results are promising, and certainly the output produced for the UK is as expected.

9.5.3 Other common LLMs

A final evaluation can be done on how the Expert System performs against other generic LLMs that have no additional context or finetuning on agriculture.

The 72 questions presented by the eight total participants from the previous study were run through OpenAI's ChatGPT and Google's Gemini. Each participant then received a copy of another participant's questions, as well as three different answers to each question, one answer produced by OpenAI, one answer produced by Gemini, and one answer produced by the system.

Question: How many tomato seeds should I sow?

Gemini: You should typically sow at least two tomato seeds per pot or module. This increases your chances of getting at least one healthy seedling to transplant later. Then, you can thin out the weaker seedling once they sprout.
ChatGPT: It depends on factors like space, variety, and desired yield. Generally, sow 2-3 seeds per planting hole to ensure germination.
System: You should plant 0.21kg of tomato seeds to fill your smallholding.

As the questions and answers were in two different languages, the Bulgarian participants and English Participants had to be kept in two separate groups, such that the first English participant (EP1) had their questions sent to EP2, EP2 to EP3 and EP3 to EP1. The same process was followed by the Bulgarian participants. This ensured that they didn't know the answers to the questions already and were not biased towards a specific answer.

To keep in line with the system, answers from the Bulgarian participants were first translated into English and then fed to the GAIs, before being translated back to Bulgarian. This was all done via LibreTranslate to maintain consistency with the system.

Each answer was presented in a random order for each question for each participant as to minimise bias. They were also asked to review each question and 3-answer pair in April, a month after they had last interacted with the system to give them the biggest possible opportunity to forget any mannerisms of the system that may make its answers easily identifiable. To each question and 3-answer pair, the participants were asked to rate which response they felt was the most informative.

An example of how the study works is as follows:

> EP1 receives EP3's nine questions, with three possible answers to each question. With each question to answers set, they are asked to choose which one they feel is most informative, writing down 1, 2, or 3 as their choice. They do this for all questions. At the end, their choices are sent in, and the numbers are decoded back to what produced the answer, for instance, if, for the first question, EP1 chose 2, this may be decoded to Gemini. All decoded answers from all participants are then collated, and choices tallied up between System, ChatGPT and Gemini.

As we can see from Figure 35, participants rated the System as producing the most informative response more than three quarters of the time, far ahead of both ChatGPT and Gemini.

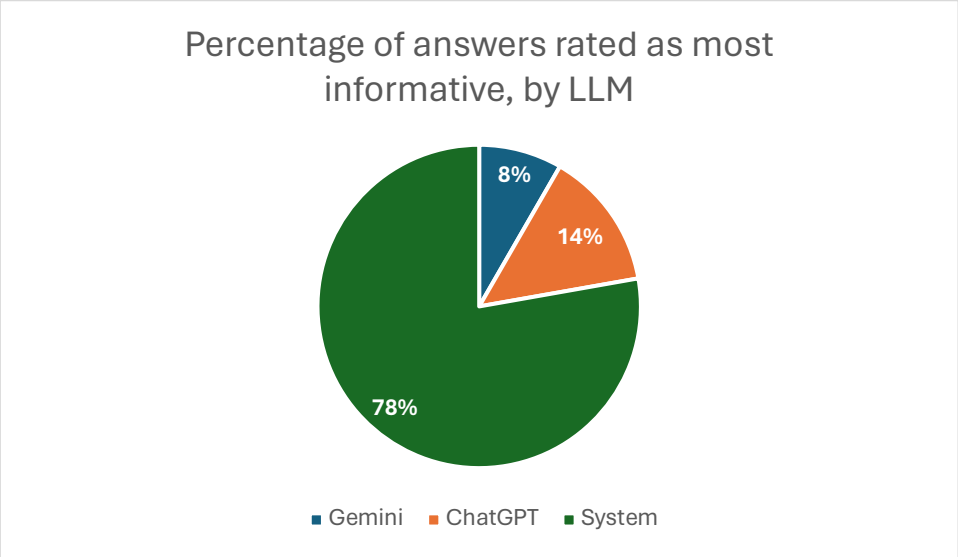


Figure 35 A pie chart representing the participants responses regarding informative responses.

When asked why they chose a different answer rather than the one produced by the system, the most common response was that they didn't feel the answer was applicable to their circumstances (for instance, the response by the system referred to a different type of weather than the one the participant had had at that time). This is something that is not replicable in a real use scenario, and therefore it can be hypothesised that smallholders in production would find the answer produced by the system the most informative one more than 78% of the time.

9.5.4 Summary

Overall, the project's Expert System is the first SMS-based AES that provides a global solution for the whole crop cycle, for thousands of crops, meeting the "one stop solution" objective. The data from the first qualitative study supports its applicability to a global context.

The use of a novel RAG based approach means Agriculture 4.0 principles can be utilised to deliver real time data tailored to the user's smallholding, unlike with Machine Learning best fit answer matching approaches with API querying on top, or fine-tuned LLMs, where Tzachor et al. stated such real-time tailored data was beyond our current capabilities. By having previous message history as context, the user can also ask secondary clarifying questions or provide more information, making the conversation have a more back-and-forth and therefore natural feel, as Tzachor et al. stated. Notifications also prompt the user to engage with the system and allow for timely information to be passed on. This proposed RAG approach is a promising solution to automated SMS-based AES.

The evaluation by DeepEval shows the RAG based approach is highly accurate, as well as producing no toxic or bias content meaning it is suitable for production. The second qualitative study also shows that people perceive the system to produce the most informative content when compared to other generic LLMs more than three quarters of the time.

The project has also had the opportunity to amass several datasets, which will be made into publicly available RESTful APIs for others to be able to use, as well as designing a lightweight Kotlin-Python interoperability, which will be made into a Gradle plugin.

The need to supply smallholders with accurate answers on how many seeds to sow, tailored on a smallholding-by-smallholding basis has also led to a suggestion on how the seed rate formula can be made to incorporate individual farmers circumstances and give them a much more accurate figure, using location and optimal crop-based data.

Overall, this part of the system successfully meets all the objectives set out at the start of the section.

10 Quantitative User Adaptation

Most aspects of the expert system require knowledge about the user – for instance, knowing their location to be able to extract soil type, relevant weather, optimal conditions of plants and other such factors. This knowledge may be provided by the user, and then can be stored in the database for future use and tailoring.

This section will go through how this information about smallholders is extracted so that agriculture-4.0 tailoring on a granular per-smallholding basis can be achieved, before evaluating the effectiveness of the approach taken.

10.1 Implementation

There are two possible approaches to extract relevant information from messages sent by the user.

The first approach is a non-NLP based approach, wherein given a set of characteristics (e.g. SIZE, NAME, LOCATION), a bag of words is produced (such as "acres", "meters", "hectares", "squared", "size" for SIZE). Any structure rules such as "name is/am called" before NAME, or a number before SIZE are also encoded. Incoming sentences are then searched word by word for a match in the bag of words, using an algorithm such as Jaro-Winkler for the edit distance (to account for singular/plural etc). If a word match is found, the preceding and following 'n' words are taken (where 'n' depends on rule the system is matching against) and are checked against a rule. If a match is found, the relevant information is extracted.

However, with this approach, shortfalls are easily found. Say the system receives a message "I would like to plant 2 acres of potatoes. How many seed potatoes do I need?". The system would match against "acres" and match the rule of a preceding number. If the user hasn't provided their smallholding SIZE yet, the system would take '2 acres' to be the size of their smallholding, and, after a conversion into hectares, store it into the database. More rules can be added to each section to deal with such scenarios, but due to the nature of natural language input, these rules could go on ad infinitum.

The second approach uses a Natural Language technique called Named Entity Recognition (NER). Instead of relying on pre-defined rules, NER can use a machine learning based approach that extracts textual data into predefined categories, incorporating surrounding context, and is therefore much more applicable for the

system, giving the wealth of different input the system could receive. NER typically involves two main steps: segmenting text into individual tokens using a tokenizer, and then performing entity recognition on the tokens by identifying patterns, linguistic features, and context.

Many common NLP libraries such as Spacy or NLTK provide out of the box NER solutions including categories such as People, Organisations, Locations, Products and Events. As we saw in the previous section however, the system however requires knowledge of a user's location (city and country), the size of their smallholding for calculations, their main crop for best advice. Knowing the user's name as well allows for a personal touch when responding to queries as well.

```
My smallholding in Swansea, Wales is 5 acres and I grow wheat. My name is Jacqueline.
{<UserDetails.SMALLHOLDING_SIZE: 'SMALLHOLDING_SIZE'>: '5 acres',
<UserDetails.MAIN_CROP: 'MAIN_CROP'>: 'Wheat',
<UserDetails.NAME: 'NAME'>: 'Jacqueline',
<UserDetails.LOCATION_COUNTRY: 'LOCATION_COUNTRY'>: 'Wales',
<UserDetails.LOCATION_CITY: 'LOCATION_CITY'>: 'Swansea',}
```

Figure 36 A user's message, and the corresponding extracted entities.

NER systems can be trained using supervised learning approaches, where they learn from annotated datasets containing labelled examples of named entities. As there are no training sets of data available, the dataset can be created using GAI, utilising various prompts to produce messages in chunks of 10. Some of these prompts are adverse - for instance "My smallholding is 1 km^2. I'd like to grow wheat on half of it. Can you please also send messages to my friend Jose, his smallholding is 3 acres.". The full list of prompts has been provided as Appendix A.

These messages can then be manually annotated using the NER Annotator for Spacy, which allows for custom entity tags to be created to label data with. The labelled training data can then be exported into JSON.

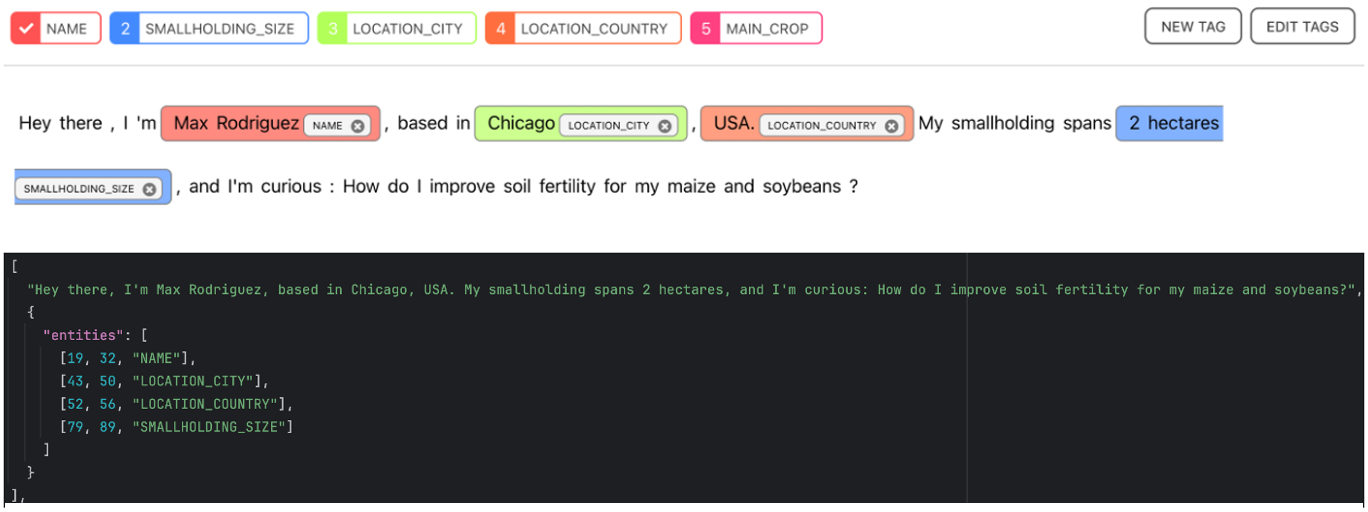


Figure 37 Using NER Annotator for Spacy on training data, and output JSON with tagged entities.

The exported JSON can then be converted into DocBin format via Spacy for NER training. This is split in 80/20 training to testing, in line with the recommended ratio provided by Spacy.

```
=========================== Initializing pipeline ===========================
✔ Initialized pipeline

============================= Training pipeline =============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00     51.59    0.00    0.00    0.00    0.00
  7     200         88.29   1551.06   95.11   93.86   96.40    0.95
 16     400         14.89    173.54   94.55   95.41   93.69    0.95
 27     600         19.34    127.03   95.11   93.86   96.40    0.95
 41     800         22.85    123.99   96.00   94.74   97.30    0.96
 58    1000         20.08    113.59   95.54   94.69   96.40    0.96
 80    1200         53.17    160.20   95.07   94.64   95.50    0.95
106    1400         63.54    177.88   95.11   93.86   96.40    0.95
138    1600         59.06    187.34   93.75   92.92   94.59    0.94
177    1800         10.54    119.49   95.07   94.64   95.50    0.95
224    2000         10.76    132.02   93.75   92.92   94.59    0.94
282    2200          5.13    146.05   92.44   91.23   93.69    0.92
348    2400          8.77    171.10   92.92   91.30   94.59    0.93
✔ Saved pipeline to output directory
Data/model-last
```

Figure 38 Terminal output when training model.

On training 96% accuracy was obtained in the 41-58th Epoch. This is the accuracy used by the system through the 'model-best' output. A 'model-last' is also produced, allowing for further training once real data has accumulated on the system.

```
# Import the spaCy library
import spacy
import sys

# Load the trained spaCy NER model from the specified path
nlp = spacy.load('Data/model-best')

fname = 'Data/test.txt'

# Open the PDF document using PyMuPDF (fitz)
with open("Data/test.txt", 'r') as file:
    text = file.read()
    doc = nlp(text)
    ents = doc.ents
    print([e.text for e in ents if e.label_ == "SMALLHOLDING_SIZE"])
    print([e.text for e in ents if e.label_ == "NAME"])
    print([e.text for e in ents if e.label_ == "LOCATION_CITY"])
    print([e.text for e in ents if e.label_ == "LOCATION_COUNTRY"])
    print([e.text for e in ents if e.label_ == "MAIN_CROP"])
```

```
ner-test ×
⋮
/usr/local/bin/python3.12 /Users/jacqueline.dobreva-skevington_t4c/Documents/Uni/AES-Q
['3 acres', '6-acre', '5 hectares', '4-acre', '2-hectare', '8 acres', '7-acre', '3-hec
['Daniel Wilson', 'Emily Davis', 'Sophia Smith', 'Liam Johnson', 'Olivia Brown', 'Etha
['Chicago', 'Sydney', 'London', 'Toronto', 'Auckland', 'Buenos Aires', 'Los Angeles',
['USA.', 'Australia.', 'UK.', 'Canada.', 'New Zealand.', 'Argentina.', 'USA.', 'Austra
['tomatoes', 'peppers', 'wheat', 'barley', 'oats', 'sunflowers', 'rapeseed', 'fruits',
```

Figure 39 Example of model-best test data. The terminal output displays the smallholding size, then the users' name etc as per the code.

10.2 Evaluation

On running the Spacy evaluator, the following results are obtained:

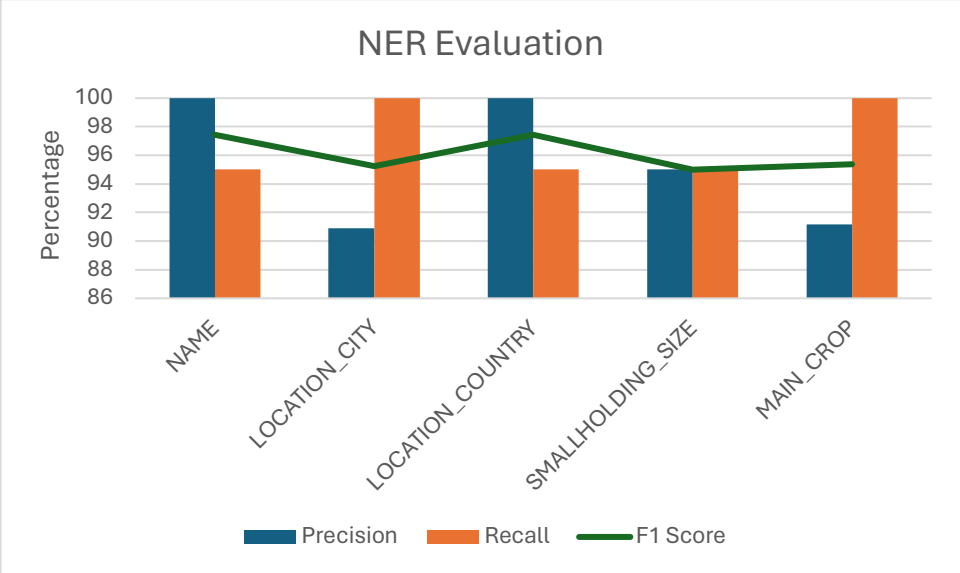


Figure 40 The precision, recall and F1 score for each named entity of the custom model.

Wherein precision denotes how many of the identified entities are correct, recall measures how many of the true entities are identified by the system, and F1 Score is the harmonic mean of precision and recall.

Overall, precision is 94.74%, recall is 97.30%, F1 is 96.00%, suggesting the model maintains a good balance between precision and recall. Speed is 1734 tokens per second, which indicates efficient performance. This is especially useful for our SMS-based AES as it is a real time messaging system, where message delays would result in poor user experience.

This NER based approach allows the system to extract entities accurately and quickly from user text, key for tailoring responses to each individual user. It also deals with adverse scenarios such as having two sets of information (e.g. “I would like to plant 2 acres of potatoes. How many seed potatoes do I need?”), through using surrounding context, unlike traditional programmatic rule-based approaches. This facilitates the meeting of the agriculture-4.0 objective by providing the user real time data tailored to their specific circumstances.

11 Qualitative User Adaptation

A leading issue in all AES implementations, although particularly prevalent in non-face-to-face solutions like SMS-based ones, is that given some information, smallholders don't act on it. Creating any AES is only useful if it has an effect. There are many reasons smallholders don't act on agricultural information given to them – government short-term drive policies and resource constraints being just some [57]. These are obviously outside the control of an automated SMS-based solution and require groundwork and government initiatives to overcome. However, there are other reasons we can challenge.

This section presents background on how the answer provided by the Expert System can be tailored, before synthesising the research into a series of objectives for the system. The project then presents the first application of user suggestibility in agriculture, taking a novel approach and tailoring based on knowledge and other user characteristics. This is all finally evaluated at the end.

11.1 Existing Research

Precision Development (PxD), mentioned as a leading deployed SMS implementation previously, has done some pioneering work in this area. They do not just develop technology, but also do A/B tests and conduct research to do with SMS AES, some of which is focused on how to improve their effectiveness. The have found that "small tweaks in messaging" and "customisation" increases engagement with their system – suggesting the need to tailor SMS-based systems not only quantitively (e.g. using weather or market rate data specific to a farmer's geography to increase accuracy of response), but also qualitatively – based on the user themselves. For instance, they found that in their 'Fall Armyworm' system (which was deployed to help farmers understand and deal with the fall armyworm pest), using urgent language increased engagement by 3%.

With the development of LLMs there is opportunity to tailor specifically to each user in an automated context, and thereby further increase engagement. Below two case studies are presented on other methods that can be applied to this system. Neither are specific to AES or Agriculture as there is a lack of research in this area focusing on user suggestibility but provide a good grounding of the background in this area.

11.1.1 Case study – eHealth [58]

eHealth is used by people over the world to inform, manage or exchange aspects of their health, and has seen an increase in use since the COVID-19 pandemic. However, users may fail to engage with the system, or misunderstand presented information. The study referenced states that previous approaches to communication have been "overly generic, impersonal, confusing, and boring", similar to the issues faced with AES. The study links research done within increasing effectiveness of student engagement in an educational context, stating that when students are more engaged, it increases their learning and participation (in this system's case, implementation of suggested practices), and also increases their perceived credibility of the educator – something also desirable within our system. The study suggests the following tailoring to increase engagement:

1) Using names to make messaging more personal.
2) Removing jargon in favour of easy to understand and clear messaging.
3) Provide opportunity for casual interaction to increase rapport.
4) Encourage interaction via personalised notifications.
5) Use context for elaboration and clarification.

Point 5 mirrors what was suggested by Tzachor et al., and has been achieved through the use of RAG. Sending personalised notifications has also been achieved. Point 1, 3 and 4 will be discussed further in the section 'Message Handling', where the message pipeline uses the user's name retrieved from the NER ('Qualitative User Adaptation') and can deal with a multitude of different message types, including casual interaction. The last remaining point, 'Removing jargon' is implemented as part of knowledge tailoring discussed further in this section.

11.1.2 Case Study - Voice [59]

We have already briefly discussed the concept of user mirroring in the Expert System section, when touching on Adaptive Decision Support Systems (ADSS), which suggested tailoring messages based on a user's knowledge level. Other mirroring can also be achieved, as discussed in the referenced study. The study focuses on voice and concludes with the need for voice tailoring based on age, gender, ethnicity, and accent using a study of 214 participants, a finding mirrored by other studies specific for chatbots on a lesser scale ('racial mirroring' [60] instead of holistic 'interface mirroring' as discussed in this study). It also discusses the need for human like interaction, stating

that when conversing with an automated system, participants were looking for something which had "human characteristics". The study concludes that "the majority of the participants perceived that a match between their own [...characteristics] is extremely important".

11.2 Objectives

Using the research discussed above, the system aims to provide a novel concrete implementation in the context of AES, aimed on improving the 'user suggestibility' of messages, increasing engagement and therefore effectiveness. The two key areas to be discussed are described in more detail below.

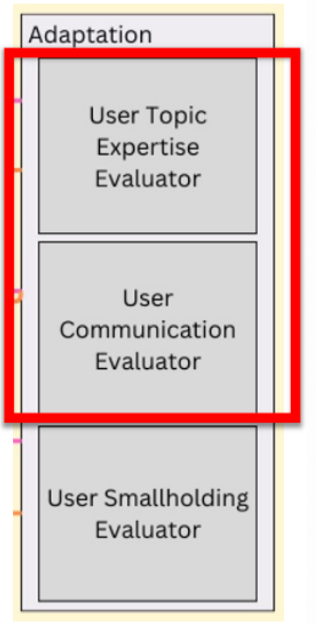


Figure 41 The part of the project architecture the discussion on the right refers to.

1) Tailoring based on users' knowledge level. This takes inspiration from both ADSS, which adapts its responses based on users' knowledge of company procedures, and the case study on eHealth. In practice, when the Expert System generates a response, the response should then be post-processed to match a user's knowledge level on each topic, so it doesn't 'bore' the user with information they already know but also doesn't use concepts and terminology the user doesn't understand ('removing jargon' – eHealth, point 2).

2) Tailoring based on users' characteristics such as age, gender, and literacy by performing user mirroring. For instance, messages should sound like they are written by someone of the same age, gender and literacy level as the smallholder to make them more likely to implement the suggestions – as discussed in the case study above.

11.3 Short-Text Authorship Characteristics

To be able to tailor based on user characteristics, we first need to compute them. The research area 'authorship characteristics' deals with this, with two main branches of study: one based on authorship mapping, i.e. given a piece of text, can we identify who the author is from a given corpus of known authors? The other is based on characteristic extraction such as what the age of the user who wrote a piece of text is.

Research in these areas is usually limited to books, documents, or emails, in descending order of size, due to practical applications in plagiarism, phishing, or forensic matching, however the application to methodologies suggested in this research to shorter text, such as Tweets, is a well-known problem [61]. The shorter the text, the less corpora there is to perform analysis and matching on, and therefore the more difficult it gets and the lower any accuracy is. The project attempts to overcome some of these issues; however, solving this short-text authorship characteristic problem is something which needs its own dedicated research on, and is far beyond the scope of the project.

Given the issues in smallholders implementing suggestions, it is hoped this user suggestibility concept provides motivation for more research into this area.

11.4 'Tailoring based on a users' knowledge level' - User Topic Expertise Evaluator

To classify a user's expertise in a topic, we first need to decide what a topic is. In the expert system messages are classified by crop, and part of the crop cycle, therefore a similar approach is taken here for consistency.

Crops and the crop cycle are organised into a knowledge area matrix, with each crop and crop cycle pair (e.g. <RICE, GROWING>) having an associated knowledge level value.

crop	topic	knowledge_level	user_id (UUID)
MAIZE	GROWING	0.5074994375506203	9d802da2-418e-46e0-a234-…
WHEAT	GROWING	0.4574994375506203	9d802da2-418e-46e0-a234-…
RICE	SELLING	0.5274723043445937	9d802da2-418e-46e0-a234-…
BARLEY	PLANNING	0.6378998797596874	9d802da2-418e-46e0-a234-…

Figure 42 Part of the user_knowledge_area table showing some knowledge areas and a user's associated knowledge level.

A basic approach can be taken to calculating a user's knowledge in each knowledge area using the assumption that the more questions asked about the topic, the more is unknown. This assumption of course has some flaws – if the user, for instance, is in the UK, and starts to use the system in winter, they may be interested in knowing when to sow wheat or potatoes, and they won't be asking about, say, how to preserve tomatoes, as that is something that is only relevant to late-July/August. Therefore, this is a system that gets more accurate over time, as more crop cycles are completed.

To demonstrate this, let us introduce three knowledge areas – *KA1*, *KA2*, *KA3* – where *Knowledge*(*KAn*) is a number between 0.0 and 1.0 where 0.0 denotes a low knowledge of a given are and 1.0 denotes a high knowledge. *KA1* and *KA2* regard processes that

occur during the winter. Say in the first half of the first year, the smallholder asks lots of questions about *KA1*, and therefore we assume *Knowledge*(*KA1*) is 0.1. In the second half of the year, they ask questions about *KA3*, making *Knowledge*(*KA3*) 0.3. As the total number of messages has also increased (see formula below), *Knowledge*(*KA1*) is now 0.4. In the first half of the second year, the user becomes interested in a new crop they have never grown before and asks lots of questions about *KA2*, as well as reminding themselves about some aspects of *KA1*. *Knowledge*(*KA2*), due to the volume of questions they asked is 0.1, *Knowledge*(*KA1*) stays at 0.4, and *Knowledge*(*KA3*) increases to 0.4 as well. As we can see, the more years that pass, the knowledge of each area begins to even out, and sudden blips in interest also swing results less, meaning that the knowledge of all areas is more accurate overall.

	Year 1, first half	Year 1, second half	Year 2, first half
Knowledge(KA1)	0.1	0.4	0.4
Knowledge(KA2)	-	-	0.1
Knowledge(KA3)	-	0.3	0.4

Table 3 A visual illustration of the example provided above. Dashes mark undiscussed knowledge areas for clarity, but in the system, they would be defaulted to 0.0.

To calculate the knowledge of a given area, the following formula is used:

$$Knowledge(Area) = 1 - \frac{2}{1 + e^{-0.2(Q_t(Area) - Q_t)}}$$

$$Where\ Q_t\ is\ the\ total\ number\ of\ questions\ the\ user\ has\ asked$$
$$and\ Q_t(Area)\ is\ Q_t\ asked\ about\ a\ specific\ area$$

The equation is based on an inverse sigmoid function, bounding knowledge between 0.0 and 1.0 where 0.0 is the least knowledgeable and 1.0 is the most. Using a formula based off a sigmoid function also nicely models how knowledge works, where it is very difficult to know absolutely everything about a certain knowledge area. Figure 43 provides a visual example of how the formula works, when Q_t(Area) is 5. The x-axis represents the total number of questions asked by the user (Q_t), and the y-axis represents the user's knowledge. The leftmost point (5, 0) shows how if the user has asked 5 questions about the topic, and they have only asked 5 questions in total (i.e. all their messages have been questions about this topic), we can assume the user knows very little about this area (0). In comparison, the second point shows a scenario where the user has asked 5 questions about the topic but has 15 messages in total – here the

user's knowledge is calculated as 0.462, still below average (0.5), but much more middling.

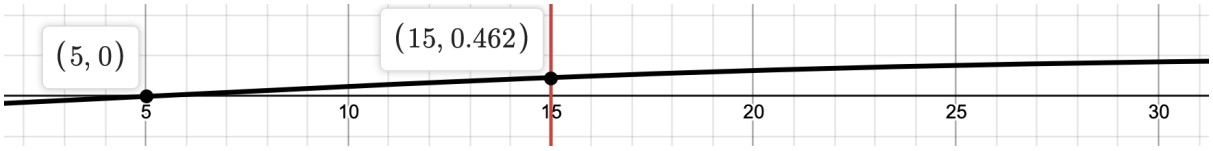


Figure 43 A graph showing the equation at Qt(Area) = 5

As responses from the Expert System contain the most detail, as responses have not yet been summarised by post-processing, they are suitable for someone with no knowledge in an area. Therefore, a default of 0.0 is set for knowledge areas which the user has asked no questions about.

	Crop cycle			
Crop		Buying	Selling	...
	Maize	0.73	0.0	...
	Rice	0.49	0.56	...
	...	...	...	...

Table 4 An example Knowledge Area Matrix with some example crops and areas of the crop cycle. The matrix is filled with sample values denoting the user's knowledge.

However, everything discussed above has been made on the assumption that once a user asks a question and receives an answer, they do not forget. This of course, is not the case. To overcome this, Ebbinghaus' formula for the curve of forgetting needs to be introduced to the calculation.

$$R = e^{-\frac{t}{S}}$$

Where R is retrievability, t is time and S is stability of memory, variable to topic

And the overall formula can then become:

$$Knowledge(Area) = 1 - \frac{2}{1 + e^{-0.2(Q_t(Area) - Q_t)}} * R$$

Such that the more time that has passed since last interacted with a specific knowledge area (e.g. the last time a question has been asked about growing rice), the more the user will likely have forgotten (following Ebbinghaus' curve) and therefore the lower the knowledge of a specific area should be.

11.4.1 Getting a base for S

The S in Ebbinghaus' formula depends on the task someone is performing. For instance,

someone witnessing a once-in-a-lifetime moment may have a much larger memory stability of that moment than someone asked to recall what the first sentence of a book they just read is. To determine the value of S in a situation where users are likely to engage with the content only for a few seconds before moving on, nine members of family and friends were asked to memorise answers to nine simple questions. The answers only involved one factor. For instance: < "How deep should I plant corn?", "2.5cm" > is a question answer pair with one factor, whereas < "Should I water corn?", "Water corn only when temperatures are above 25 degrees, and it has not rained for 5 continuous days" > has an answer with two factors, as underlined.

1) Q: When should I plant corn? A: June.
2) Q: Will cobs ripen is partial sun? A: Unlikely.
3) Q: Can I grow corn in dry or heavy clay soil? A: No.
4) Q: What temperature does corn best germinate at? A: 20 degrees.
5) Q: How far apart should seeds be sown? A: 40cm.
6) Q: How is corn pollinated? A: By wind.
7) Q: How do you know when a cob is ready to harvest? A: The tassels are brown.
8) Q: When are corn kernels liquid inside? A: Not yet ripe.
9) Q: How can you prevent corn plants from blowing over? A: By mounding soil up.

Each participant was given 2 minutes to memorise the answers, equating to just over 13 seconds per question – to simulate the amount of time a user may spend engaging with each answer the system generates in response to an agricultural question. Each participant was then asked one of the nine questions. The number of participants that answered the question correctly was recorded. Each following day, at the same time (as a control), the participants were asked another question, such that they were never asked the same question twice (as that would affect the result as they would remember the answer better) and that no two participants were asked the same question on the same day (to remove possible bias from 'easier' to remember question and answer pairs). In essence, the order looked as follows:

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9
P1	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
P2	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q1
P3	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q1	Q2
etc.	...								

Table 5 Rotating questions amongst participants. (P1 = Participant 1 etc).

At the end of the study, the results were plotted on a line graph shown below. We can see how on Day 1, most participants (8/9) recalled the answer. However, by Day 5, more than half could not remember.

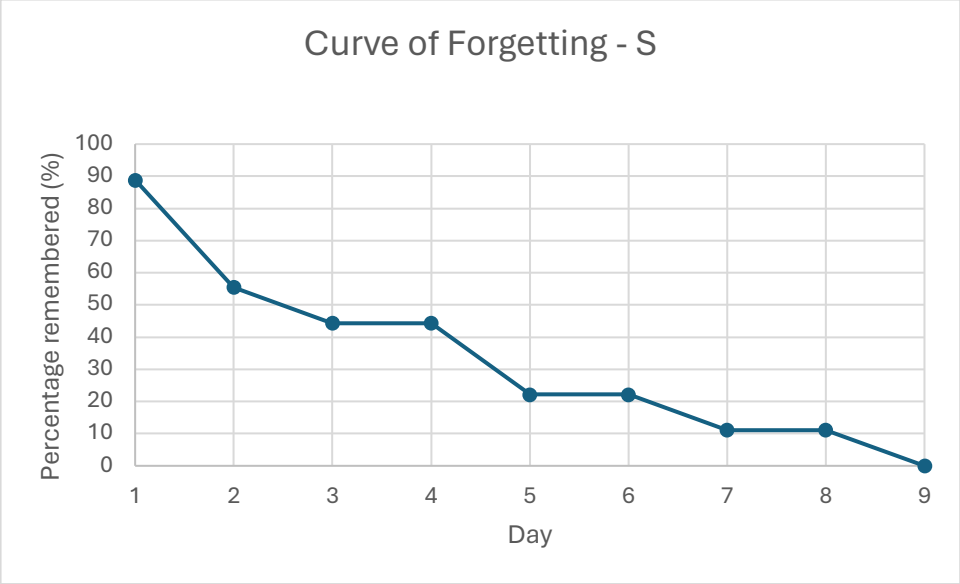


Figure 44 A line graph displaying the percentage of participants who recalled the answer to each question as time goes on.

It is also worthwhile noting that participants over 50 performed worse throughout the entire study, especially in the last three days. This provides further opportunities for tailoring.

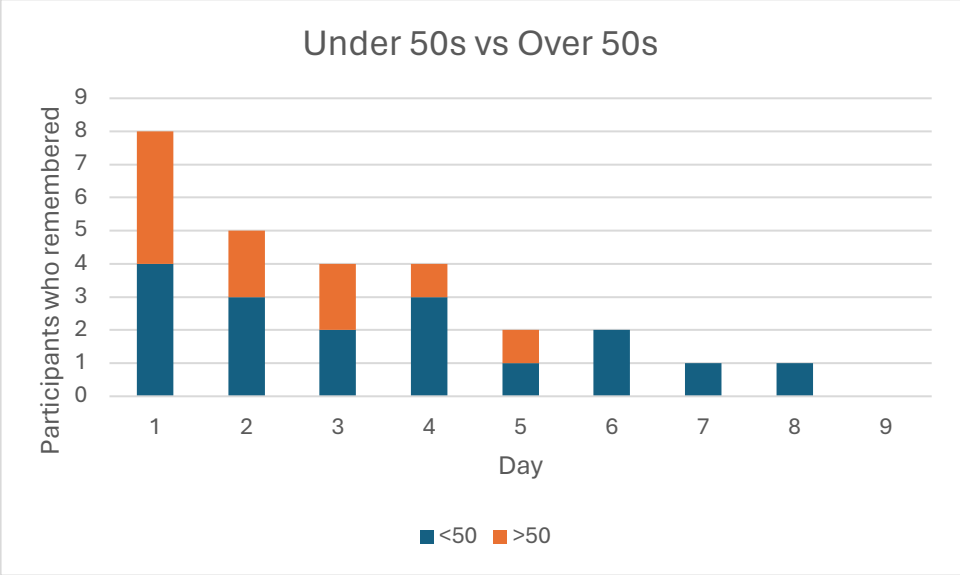


Figure 45 A stacked bar chart displaying the overall participants who remembered the answer divided into over 50 and under 50 categories. Participants over 50 on average performed worse than those below.

The curve of forgetting obtained during the study can now be compared to variations of Ebbinghaus' curve of forgetting obtained by tweaking the parameter 'S'. As seen below, the curve with S = 3.0 provides the best fit.

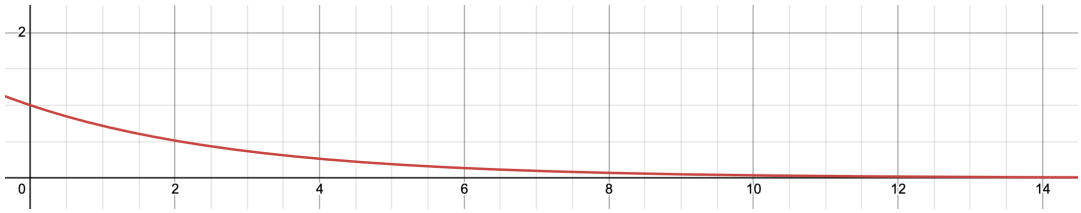


Figure 46 Ebbinghaus' curve of forgetting, with s = 3.0

Given the observation about age, one curve of forgetting does not fit all users. It can also be assumed that age is not the only parameter that affects the rate of forgetting. Therefore, further tailoring can be achieved by tweaking the S for each user starting from the base value of 3. This means that each message the user receives is tweaked exactly according to their characteristics.

```
/**
 * Different users will have different half lives. This provides a rudimentary example calculation based on a
 * users age and literacy level.
 *
 * @param age - the users age
 * @param literacy - the users literacy level
 * */
 Jacqueline Dobreva-Skevington *
fun calculateHalfLife(age: Age, literacy: Float): Double {
    val baseHalfLife = 3.0 // Base half-life

    val ageHalfLife = when (age) {
        Age.YOUNG -> baseHalfLife * 1.2
        Age.ADULT -> baseHalfLife
        Age.AGED -> baseHalfLife * 0.8
    }

    val literacyAdjustment = -(0.5 - (literacy / 100.0)) // Convert literacy to a factor between 0 and 1
    val literacyHalfLife = baseHalfLife + literacyAdjustment

    return (ageHalfLife + literacyHalfLife) / 2.0
}
```

Figure 47 Tweaking the 'S' value for each user according to their characteristics - an example of granular tailoring.

The function takes the base half-life as determined by the study (S = 3) and tweaks it based on age, with younger people forgetting over a longer period than older users. It also adds a number between 0 and 1 based on the user's literacy, with this parameter leaning towards 1 the more the calculated literacy rate of a user is. These two tweaks are then averaged to produce the tailored 'S' for each user.

To show the effect of this tweaking, the two figures below are enlarged images of graphs around the point where x (the number of days elapsed) is four. Each graph displays the effect of tweaking only one aspect, with the other one being left as the average (e.g. a literacy level of 0.5 or an age of 'ADULT') In the first graph, we can see that a younger person has retrievability of 0.33 at the 4^{th} day, whereas an average person has one of 0.26, and someone older has 0.19. Similarly for literacy, the green line shows a user with the highest possible literacy rate, then an average user and finally, the red line denotes the user with the lowest possible literacy rate.

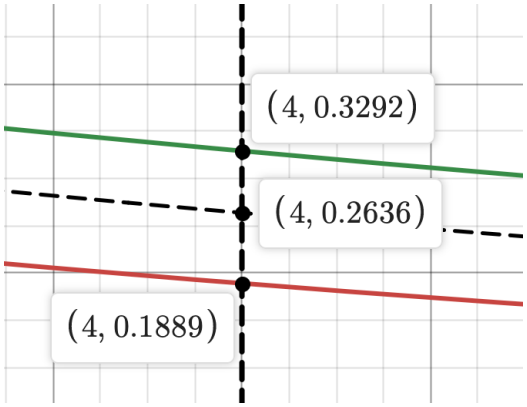


Figure 48 Tailoring on age alone.

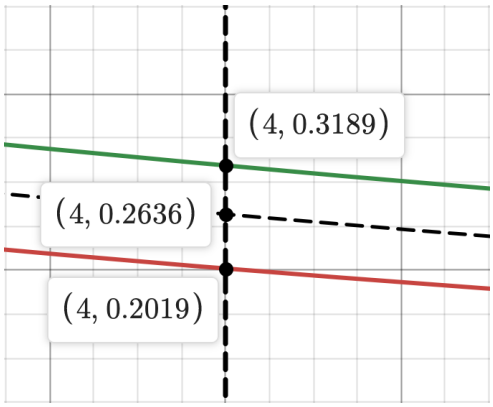


Figure 49 Tailoring on literacy alone.

Putting everything together in the context of the system, all of a user's messages are retrieved (Qt), and grouped together by knowledge areas (Q_t(Area)). 'R' is calculated, along with the base formula for Knowledge(Area), before they are multiplied together and saved back to the repository. Implementation-wise, the function is scheduled to occur once a night, as it is a computationally intensive task, and is marked as *@Transactional* so the database is rolled back to its previous state is the function fails, increasing safety.

```kotlin
/**
 * Calculates a user's expertise on a given Knowledge Area (KA) given all their messages.
 * This basic model assumes that the more questions asked about a KA, the more the user doesn't know.
 * This gets more accurate over more and more crop cycles.
 *
 * Ebbinghaus curve of forgetting is also included as the longer it's passed since the user has asked a question
 * about a particular topic, the more they will have forgotten about it.
 * */
 Jacqueline Dobreva-Skevington *
@Scheduled(cron = "0 0 1 * * ?")
@Transactional
fun calculateUserExpertiseOfTopicAndCrop() {
    userRepository.findAll().forEach { user ->
        val messages = messageRepository.getMessageByUserIdAndType(user.id)?.map { it: Message
            it.messageTopics.firstOrNull()
        }

        messages?.groupBy { it?.knowledgeArea }?.mapNotNull { it: Map.Entry<KnowledgeArea?, List<MessageTopics?>>
            (it.key ?: return@mapNotNull null) to it.value.size
        }?.sortedByDescending { it.second }?.onEach { it: Pair<KnowledgeArea, Int>

            val lastInteractionTime = user.lastInteractionTime[it.first] ?: it.first.modifiedAt
            val decayFactor = calculateDecayFactor(user, lastInteractionTime, LocalDateTime.now())
            val scaledKnowledge = Utils.scaleProbability(
                it.second.toDouble(),
                messages.size.toDouble(),
                inverseResult: true
            )

            if (!user.knowledgeAreas.any { ka ->
                    ka.topic == it.first.topic && ka.crop == it.first.cropName
                }) {
                val ka = userKnowledgeRepository.save(
                    UserKnowledge(
                        user.id,
                        knowledgeLevel: scaledKnowledge * decayFactor,
                        it.first.topic,
                        it.first.cropName
                    )
                )
                user.knowledgeAreas.add(ka)
            }
        }
    }
}
```

Figure 50 Calculating Knowledge Levels.

11.5 Knowledge Classifiers

To perform tailoring based on a user's knowledge areas, messages must be classified into knowledge areas first. Each incoming message may have multiple questions, each to do with different knowledge areas. Therefore, messages in the database have a many-to-one relationship with knowledge areas.

```
/**
 * The knowledge areas associated to a given message
 * */
@Id
@ManyToOne
val knowledgeArea: KnowledgeArea = KnowledgeArea(),
```

Figure 51 Part of the *MessageTopic* entity mapping to knowledge areas.

As discussed previously, each knowledge area is made up of a crop, and a topic, so to put each message into a knowledge area, we must know the messages crop and topic first. To do this the system uses a zero-shot classifier model from HuggingFace.

Zero-shot classification can be thought of as an instance of transfer learning, where a model does something different to what it was originally trained to do. A zero-shot classifier is trained on labelled data, and then, provided a sentence and labels it hasn't seen before, it can categorise which of those labels match the sentence best. The model is taught to generalise to unseen classes by learning the relationships between different classes or categories based on semantic embeddings or textual descriptions.

In zero-shot classification, each category is associated with a textual description or some form of semantic representation. During training, the model learns to map input data to these semantic representations, enabling it to classify unseen classes at inference time. This approach leverages the semantic similarities between classes to make predictions for new categories.

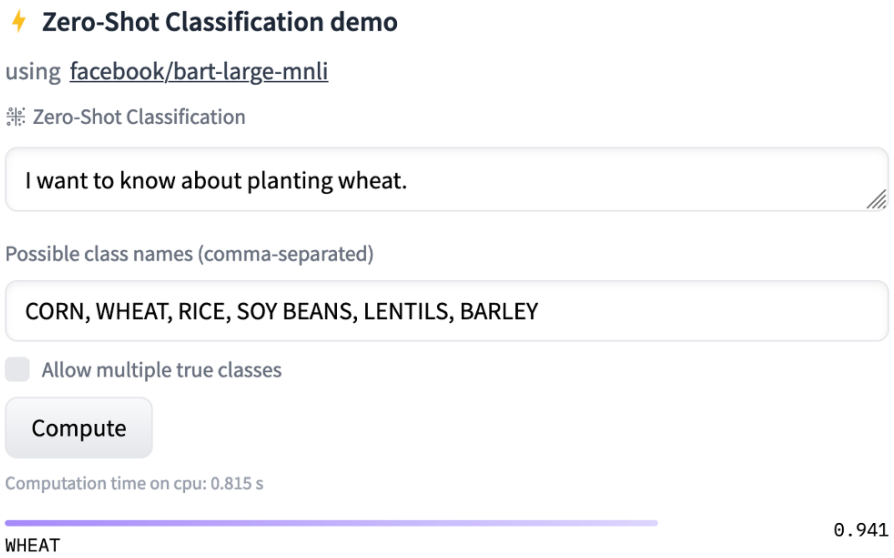


Figure 52 Zero-shot classification demo on HuggingFace. The project uses the model displayed here.

Zero-shot classification is particularly useful for cases where it would be impractical to produce a labelled dataset - as with this project. Messages can have many permutations, and the list of crops and topics can expand to beyond initial training data, meaning a traditional model would have to be retrained frequently.

An interface to the model is implemented in Python, leveraging Kotlin-Python interoperability. Each incoming question in each message is run through the model for both crop and topic, producing potentially multiple knowledge areas for each message, ensuring that knowledge level calculations for each user can be performed.

```
fun tagMessage(question: String, userId: UUID): KnowledgeArea? {
    val crop = getCropOfMessage(question)
    val topic = getTopicOfMessage(question)

    if (crop != null && topic != null) {
        return knowledgeRepository.findByIdOrNull(KnowledgeAreaId(topic, crop))
            ?: knowledgeRepository.save(KnowledgeArea(topic, crop, LocalDateTime.now())).also { it: KnowledgeArea
                val message = messageRepository.findFirstByUserIdAndTypeOrderByCreatedAtDesc(userId)!!
                val topics = messageTopicsRepository.save(MessageTopics(it, message))
                if (!message.messageTopics.any { it: MessageTopics
                    it.sms.id == message.id &&
                    it.knowledgeArea.topic == topic &&
                    it.knowledgeArea.cropName == crop
                }) {
                    message.messageTopics.add(topics)
                }
            }
    }
    return null
}
```

Figure 53 Classifying each part of the message into appropriate Knowledge Areas.

11.6 'Tailoring based on users' characteristics' - Other Evaluators

This section deals with the second objective for user-based tailoring - tailoring by user mirroring.

11.6.1 Age and Gender

Both the age and gender evaluators use premade models on Hugging Face. The *Abderrahim2/bert-finetuned-Age* evaluator achieved an accuracy of 72.49% on the evaluation set, whereas *padmajabfrl/Gender-Classification* reportedly achieved an 100% accuracy on the training set at the 5th epoch. Both models use BERT (Bidirectional Encoder Representations from Transformers). Transformers are a deep learning model, where every output is connected to every input and the weighting of

which is based on their connection. BERT, in comparison to other models, can read text bidirectionally, which means it can use surrounding text to establish context. This was particularly desirable given the short-text authorship characteristic problem described earlier – by being able to use context in both directions, the model has more corpora to evaluate on.

Another way to attempt to reduce the impact of the short-text authorship characteristic problem is by evaluating all messages together. To check whether this does minimise this issue (if it doesn't it's better not to do it, as it is computationally intensive to re-evaluate users) 12 participants from work were asked to send ten messages each to the system. The age and gender of each participant was known in advance. As a control, they were all asked to send the following collection of messages: three messages detailing information about themselves, four messages asking agricultural queries, three general chat messages (such as "What do you do?"), emulating the ratio a normal user of the system may have. They were not told on what topic or area to base messages around so the biggest distinction on age and gender could be had.

The messages of each participant were then passed through the evaluator as follows:

1) Only using the first message to calculate their age and gender.
2) Evaluating each message separately and then taking the most outputted answer - if three messages result in an output of female, and seven result in an output of male, take male to be the answer.
3) Combining all the messages together and then passing that to the evaluator - if they send the messages "a", "b" and "c", the text "a b c" is passed to the evaluators.

The impact of each solution programmatically was also evaluated – the first approach would delay the response of the first message back by on average 2 seconds (how long it takes to go through the evaluators) and then all other messages would be received at a normal rate. The third option would be run as an overnight task, with combining messages taking O(n) time proportional to the number of the users' messages, and then the evaluation taking two seconds. More containers can be spun up (e.g. on AWS) so users can be evaluated concurrently if need be. The second option would result in every message having an extra 2 second delay, and an additional column in the database – the number of messages that gave the output that is currently their calculated characteristic.

Say that user A has 100 messages. They have been determined to be a male, with 57 messages supporting that claim. A new message comes in and is evaluated to be sent by a female. The user now has 101 messages, but 57 is still more than 50% so the user remains evaluated as a male. By storing the number of messages supporting the claim as an extra column in the database, the system saves having to re-evaluate all the messages and perform all the calculations again.

The graph below shows the results of the three different approaches. Approach 1 had the least total average accuracy at 37.5%, approach 2 jumped up, with a total average accuracy of 70.83%, however approach 3 had the highest total average accuracy at 79.17%, showing that by combining the messages to be evaluated, a comparable accuracy is obtained to the models evaluated accuracy, and thereby reducing the impact of the short text authorship characteristic problem.

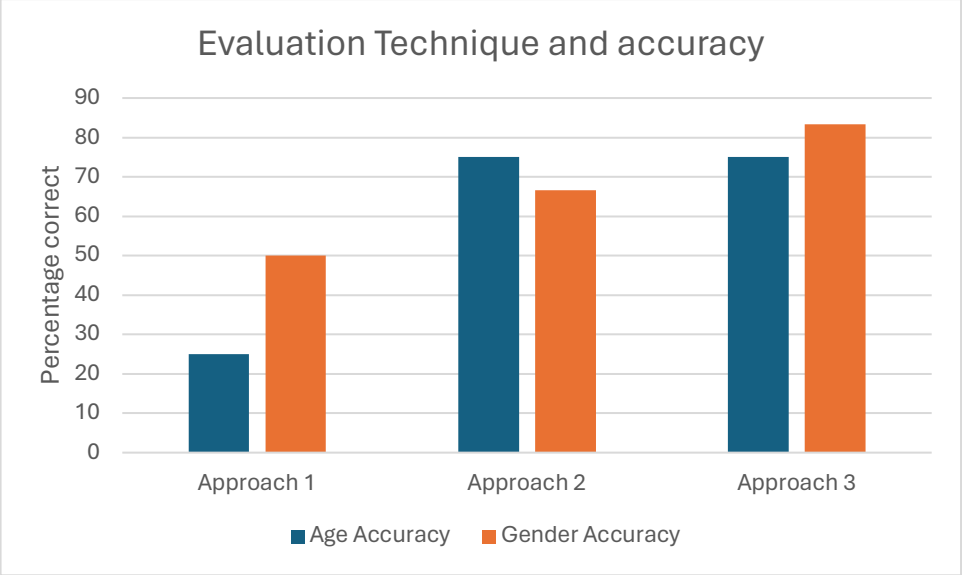


Figure 54 A bar chart comparing the 3 different approaches suggested earlier. Approach 3 has the highest accuracy.

Of course, this is in part due to the discreetness of the data – gender only has two possible options, and the model for age only had three – young, adult, and aged. It is possible that a model that predicts age on a more continuous scale may perform much worse due to the small corpora available. However, ten messages are still a very low corpora – on average, each participant sent just below 200 words. It may be that the more messages a user sends, the more accurate the prediction gets still.

```kotlin
@Service
@Configuration
@EnableScheduling
class GenderEvaluator(
    private val messageRepository: MessageRepository,
    private val userRepository: UserRepository,
    private val attributeEstimator: AttributeEstimator
) : Logging {

    /**
     * Estimates the gender of the user given a list of all their messages, using a Gender Model from HuggingFace.
     * This estimated gender is then saved to the DB
     * */
    Jacqueline Dobreva-Skevington *
    @Scheduled(cron = "0 0 1 * * ?")
    @Transactional
    fun getGenderEstimate() {
        userRepository.findAll().forEach { user ->
            val messages = messageRepository.getMessageByUserIdAndType(user.id).also { it: List<Message>?
                if (it.isNullOrEmpty()) return@forEach
            }
            val gender = messages?.let { attributeEstimator.estimateGender(it.map { sms ->  sms.message }) }

            logger().info("User ${user.id} estimated gender is $gender")

            user.gender = gender
            userRepository.save(user)
        }
    }
}
```

Figure 55 Implementation of Approach 3 - evaluating characteristics based on amalgamated messages.

Given the findings, approach three was implemented. As discussed, the evaluator is set as a scheduled task which runs overnight. It is annotated with *@Transactional* for safety so that the database rolls back to its previous state if an error occurs. In the code sample above, each user is iterated through, and for every user, all messages are amalgamated. Given the users amalgamated messages, the model is run (in this case gender), and the gender gets saved to the users' entity.

11.6.2 Cross-Lingual Evaluation

A key aspect of the system is its global nature. As such, it is expected that in production, messages from many different languages would be sent in. These messages all get translated into English when received due to OpenAI, the LLM used in this project for its high accuracy in agricultural RAGs, performing better in English, as discussed earlier. Therefore, it is possible that the evaluators work less effectively on users in a different language. To test their efficacy, five family members from Bulgaria were asked to send the same collection of ten messages as the 12 participants in the previous study and run through approach three. This was to keep the conditions as similar as possible. All participants conversed with the system in Bulgarian, and once

again, the age and gender of each participant was known. Below is the graph comparing the accuracy of the third approach between the Bulgarian-speaking participants whose messages got translated and the 12 participants from the previous study who conversed in English.

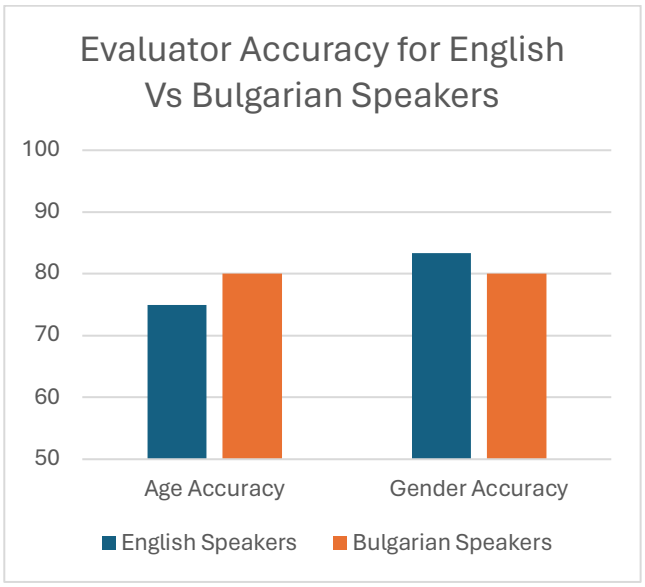


Figure 56 Comparing the model accuracy between translated and non-translated messages.

As we can see, both translated and non-translated messages had similar accuracies, with the accuracy of non-translated messages coming at just over 79% and the accuracy of translated messages at 80%. However, the Bulgarian speaker study did not have a particularly large sample size due to limitations in family size, and thereby while the initial results are promising, it may be that with a larger study, the results are different.

11.7 Literacy

The final characteristic tailored on is literacy. As discussed before, users will only implement suggestions if they can understand them, and with only two thirds of children completing primary school [62] in low-income countries, tailoring on literacy is good way to achieve this.

No models could be found for estimating a user's literacy level, and with a lack of a dataset, a more programmatic approach was taken. This approach makes several assumptions:

1) The less literate a user is, the less verbose they will be through texting, and therefore the less they will write.

2) The less literate a user is, the more errors they will make per message.
3) The less literate a user is, the less complexity will be present in a message.

To be able to tailor, it is desirable to have a continuous literacy score of 0 to 100 where 100 was most literate. As such, all functions returned a number between 0 and 100.

The first function uses assumption 1. The average message contains around 17.5 words (11-25) [63], and if users go slightly above or below that, we don't want it to have too much of an effect. Therefore, a sigmoid function can be used once again, with 17.5 being the bound (where the result is average – i.e. 0.5). This was then multiplied by 100 to scale it.

```
/**
 * Taking the average number of words in the users all time messages, the number is scaled between 0.0 and 100.0
 * where 100.0 is most verbose. The scaling uses 17.5 to provide an average score of 50.0 as the average number of
 * words in a message is 17.5
 *
 * @param averageWordCount - the average number of words in all the users messages
 * @return a double between 0 and 100 where 0 is least verbose and 100 is most
 * */
Jacqueline Dobreva-Skevington
private fun calculateWordCountScore(averageWordCount: Double): Double {
    return scaleProbability(averageWordCount,  bound: 17.5) * 100.0
}
```

Figure 57 Function to scale the average word count per message into a literacy level score.

Using the second assumption, JLanguageTool was utilised, which had a free API to check grammar and spelling mistakes. The function returns the number of violations matches it has to JLanguageTool's list of rules, which is scaled to 100 based on the messages total number of words.

Using assumption 3, two more functions were created. The first deals with the Token-Type Ratio of the message, by getting the number of unique words in the message over the total number of words, and scaling that up two 100. This approach assumes that the more unique words there are in the message, the more complex it is. The final function uses Flesch Reading Ease Score (FRES) test, which is calculated using the following formula:

$$FRES\ Score = 206.835 - 1.015\left(\frac{total\ words}{total\ sentences}\right) - 84.6\left(\frac{total\ syllables}{total\ words}\right)$$

Where a score of 100 indicates an extremely easy text to read, one that can be understood by very young children, and a score of 0 indicates an extremely specialist

text that can only be read by a few people in the relevant field. The FRES Test is ubiquitous, used in common word processing software such as Microsoft Office Word, and by the US Department of Defence as a standard test for the readability of its documents and forms [64] as a few examples. To fit with the literacy scale used by the rest of the functions, this is inversed so that 0 is an extremely easy text rather than 100.

These four functions are then amalgamated and averaged to produce an overall literacy score.

```
userRepository.findAll().forEach { user ->
    val messages = messageRepository.getMessageByUserIdAndType(user.id).also { it: List<Message>?
        if (it != null) {
            if (it.isEmpty()) return@forEach
        }
    }

    // Calculate average word count per message
    val averageWordCount = messages?.map { it.message.length }?.let { calculateWordCountScore(it.average()) }

    // Calculate average number of errors per message - weighted
    val averageErrorsPerMessage = (messages?.map { errorsInMessage(it.message) }?.average() ?: 0.0) * 6

    // Calculate average message readability - weighted
    val averageReadability = (messages?.map { messageReadability(it.message) }?.average() ?: 0.0) * 2

    // Calculate average vocabulary complexity (Type-Token Ratio)
    val averageVocabularyComplexity = messages?.map { calculateTypeTokenRatio(it.message) }?.average()

    // Combine individual metrics to calculate overall user literacy level
    val literacy = ((averageWordCount ?: 0.0) +
            (averageErrorsPerMessage) +
            (averageReadability) +
            (averageVocabularyComplexity ?: 0.0) / 10.0).toFloat() / 4 //scale back to 0-100

    user.literacy = literacy
    userRepository.save(user)
```

Figure 58 Calculating overall literacy level of users.

The function is weighted according to an informal test amongst family members involving participants with English as a second language and ones which are native, using their own self assessed literacy level on a scale of 0 to 100.

Literacy evaluation is also a scheduled task that runs overnight, as with the other evaluators to maintain consistency, and considers the entire message history to try negating the short-text authorship characteristic problem.

11.8 User Suggestibility – In Practice

Now the relevant characteristics of each user are evaluated, they can be used to tailor based on the user. As discussed at the start of this section, user mirroring is a way of tailoring that involves copying certain aspects of the user, to make them more suggestable to performing a certain action. A low-level example of this is Android's new Material 3 wallpaper styling, where developers can match the theme of their app to the user's wallpaper, and thereby make the app look more familiar to the user, and potentially lead to them engaging with the app for longer. In terms of the system where the UI is the messages the user receives, to implement user mirroring, the messages that users receive should match as close as possible to the users own characteristics – in other words, the messages received by the user should look like they were written by someone with the same characteristics as the user.

There is a strong body of research echoing the need for such an approach – with the need to make users implement suggestions, the knowledge of what makes them not, and studies showing the need for individualised approaches – for instance gender specific approaches are needed to reduce the gender gap within farming [65].

To accomplish user mirroring, the large language model OpenAI has been leveraged. LLM have been shown to be good at impersonation, with a rise in cyber-crime based on impersonating people [66] . Companies have also been created to intentionally impersonate deceased loved ones, such as 'HereAfter AI' [67] and 'StoryFile' [68]. As such, once responses based on a user's message are compiled, they are fed to an OpenAI agent with the following prompt for age, gender and literacy:

"You are a bot that re-writes suggestions (so users are more susceptible to them) by mirroring their characteristics. Given a set of characteristics (age, gender, literacy level), you want to rewrite the user input to that the input. User literacy is rated from 0 to 100, where 0 is extremely low (user cannot understand English) and 100 is extremely high (user can understand anything).
User Literacy: {}, User Gender: {}, User Age: {}
DO NOT add any new text. Your job is to only rewrite messages. DO NOT OFFER ASSISTANCE. Keep ALL punctuation as is."

A similar prompt is used for knowledge areas, where the LLM is asked to summarise according to a user's knowledge level of the area, with 0 being no summarisation, and 1.0 being only core details are kept.

11.9 Evaluation

To evaluate the efficacy of tailoring, the 12 participants from the non-translated Age and Gender evaluation study were asked to send 5 more messages and then complete a SUS questionnaire. The same 12 participants were selected as their conversation history was already in the system, and their characteristics had all been evaluated using the final approach selected for use in the system. They were instructed to send only agricultural questions this time as the efficacy of user suggestibility needed to be monitored. A between subjects' design was chosen, and the participants were split into two groups, both having roughly the same weighting of age and gender. Between subjects was selected in contrast to within subject as there was no quantitative measuring to be done (the participants weren't smallholders that could be observed as to which suggestions they did and did not implement).

The first group's messages went through a system without user tailoring at the end, whereas the second group had their messages tailored. After they sent the 5 questions and received the answers, each group was given the following survey to complete:

1) I found the system easy to use.
2) I think the system was unnecessarily complex.
3) I found the system friendly.
4) I found the system unnecessarily cumbersome to use.
5) I would implement the suggestions given by the system.
6) I did not understand the suggestions given by the system.
7) I felt the system used normal language when responding to me.
8) I could not understand what the system was saying.
9) I felt the system explained things well.
10) I felt belittled when talking to the system.

Rating each statement from 1 to 5 where 1 was strongly disagree, and 5 was strongly agree.

Neither group knew which version of the system they were getting, so therefore any bias based on the fact the participants knew the person behind the system was mitigated as much as possible.

The scores were normalised to 0-4 for the SUS calculation, and then collated in the following table.

	Tailored Messages	Standard messages
Q1	3	3
Q2	2	2
Q3	4	2
Q4	1	1
Q5	4	2
Q6	1	1
Q7	4	3
Q8	1	1
Q9	4	2
Q10	1	1
SUS Score	82.5	65

Table 6 The results to the SUS questionnaire comparing normal and tailored messaging.

Showing that participants were 1.27 times more suggestable to messages after they'd been tailored – if 1000 messages are sent out, and before only 500 people would implement them, with tailoring that number would be at 635.

It's also interesting to note in the radar charts below that while both groups thought the system used human-like language, the non-tailored group found the system less friendly by two points and was also less likely to implement the tailoring by two points as well and felt that the messages were less well explained by two points. Please note that one radar chart has a scale up to 4, and the other a scale up to 3.

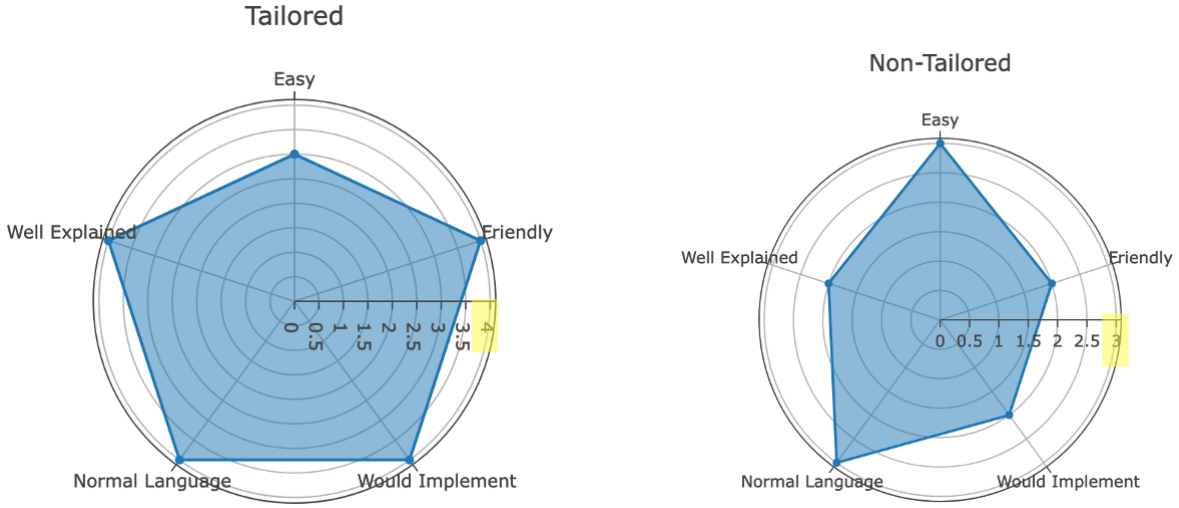


Figure 59 Radar chart of SUS questionnaire for the impact of tailoring.

11.9.1 Cross-lingual Evaluation

As with before, the five Bulgarian speaking participants were asked to perform the same tasks as the English-speaking ones. The questionnaire was then translated and sent out for completion. Surprisingly, an even bigger disparity was present here. The standard messages received an overall SUS score of 47.5, with the tailored messages remaining the same at 82.5, showing that tailoring increased suggestibility of messages by 1.76 times. Contributing to the low score on non-tailored messages was an average rating of 1 on friendliness, as well as an average of 1.5 on feeling messages were well explained.

As with previous evaluation on cross-lingual capabilities, it must be once again noted that due to restrictions in familial size, only a small sample pool was present for translated evaluation, and therefore the findings can only be as accurate as the number of participants involved. However, once again, the initial findings support the effectiveness of the novel user suggestibility concept presented in this dissertation to combat users not implementing suggestions provided to them by AES systems.

11.9.2 Summary

Overall, the project presents a novel user suggestibility concept, tailoring on age, gender, literacy and knowledge level. Through some qualitative studies, we have shown the proposed concept does have an impact on the suggestibility of messages, despite the short-text authorship characteristic problem.

The work done in this section presents a promising solution to qualitative tailoring for increased user suggestibility, leading to increased engagement and likelihood suggestions are implemented by smallholders, and thereby increasing the effectiveness of the solution.

12 Message Handling

12.1 SMS Interfacing

The project uses GatewayAPI to send and receive messages, as one of the only SMS-providers that support receiving messages on UK short codes. Interfacing with the API is done using a RESTful Controller which conforms to standard API constraint standards. The Spring-Web module provides an *@RestController* annotation, removing the need for boilerplate code and providing out of the box error handling. The controllers implemented respond to webhooks denoting received messages and status changes.

```
/**
 * Webhook for receiving incoming messages from users
 *
 * @param resource - the received message
 * @return the saved message dto
 * */
 Jacqueline Dobreva-Skevington *
@PostMapping("/receive")
fun receiveSMS(@RequestBody resource: RecievedMessageDTO): MessageDTO {
    logger().info("Received Message with id ${resource.id} and message ${resource.message}.")

    //Translate the message into English if necessary and save to the DB to the associated user.
    //If no associated user, create a new one and determine preferred language
    val sms = recieveSmsService.save(resource)

    logger().info("Sending ${sms.id} to message handler.")
    recieveSmsService.sendToMessageHandler(sms)

    return sms.toDTO()
}
```

Figure 60 The Rest Controller receive SMS webhook.

When a message is received and gets caught by the receive SMS, it goes through the following pipeline:

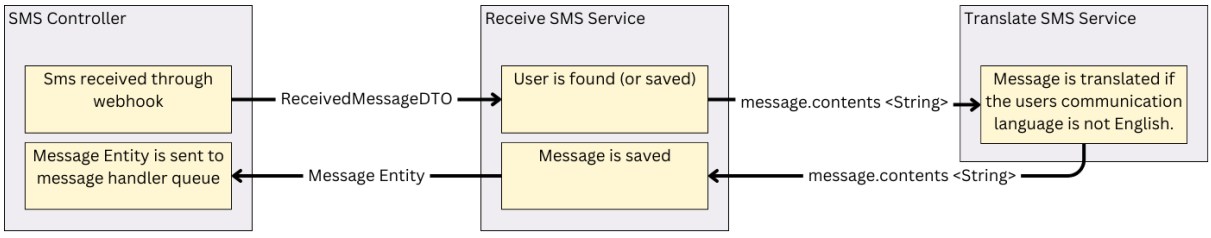


Figure 61 The flow of a received message.

If the user sending the message is not a first-time sender, their information is retrieved

from the database, along with their '*preferedLanguage*'. If the preferred language isn't English, the message is translated into English, and then saved into the database for future use before being sent to the message handler queue. If the user is new, languages of incoming messages are determined using LibreTranslate, a free language API with support for over 45 languages. The user is then saved to the database, along with their preferred language, and the message is once again translated if needed and sent to the message handler queue.

```
@Transactional
fun save(resource: RecievedMessageDTO): Message {

    //find user to associate message with or create a new user
    val user = userRepository.findByPhoneNumberContaining(resource.phoneNumber) ?: let { it: RecieveSmsService
        userRepository.save(
            User(
                id = UUID.randomUUID(),
                phoneNumber = listOf(resource.phoneNumber),
                //if it's a new user, figure out their language from their country code
                preferredLanguage = LanguageCode.fromLanguage(getLanguageCodeForCountry(resource.country).toStandardLanguage())
                    ?: LanguageCode.EN,
                stopCollectingInformation = false
            )
        )
    }

    val lang = user.preferredLanguage ?: LanguageCode.EN

    //if the incoming message isn't in english, translate it into english
    if (lang != LanguageCode.EN) {
        resource.message = translateSmsService.translateMessage(resource.message, fromLanguage = lang)
    }

    //return the new message entity and save it to the DB
    return Message(
        resource.id,
        resource.phoneNumber,
        resource.message,
        LocalDateTime.now(),
        user,
        MessageType.INCOMING
    ).also { messageRepository.save(it) }
}
```

Figure 62 The Receive SMS Service displaying what occurs when a message is received by the system.

12.2 Message Handler

One of Precision Development's (PxD) shortcomings was its rigid flow-based structure. A natural language application needs to be able to respond to a multitude of different message types, and all their various combinations and permutations. There are two base message types that the system needs to respond to: agricultural queries, and general conversation (a necessity for making the system seem more personable and encourage further interaction, as discussed earlier in the eHealth case study in the

section 'Qualitative User Adaptation'). With these two, it is simple enough to take a rule-based approach checking for an agricultural related question, extracting that to be sent to the expert system, and doing a '*message.replace(agriculturalQuery)*' to obtain the general messaging.

```
import spacy

nlp = spacy.load("en_core_web_sm")

1 usage  new *
def isQuestion(sentence):
    """
    Determine if a sentence is a direct question or implies a question based on syntactic structure.
    """
    # Check for sentences starting with wh-words
    if sentence[0].text.lower() in ["how", "what", "when", "where", "which", "who", "whom", "whose", "why"]:
        return True

    for token in sentence:
        # Check for inversion: verb + subject
        if token.dep_ == "ROOT" and token.tag_ == "VB":
            for child in token.children:
                if child.dep_ == "nsubj":
                    return True
        # Check for implied questions
        if token.dep_ == "xcomp" and token.head.text.lower() in ["want", "wonder", "ask", "know"]:
            return True
    return False


text = "This is an example text. What is your name? Where are you from? I want to know when to plant seeds."
doc = nlp(text)

questions = []

for sent in doc.sents:
    if isQuestion(sent):
        questions.append(sent.text)

print(questions)
```

Figure 63 An example of using the Spacy Library in Python to extract questions, including indirect questions. This could easily be extended to match only agricultural questions by adding in matching to common agricultural terminology.

However, the system also has several other requirements:

1) The need to save any feedback provided by users for continuous system improvement.
2) The need to accept referrals (a 'refer a friend' system), so the program has larger outreach.
3) The need to scrape information the user provides about themselves and their

smallholding (used in 'Quantitative User Adaptation') – sentences containing such information should not just be relegated to a general chatbot.

4) The need to accept stop requests (such as "Please stop sending me notifications" or "Please stop asking me about more information") to comply with regulations on unsolicited message sending.

Therefore, the following messages need to be detected: AGRICULTURAL_QUESTION, INFORMATION, REFER_FRIEND, FEEDBACK, STOP and GENERAL, of which STOP contains INFORMATION and NOTIFICATION.

Given the scale and complexity of determining these message types, traditional rule-based approaches do not work. The system also needs to be extensible – allowing for more types to be easily added and removed. For instance, from the review provided after the Presentation, FEEDBACK as a detectable message type needed to also be implemented. This extensible pipeline works as follows:

First, a generic interface needed to be created that all classes that handle different message types implement.

```
interface MessageHandler {
    val messagePartType: HandlableMessageType
    new *
    fun extractPartAndReturn(remainingMessage: String, userID: UUID): List<String>?
    Jacqueline Dobreva-Skevington
    fun generateAnswer(prompts : List<String>, userID: UUID): List<String>?
}
```

Figure 64 The *MessageHandler* Interface. Each implementation has an associated message type, a function to deal with the message in its current format, and a function to generate the answer.

Then we need a message handler service which iterates through each available handler. For each handler, the *extractPartAndReturn* function (see Figure 64) is called. If it returns null, no such message type exists, else it returns the sentences that match that message type. These sentences are then passed to the *generateAnswer* function which generates an answer accordingly, and the sentences are removed from the original prompt. The produced answers are added to a variable to send to the message compiler.

```
val request = messageRepository.findByIdOrNull(message.messageId)!!
var prompt = request.message
val toReturn = mutableMapOf<HandlableMessageType, List<String>>()

messageHandler.forEach { handler ->
    handler.extractPartAndReturn(prompt, request.user.id).ifNotNullOrEmpty {
        prompt = prompt.replaceList(it)
        handler.generateAnswer(it, request.user.id)?.let { answers ->
            toReturn.put(handler.messagePartType, answers)
        }
    }
}
```

Figure 65 Part of the *MessageHandlerService handleRequest* function. This function is called every time the queue processes an item.

12.2.1 Handlers

The last part of implementing the extensible generic pipeline is to implement the message handlers themselves. Each Handler is annotated with a Spring *@Order(n)* annotation, where 'n' is the order, ascending, that each class should appear. The classes are injected into the *MessageHandlerService* class (Figure 65) by placing '*private val messageHandler: List<MessageHandler>*' in the constructor. The *messageHandler* variable is then a list of *MessageHandler*'s, In the order specified by the Spring @Order annotation.

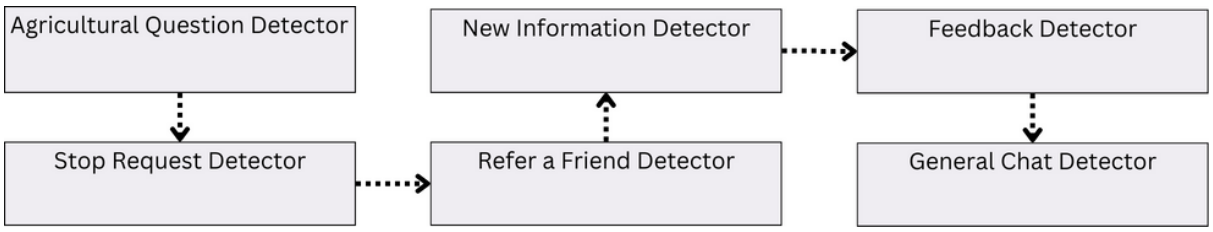


Figure 66 The available message handlers. The arrows represent the order they go through.

Figure 66 shows the available message handlers, with the dotted arrows showing the order they are placed in. Creating an order between handlers is necessary as, for instance, imagine if the user sends in the message "Will it rain today? I'd like to refer my friend John Doe." If the refer a friend detector was randomly placed after the new information detector, then the new information detector may see the name "John Doe" and assume that it is the smallholder's name, extract it and save it to the database, and then pass the message on to the next handler with the sentence "I'd like to refer my friend John Doe." removed from it. The refer a friend detector would then pick up the remaining message, "Will it rain today?" and John Doe would never get referred.

To discuss each handler more in depth, the following message will be sent through the

system, with each step explained: “Hello. This is a test message. What is your favourite colour? Please stop sending me notifications. I'd like to refer a guy called Jamali to this service. His number is 07500338340. This system gives good feedback on corn growing but could be better at pest management explanations. My smallholding is 10 meters. At what depth should I plant aubergines? When do I harvest tomatoes?

When the message first gets picked up by the message handler, the first handler it goes through is the Agricultural Question Detector.

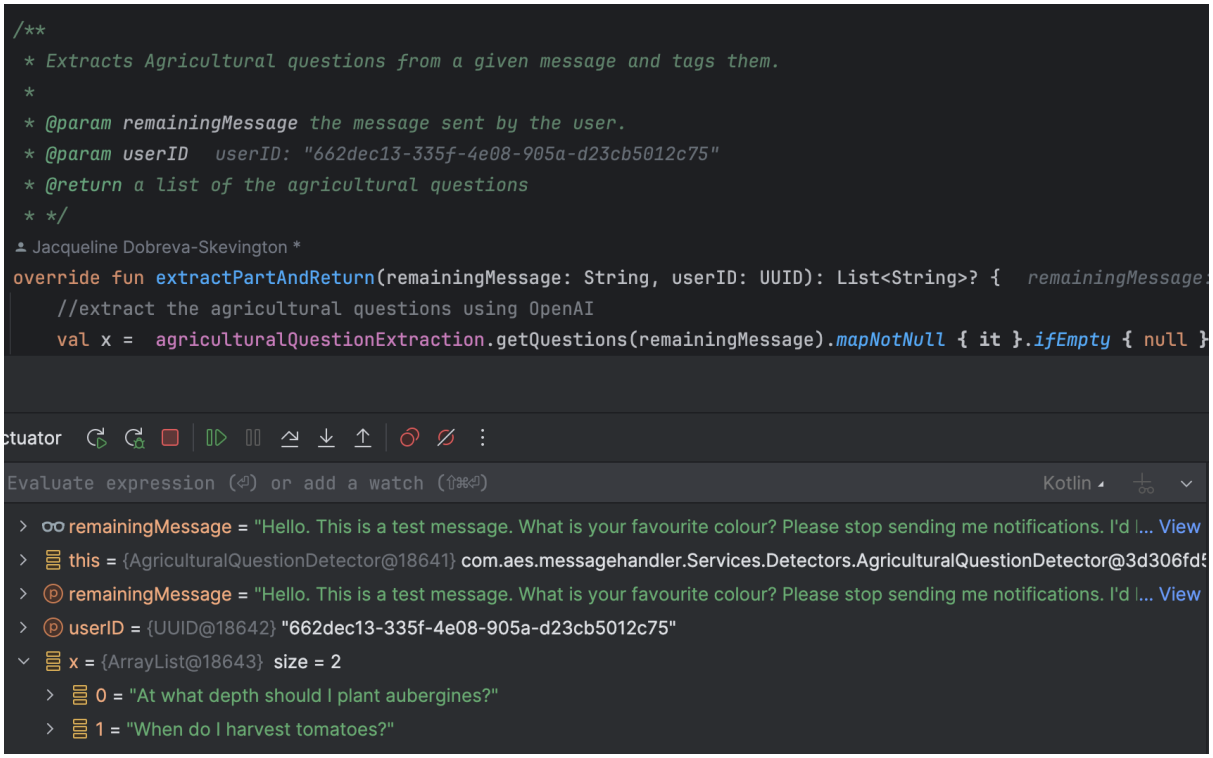


Figure 67 The *extractPartAndReturn* function of the Agricultural Question Detector with the breakpoint output in debug mode.

Here we can see in the temporary variable x the two agricultural questions have been correctly extracted from the message. This is done using the LLM approach discussed earlier. The Python code driving this is called using the Kotlin-Python interoperability functionality set up throughout the project. The two questions extracted are removed from the total prompt ready to be passed to the next handler, but before it does get passed on, the extracted messages are separately run through the Expert System, the working of which was discussed earlier in the Expert System section. The results of this get added to the *toReturn* variable in the message handler service (Figure 65).

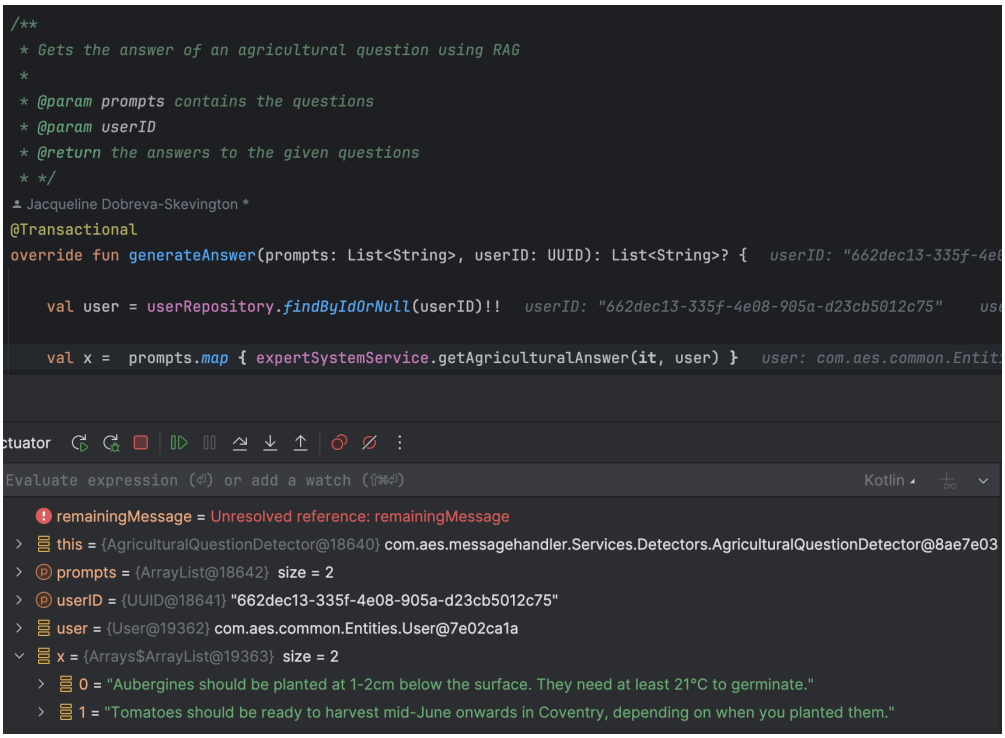


Figure 68 The response provided by the expert system along with the *generateAnswer* function in the agricultural question detector.

The first response uses the SID database to extract information about seeds, and the second uses the ECOCROP dataset, along with the Trefle API and weather knowledge specific to the smallholding location, displaying the use of Agriculture 4.0 principles.

The next handler is the Stop request detector. The system automatically sends out notifications to the user depending on event triggers or timed reminders. However, in some countries, providers can be banned if they are deemed to be sending unsolicited texts that a user has not asked for, such as notifications. Furthermore, the system will keep asking information about the user to gain a more holistic picture of the user and be able to tailor to them better. Understandably, however, some users may find this irritating if they do not wish to divulge any information. As such, it is important to allow the user to request either type of message to stop, to keep the system compliant and enhance user experience respectively. The stop request is once again identified using an LLM and interfacing with it through Kotlin-Python interoperability. When it is identified, it is sorted either into the correct stop request category, and this information is saved to the database in the user table under the stop_collecting_information and stop_sending_notifications columns.

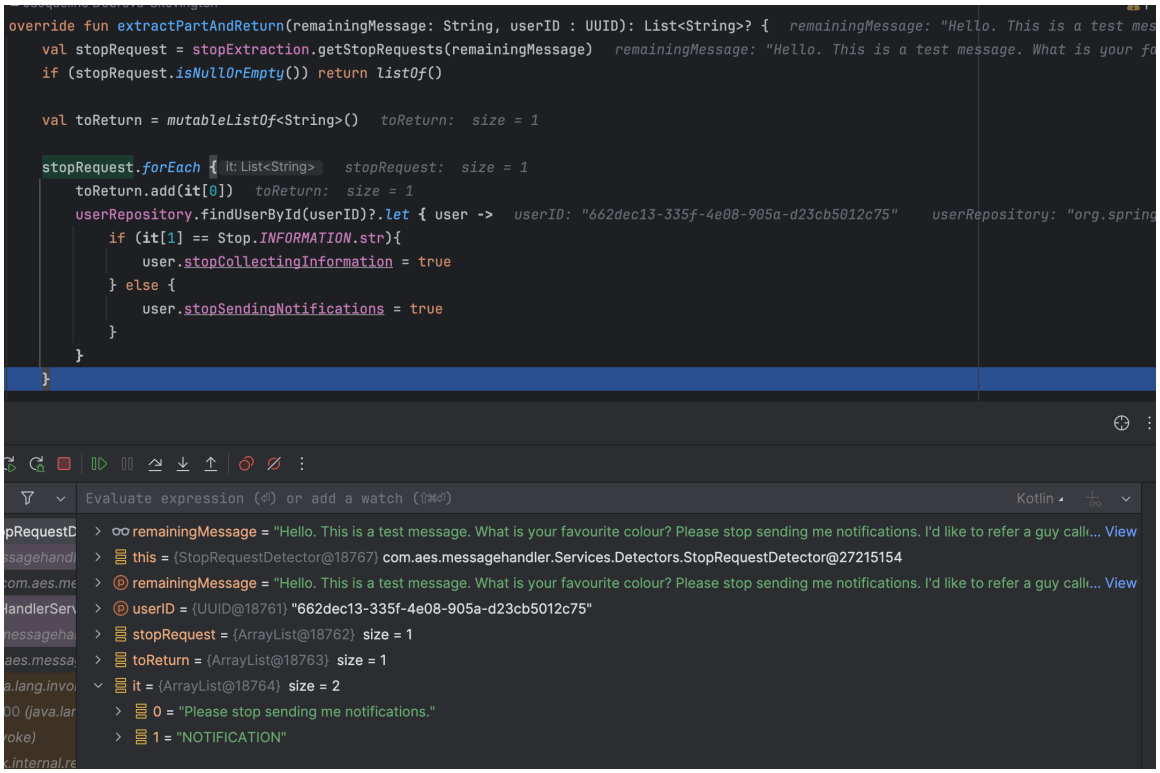


Figure 69 The stop request detector detecting the relevant part of the sent message and categorising it into the correct stop request.

Here we can see the stop request detector in action, with the debug console variable 'it' expanded to show how the stop request has been correctly identified. Once again, the associated sentence (in this case, "Please stop sending me notifications.") is removed from the main prompt, and the answer for the detector is generated and added to the *toReturn* variable. In this case, the answer is a blanket answer of "I will stop".

The next handler along the chain is the refer a friend detector. As discussed, this must come before the new information handler as otherwise friend referrals may be confused as new information, and the friend referral would never get through. When someone new is referred, the system needs to be able to send them a message explaining the use of the system and that they have been referred, but to do this, the referral needs to contain a phone number. As such, Google's phone number utilities library is used to check if the user referral has a phone number or not. If it does, the user is passed to the information service scraper, the same one that the new information handler utilises, which uses the custom NER made to obtain details about the user for quantitative tailoring. The new user is then saved to the database alongside any other given information, and a send message task is run every evening checking for users with no associated incoming messages (i.e. referred users) and sends them a welcoming message.

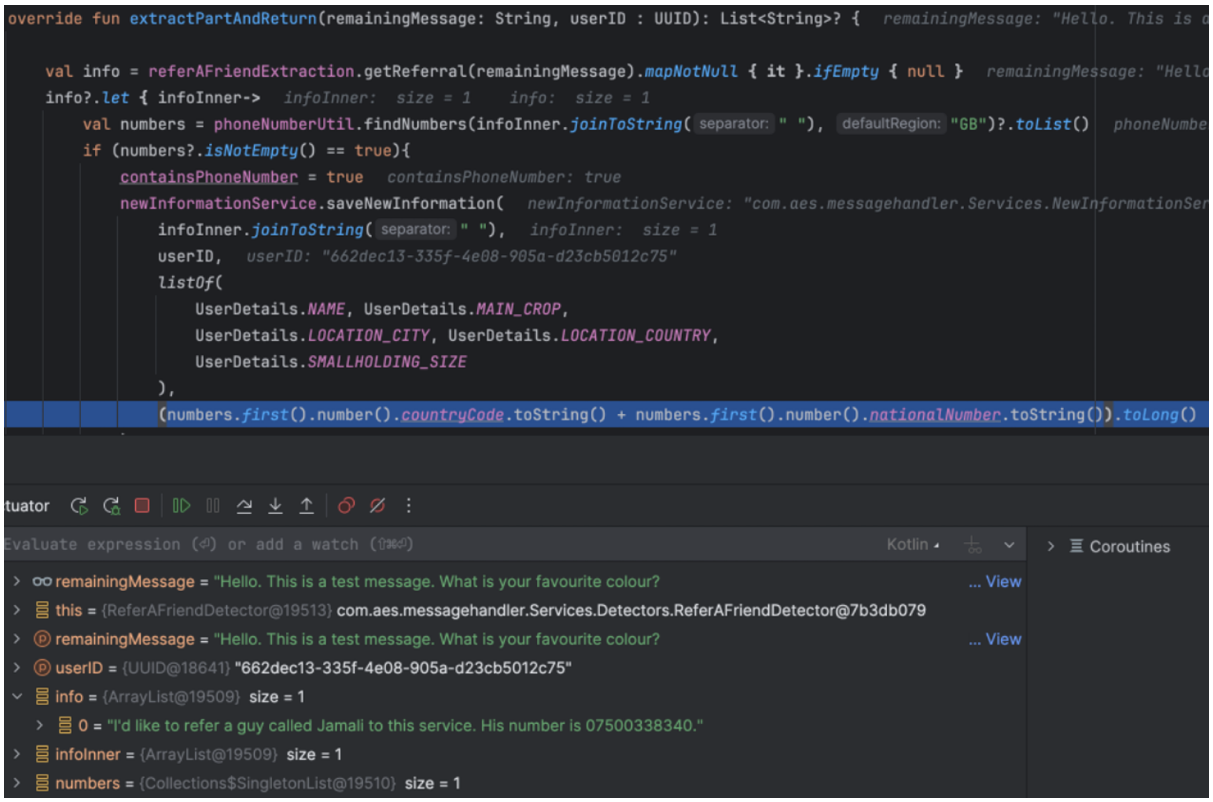


Figure 70 Refer a friend message handler.

We can see that the message part has correctly been identified – *info[0]*. As the referral contains a valid phone number, the user is saved and a standard "Thank you for your referral." is added to the *toReturn* variable in response.

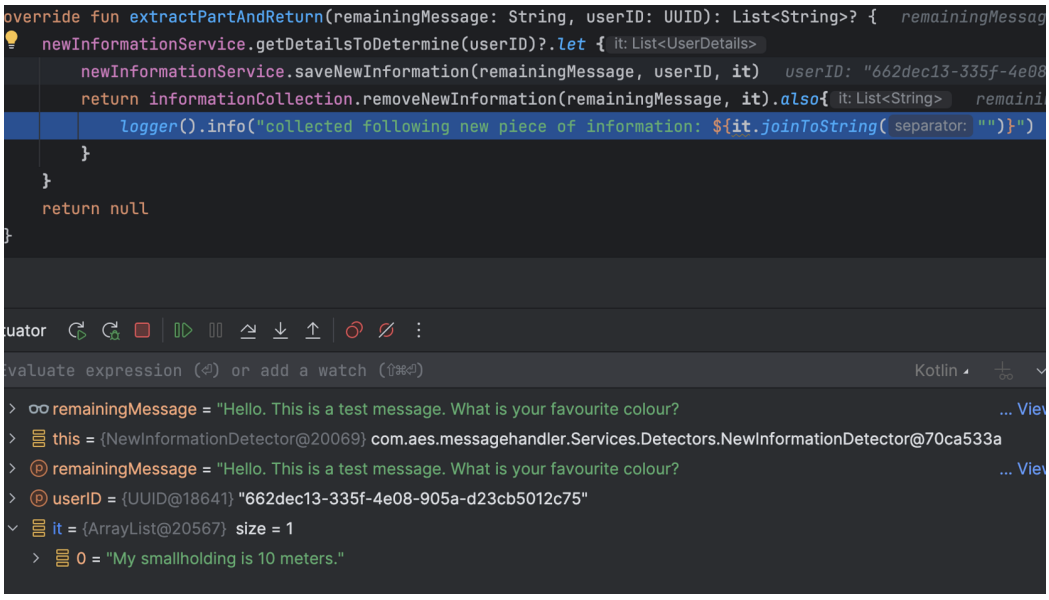


Figure 71 New information message handler.

The next handler deals with new information. We can see in Figure 71 that in *it[0]*, the new information the user provided in their message has been correctly identified.

This once again gets removed from the overall prompt, and the new information retrieved is sent to the *generateAnswer* function. The function checks whether the user has asked for information retrieval to stop, and if they haven't, it asks them about any remaining information the system doesn't yet have about them - in this case their name and main crop.

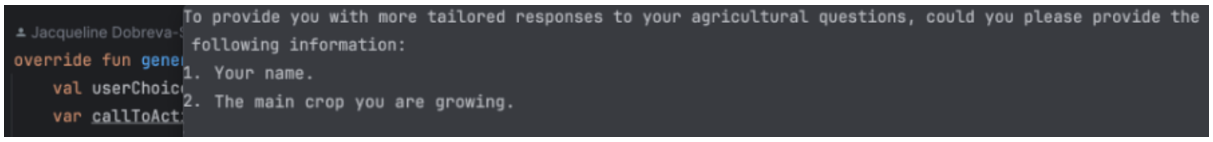


Figure 72 The output from the *generateAnswer* function of the new information message handler.

The penultimate message handler is the feedback detector, implemented after the Presentation in response to comments received. By collecting feedback, the system can continuously improve based on real user feedback once in production.

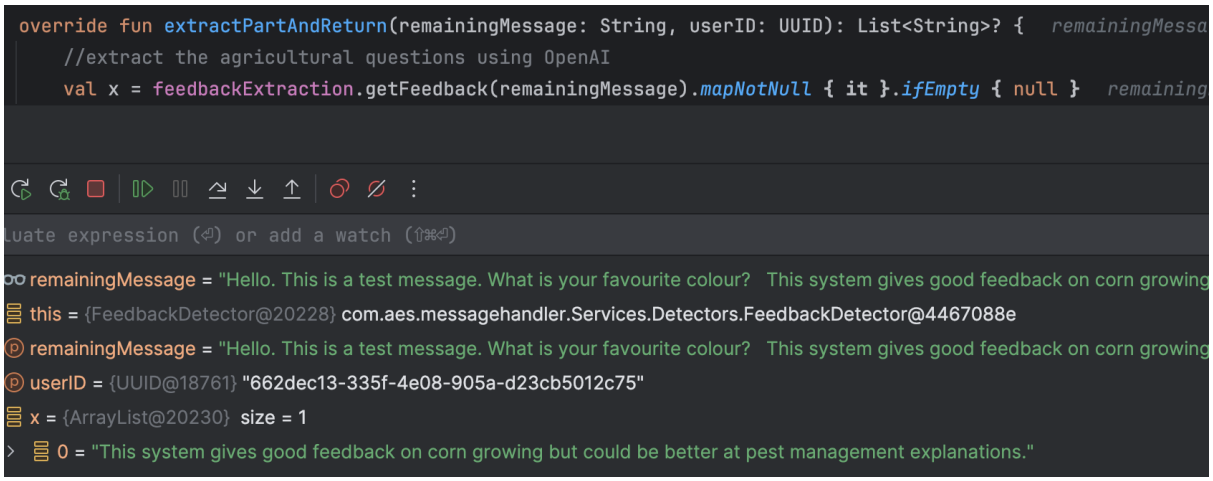


Figure 73 The *extractPartAndReturn* function of the feedback detector, with the output.

The feedback is extracted using LLM interfaced with Kotlin-Python interoperability. Once feedback is received, it is saved to the user feedback table, where the user has an @ManyToOne relationship to the entity, allowing them to give multiple pieces of feedback. The generate answer part of the handler acknowledges the feedback given by thanking them for it.

The final message handler is the general conversational handler, which takes all that's left from the prompt after all the previous sections are removed. This gets passed to a general chatbot, which has some information about what the system is, allowing the user freedom to converse whatever they wish to with the system, making it more personable and approachable, and thereby increasing the chance the user may engage with the system again.

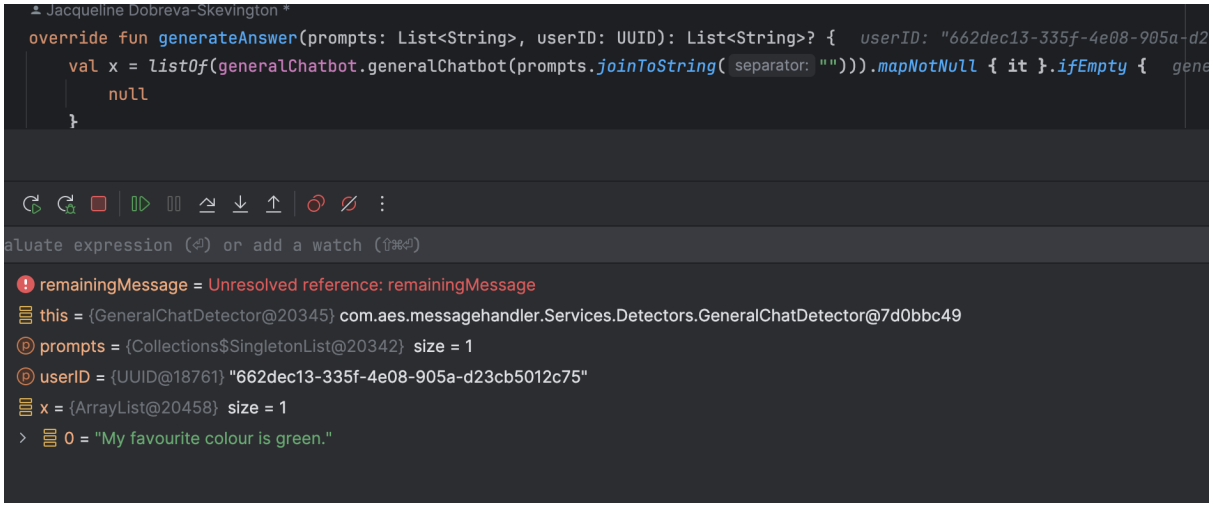


Figure 74 Response from the general chatbot at the end of the message handler pipeline.

Once all the handlers are iterated through, all their responses have been compiled in the *toReturn* variable, which is then passed to the message compile.

12.3 Message Compiler

The message compiler is the final stage of the response generation process, responsible for tailoring agricultural answers to the users' knowledge of the area the queries are on, as well as tailoring to the users' characteristics to increase the user suggestibility of the message. Once the message has been tailored, it goes through an amalgamator and fragmentizer, which brings appropriate parts of messages together, and attempts to make the length of each message approximately 17 words (to match the length of an average message), while keeping messages natural sounding (i.e. not breaking messages up in the middle of a sentence). Out of all LLMs, OpenAI was shown to be the most succinct [38] in an agricultural context, which well suits SMS messages.

Each agricultural message that is received is tagged into knowledge areas using Zero-shot classification, as discussed earlier in the Qualitative User Adaptation section. When tagged, the database is queried to see if the user has an associated knowledge level to that knowledge area. If they don't, a default of 0 (no knowledge) is returned. The two knowledge areas in the example being used are <AUGBERGINE, GROWING> and <TOMATO, HARVEST>. Neither knowledge area has an associated knowledge level for the user, so both are defaulted to 0, and subsequently the fully detailed response from the expert system isn't summarised during knowledge level tailoring, to ensure the user understands all information.

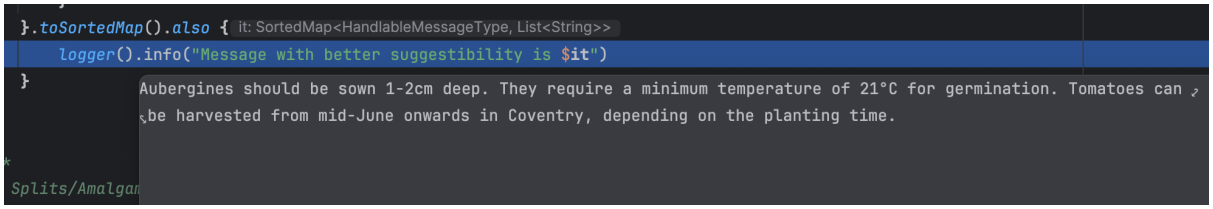


Figure 75 The agricultural question responses after tailoring.

As we can see in Figure 75, the agricultural question responses after tailoring remain the same as before to ensure the user gains maximum understanding.

Once the agricultural queries are tailored on knowledge level, the messages are sent through user characteristic tailoring, in Figure 76 below, we can see the message has been changed from "To provide you with more tailored responses to your agricultural questions, could you please provide the following information." to "To help you better with your farming inquires, kindly share the following details." The tailored messages are then all amalgamated and broken down into chunks of approximately 17 words per message, to match the average length of a message.

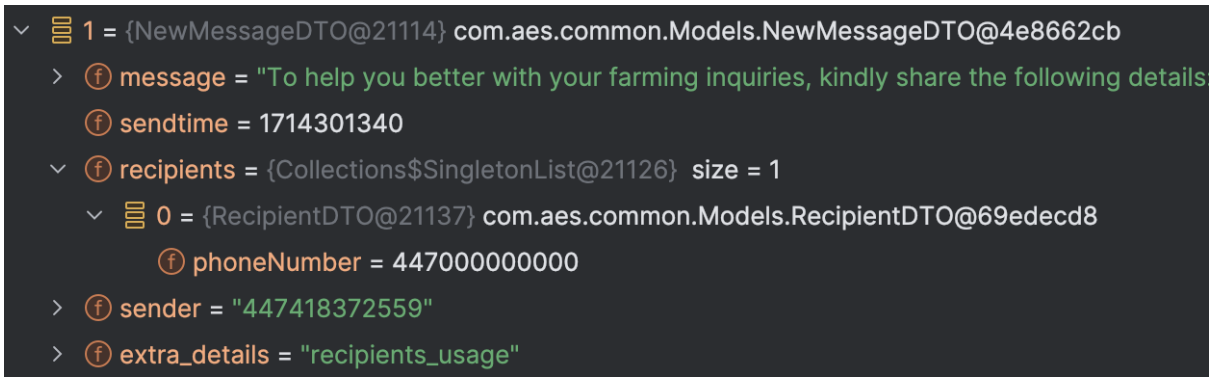


Figure 76 One of the new message DTOs ready to be sent to the user.

Each message chunk is converted into a *NewMessageDTO* (the required type for sending messages), as show in Figure 76, and are added on to the send_message_queue one by one, ready to be picked up by sms-services to send the messages to the user.

12.4 Queues

Throughout this section we have been discussing items being added to queues – for instance messages from sms-services being added to the message_handler_queue and messages from message-compiler being put on to the send_message_queue to be picked up by sms-services for sending. This is due to the microservice architecture of the project, and the need for a lightweight solution for communicating between them when

asynchronous operations are required.

When one service puts a small package of data onto a queue, a listener service then picks it up and ensures that the request can be processed successfully before removing the item from the queue. A FIFO queue is used so that messages are processed in the order which they are received. While in a production environment, this would use remote queue, such as services provided by AWS SQS, in the development environment, this was not available, so a simple strategy using file-based queues was devised.

First, a directory was earmarked for the queue, after which a calling service could write a file, numbered from 1 to 999, to the queue for processing. The queue works in a circular way to save memory, assuming that item '1' will have been processed by the time the counter reaches item '999'. Data was encoded to JSON to allow for quick serialisation and deserialization, and multiple helper method were used to ensure that the queue's integrity remained even after an error in the handling service.

This developed queue system will be published as a library and be freely available for other people to use after the submission of the project.

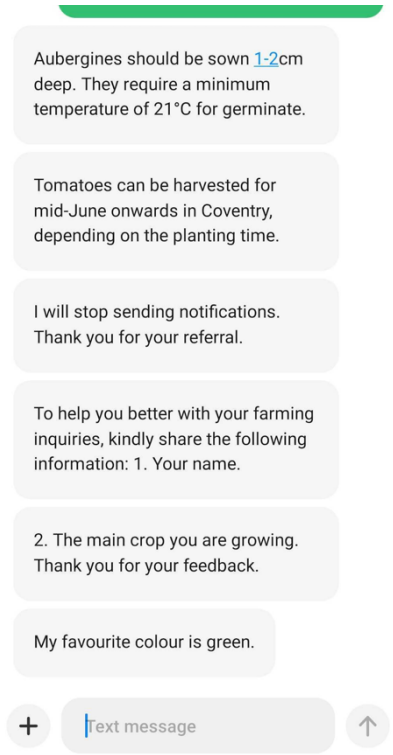


Figure 77 The messages received back from the system after sending.

12.5 Evaluation

Overall, the message handler provides an extensible and generic way to respond to user messages, without compromising on accuracy. The message compiler implements user suggestibility principles discussed earlier on in the dissertation and breaks up the messages into easily readable chunks for the user. Timing wise, responses are sent to the user within 5-30 seconds, comparable to that of an AEW responding, depending on the amount of different message types, and different agricultural questions the user sends. For the example discussed in the section, replies from the system finished being received at just over 22 seconds after the message was send.

13 System Architecture

Now that we have discussed the entire system in depth, we can put the entire system architecture into context to better understand how everything fits together. This section also provides the motivation behind the various languages and technologies used.

13.1 System Architecture and Design

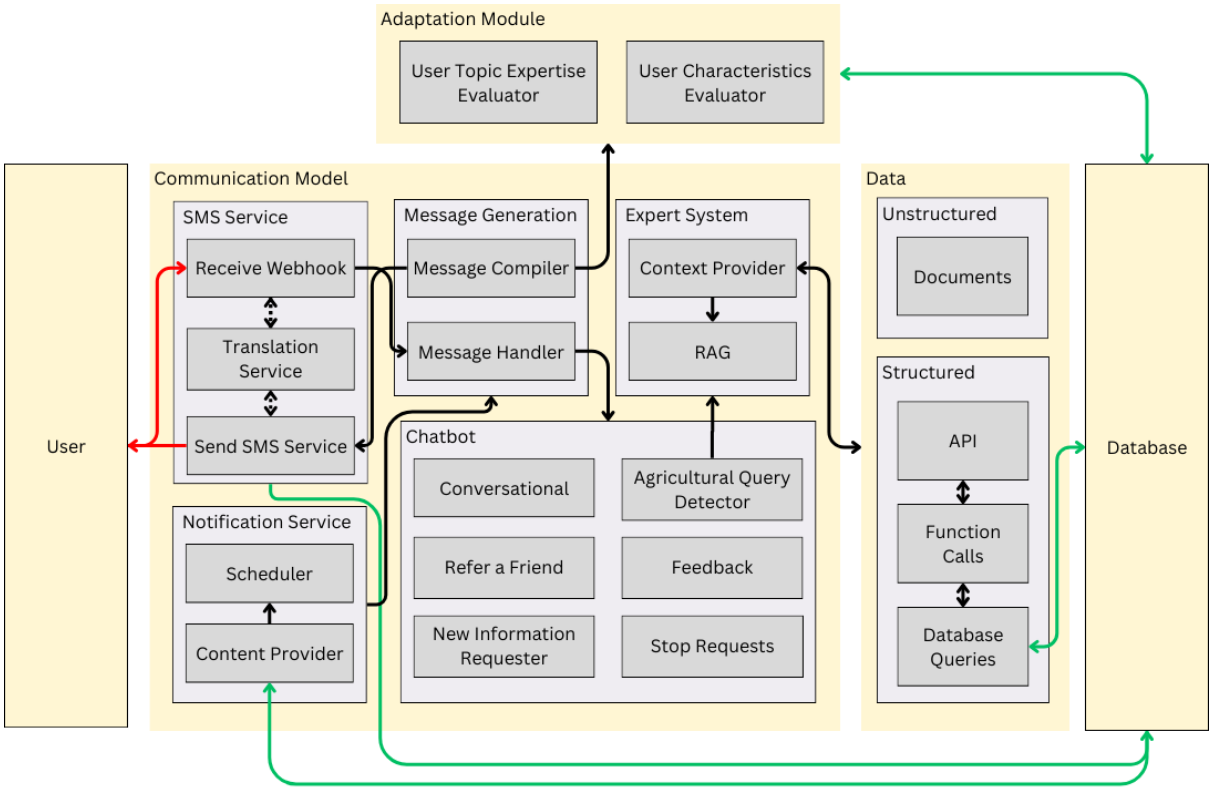


Figure 78 Overall System Architecture

The SMS Service is responsible for receiving messages and sending messages to the user. It also saves relevant information to the Database and retrieves anything it needs – such as whether to translate incoming and outgoing messages.

The Message Generator includes the Message Handler, which takes incoming messages from the SMS Service, classifies the message, and sends the relevant sections (AGRICULTURAL_QUESTION) to the expert system. The Message Compiler then takes the response from the Chatbot and the Expert System, and then blends them together using information about the user from the Adaptation Module to create an intuitive, informative response.

The Expert System interfaces with the knowledge base. It is responsible for providing an expert response to an agricultural query using the available data.

The Chatbot is responsible for generating responses to various parts of the message, as well as requesting any information the system doesn't already have about the user and their smallholding.

The Adaptation Module takes the user's message history from the database and calculates the user's expertise in various areas, as well as their personal characteristics such as age and gender. This allows the Message Compiler to tailor its level of detail and depth, as well as improve the user suggestibility of the message, increasing the chance that the smallholder implements the suggestions provided.

The Notification Service calculates whether a user should receive notifications about a certain topic using the user's message history obtained from the database based on their specific circumstances. The Scheduler then decides whether the notification should be sent to the user at regular intervals, and if so, at what duration, or whether a notification should be sent based on a certain trigger. This information is saved back into the database. If one of these events happen, then the Content Provider will provide a mock message to the message handler, which simulates that message being received by the user.

13.2 Technology Stack

Kotlin was chosen as the main language the project would be developed in due to its conciseness and readability, minimising the need for extensive boilerplate code, meaning that more time could be spend working on the wide range of objectives set out. The need to interface with llama-index (to provide the RAG framework) as well as use LLMs to provide the chatbot functionality meant that Python needed to be used as well.

A microservice-based architecture was chosen to enhance the system's flexibility, extensibility, and fault-tolerance compared to a monolithic structure. By developing the project in separate services with limited connections, each service could be developed and designed separately, complementing an Agile development approach. It also meant each service could easily be tested and debugged, increasing the speed of development.

13.2.1 Spring

As it supports microservice architecture out of the box, Spring was chosen as the server-side framework for this project. It is used with Spring Boot, which follows the convention over configuration principle, thus reducing the amount of boilerplate code needed. Spring Boot also means starter dependencies are automatically generated and handled, reducing configuration time.

The Spring Framework implements the Inversion of Control (IoC) principle, with Dependency Injection (DI), where objects define their dependencies and can receive necessary parameters without the programmer having to define them – saving time constructing parameters. Spring also incorporates automatic scheduling, an important feature when dealing with computationally intensive tasks which should be run overnight, such as user characteristic evaluation.

```
@Service
@Configuration
@EnableScheduling
class AgeEvaluator(
    private val messageRepository: MessageRepository,
    private val userRepository: UserRepository,
    private val attributeEstimator: AttributeEstimator
) : Logging {

    /**
     * Estimates the age of the user given a list of all
     * their messages, using an Age Model from HuggingFace.
     * This estimated age is then saved to the DB
     * */
    Jacqueline Dobreva-Skevington *
    @Scheduled(cron = "0 0 1 * * ?")
    @Transactional
    fun getAgeEstimate() {
```

Figure 79 Automatically Scheduled evaluator in Spring, using Dependency Injection.

Above we can see that scheduling a task is as simple as using the *@EnableScheduling* annotation, along with the *@Scheduled* annotation above the necessary method. Here, this task is scheduled to run once a night. The task is also annotated with *@Transactional*, another out-of-the-box Spring feature. In traditional applications, transactions need to be managed manually with corresponding commit and rollback utilities specified for each function to maintain Database safety. With Spring, the *@Transaction* annotation ensures that if any error occurs while executing the function, the database state is safely rolled back, preventing corruption, especially important in a production environment. Finally, dependency injection is also displayed in the constructor of the class, where parameters are automatically injected via Spring,

increasing the readability of the code, and decreasing the implementation time, along with any errors that may occur from having to construct manually.

Finally, Spring also provides tools and features needed for test writing, for instance providing the *@Autowired* annotation for dependency injection withing the test framework (JUnit doesn't allow class constructors), as well as the *@SpringBootTest* annotation which allows for Spring functionality to be ported to the JUnit test framework, reducing the complexity of test writing, and allowing modular functionality verification.

Figure 80 Spring tests used in SMS-Services to check send message functionality.

13.2.2 Hibernate and JPA

Through using Spring, we gain access to the Hibernate JPA which allows for the SQL Database to be accessed through a no-SQL-like query interface, keeping the speed of an SQL database with the ease of understanding of no-SQL queries.

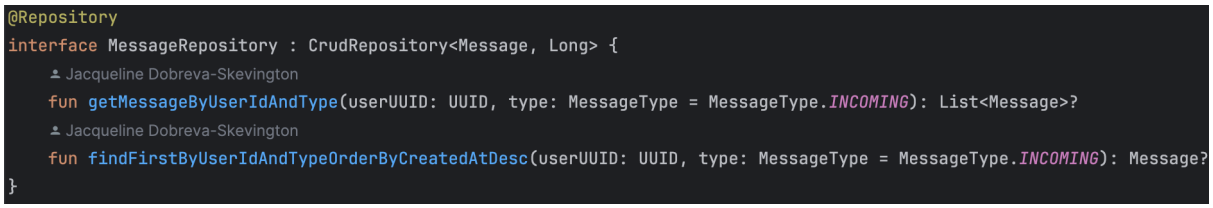


Figure 81 The Message Repository

Here we can see this in action – for instance, the second function allows us to retrieve the latest message send by a specific user. Using the Spring *@Repository* annotation, CRUD operations are automatically supported, such as *messageRepository.save(message)*. This once again reduces the amount of boiler plate

code and allowing us to only implement extraneous functionality.

Hibernate JPA also allows for automatic table and constrain creation, with entities getting automatically converted into a database table, complete with the primary key and foreign key constraints.

13.2.3 Docker

Finally, the entire system runs on a Docker container created automatically using the Spring Gradle plugin task '*bootBuildImage*', ensuring it can be easily deployed to a container management environment such as AWS Elastic Container Service, which will spin up more containers to cope with potentially fluctuating demand as and when needed, meaning that the system can easily scale.

13.3 Evaluation

The chosen languages, as well as architecture behind the project allowed the solution to be implemented cleanly and efficiently whilst maintaining best design principles. By having the system set up in a docker container, the project can easily by ported over to use a cloud-based container service and deployed for production use.

14 Project Management

14.1 Task management

The project began by setting up a JIRA project with tickets on overarching tasks that needed to be done within the first term. These became epics which were split up into subtasks. The same was then done for the second and third term during the Christmas break. Start dates of tasks were not as stringently recorded as end dates, and minor tasks which took less than a day were not added onto the JIRA board to save time. Similarly, some issues remain 'unassigned' to a user as there is only one developer.

Here is the JIRA timeline view for this project showing the epics – all epics and their associated tasks are done, apart from the dissertation, which at this point of time is in progress:

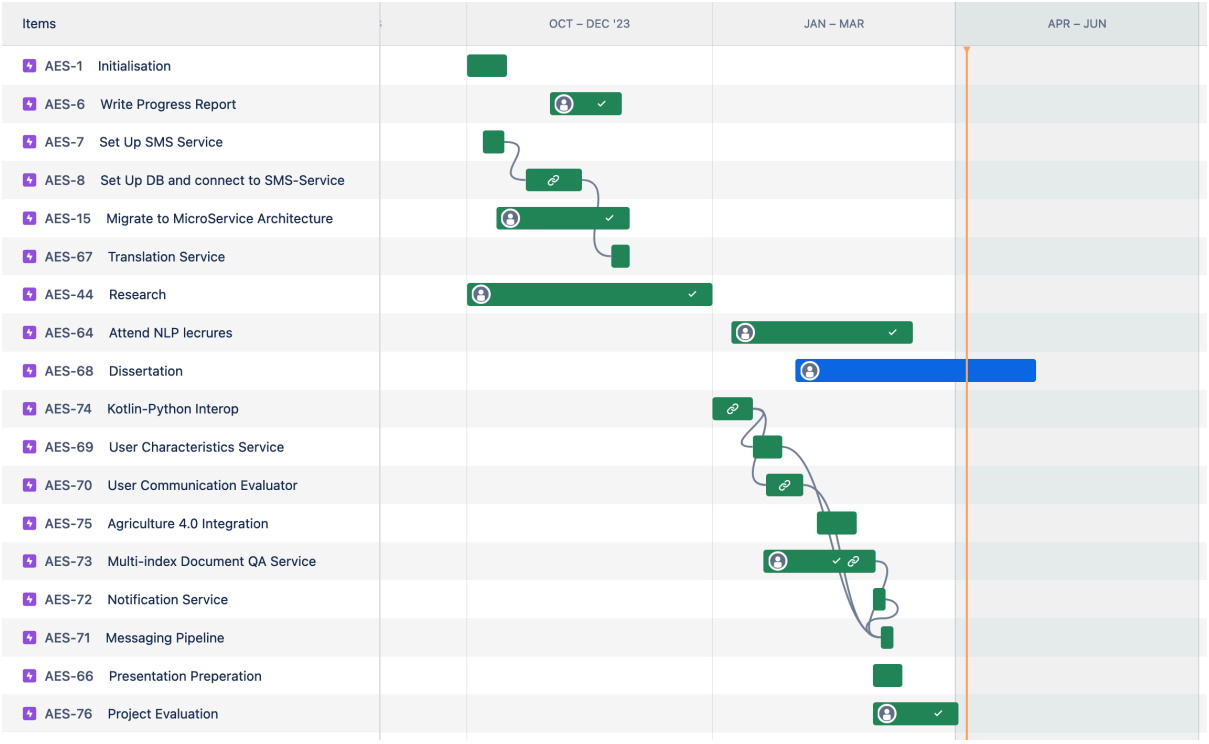


Figure 82 Screenshot from Atlassian's JIRA timeline view.

The JIRA timeline view allowed for the big picture to be seen at a glance, and dependencies to be easily managed, especially in the second term. It also meant that times where lots of work need to be completed could be seen and spread out more while keeping the project on track, especially when there were other deadlines.
Each task and epic could easily be seen using JIRAs 'Board' view. Boards are scrum boards where the project to be broken down into manageable chunks, and their progress visually seen. In the figure below, only 3 states can be seen for conciseness,

however each state breaks down into multiple other categories.

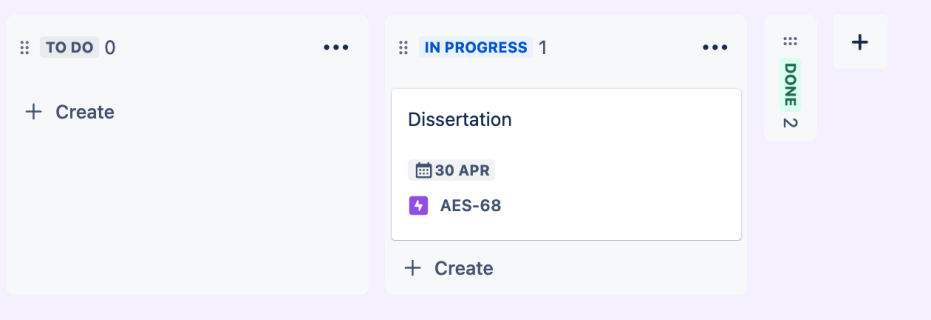


Figure 83 Tracking tasks on JIRA.

TO DO: Under TO DO are the states 'BACKLOG', which denoted tasks/epics that needed to be done, 'SCOPING' denoting tasks that are being actively researched and investigated to determine what needs to be done and 'READY FOR DEVELOPMENT' to denote tasks/epics that have been split up into manageable chunks and need no further work. This means that scope never changed midway through development, and no backtracking of features was every required, minimising delays and risks in finding that someone has done this a better way. After that is the IN-PROGRESS CATEGORY. Here we have 'SELECTED FOR DEVELOPMENT', features which have been selected to be done that day, 'DEVELOPING', features being actively worked on, 'UNIT TESTING', testing the feature works as a standalone, and finally 'INTEGRATION TESTING', testing that when put together with the rest of the codebase, all still works as expected. Integration testing was either manual 'smoke testing', where key functionality was tested to still be working correctly, or 'regression testing' for larger potentially breaking changes, where the whole system is tested end-to-end, and checked that it works the same as the old version, except where there are expected changes. Of course, during this time bugs can be found, so any of the testing stages could be moved to 'BUG-FIX NEEDED', which was a TO DO state. Any 'BUG FIX NEEDED' tickets could be moved to the IN PROGRESS state by changing to 'BUG FIXING', which would then go back through 'UNIT' and 'INTEGRATION' testing. This cycle can be repeated as many times as necessary until all functionality works as expected. During testing, if any other bugs were found in the system not related to the ticket, a 'Bug' ticket was created rather than a 'task' ticket. Bug tickets were created directly into the 'READY FOR DEVELOPMENT' state as they required no scoping. Finally, after all development and testing, tasks/bugs/epics could be moved into the DONE state, at which point no further action was required.

Type	Key	Summary	Status	Category	Due date	Priority	Created	Updated
	AES-1	Initialisation	DONE		15 Oct 2023	=	17 Oct 2023	26 Nov 2023
	AES-6	Write Progress Report	DONE		27 Nov 2023	=	7 Nov 2023	1 Mar 2024
	AES-7	Set Up SMS Service	DONE		14 Oct 2023	=	7 Nov 2023	26 Nov 2023
	AES-8	Set Up DB and connect to SMS-Service	DONE		12 Nov 2023	=	7 Nov 2023	1 Mar 2024
	AES-18	Read about Sping Boot in more depth	DONE		29 Oct 2023	=	7 Nov 2023	26 Nov 2023
	AES-19	Read about JPA in more depth	DONE		1 Nov 2023	=	7 Nov 2023	26 Nov 2023
	AES-20	Read about integrating with MySQL in more depth	DONE		1 Nov 2023	=	7 Nov 2023	26 Nov 2023
	AES-21	Set up Docker container	DONE		30 Nov 2023	=	7 Nov 2023	1 Mar 2024
	AES-22	Link services to docker	DONE		6 Dec 2023	=	7 Nov 2023	1 Mar 2024
	AES-23	Map out basic db table architecture	DONE		4 Nov 2023	=	7 Nov 2023	26 Nov 2023
	AES-24	implement dbs	DONE		5 Nov 2023	=	7 Nov 2023	26 Nov 2023
	AES-25	Add entities to services	DONE		4 Nov 2023	=	7 Nov 2023	26 Nov 2023
	AES-15	Migrate to MicroService Architecture	DONE		30 Nov 2023	=	7 Nov 2023	1 Mar 2024
	AES-67	Translation Service	DONE		30 Nov 2023	=	1 Mar 2024	1 Mar 2024

Figure 84 JIRA list view with an expanded epic showing individual tasks.

Daily work was selected from the 'READY FOR DEVELOPMENT' board. Research and other non-development work was tracked in JIRA for ease of seeing everything together, and non-applicable steps such as 'UNIT TESTING' were skipped. To balance this with other work, everything was tracked on the 'Todoist' extension.

14.2 Git

When developing, changes were stored in the form of commits using git as a source control system. This had the benefit of backing up work, as well as allowing for more fine-tuned version control management. This project followed 'trunk-based' development practices, where small, frequent updates were merged to a main 'trunk' branch that was always bug free.

Feature branches were created from the main 'trunk' branch, each linking to an Epic or large task with multiple sub tasks. Commits were made on each corresponding feature branch. Once the feature branch was closed and unit tested, it was merged back into the trunk branch for integration testing. Using this approach meant that multiple aspects of the system could be worked on at once, which was especially useful when something was particularly difficult on one branch and so a rest was needed, or there was an issue on one branch which had no solution yet. This meant that work on the project was never blocked and could always keep going, and if there was ever a major mistake, the branch could just be deleted, as the 'trunk' brank was always stable and bug free.

In the figure overleaf, we can see a screenshot of git history in IntelliJ (an IDE for JVM-based languages with integrated git support) taken during week 9 of Term 2.

Corresponding with the JIRA board, we can see that this was at the tail end of Agriculture 4.0 integration and the llama-index flow, but in the middle of when the Notification Service and Messaging Pipeline were being done.

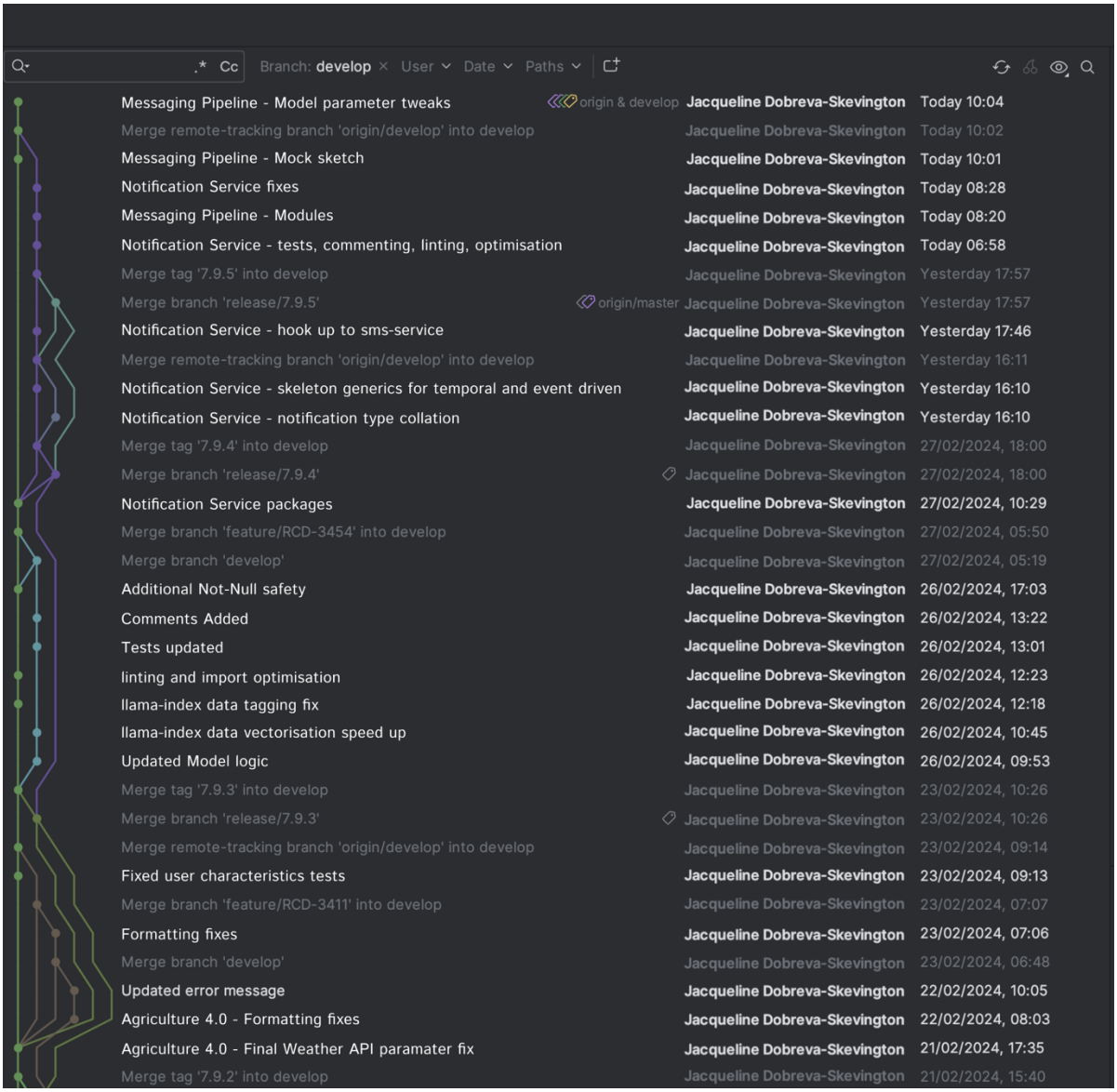


Figure 85 Screenshot of git history in IntelliJ taken during week 9 of Term 2.

14.3 Risk Management Review

During weeks 1 and 2 of the project, much research was done on SMS-Based AES' in preparation for the specification submission. However soon after submission, the research field Agriculture 4.0 was discovered, where some authors had created decision systems for the crop cycle. It was decided that simply taking this and turning its output into SMS would not be enough for a 3rd year project, and therefore further research into decision systems and authorship characteristics were done, and a RAG-based

approach incorporating user suggestibility and agriculture 4.0 was chosen, with an emphasis on making a holistic global solution. This was part of the risk "Project is not novel or complex enough, or there has been significant existing research coming to same conclusions". Due to this change, more time was spent on research than developing than was planned for Term 1, however, due to the contingency weeks put into timetable for December, there was enough time to catch up in terms of development.

The risk "Personal Computer Corruption" set out in the specification was also encountered during Term 1, when the laptop began to work for smaller and smaller chunks of times before crashing, until eventually refusing to turn on at all. While the laptop was eventually somewhat fixed (and then turned into a server for this project), it was only fixed towards the middle of term two, meaning if there was no risk management strategy in place, a whole term would have passed with no development – extremely detrimental to the project. However, at the start of Term 1, permission was obtained from work to use a work laptop for the project in the event of personal computer corruption, and everything was placed in a private repository on GitHub. Commits were also small and frequent, so at the point the computer stopped working, everything was easily set up on the work laptop and no work was lost.

14.4 Evaluation

Overall, the project was kept on track through weekly meetings with the supervisor, as well as the use of JIRA and 'todoist' to make sure everything remained on track. Early scoping of features meant that there was little to no scope creep, and never any backtracking. The use of trunk-based development was extremely useful in ensuring the project never came to a standstill, and having a thorough risk management strategy, as well as leaving blocks of contingency time, meant that issues were successfully overcome, leading to a project finished on time, and with all necessary features.

All objectives set out in the progress report were met bar 3 – one being no longer applicable to the project (talking about a Machine Learning based knowledge base), one being about creating a voice-based extension on the project which can easily be done via using out of the box solutions such as Twilio. The final objective talked about incorporating local dialect translation. As discussed later in the further work section, this is a dedicated research area of its own. All unmet objectives were in the "Could" or "Would" section, so overall, the project has been a success.

15 Conclusions

The need for an effective AES is paramount in closing agricultural yield gaps between developed and developing countries, especially as we deal with growing threats to agriculture [8]. SMS-based AES are a promising way forward, and provide a cheap, scalable platform with which smallholders can access information.

This project presents the first SMS-based AES made using RAG-based LLMs. Initial results indicate high accuracy and usability, while newly incorporating user conversation and natural language input, making the solution comfortable and familiar to use. The use of RAG overcomes traditional knowledge bottleneck issues, and the project has created the first global AES for thousands of crops, for the whole crop cycle, creating a one-stop solution for smallholders.

Leveraging Agriculture-4.0 principles, the project produces output tailored to each user on a granular per smallholding basis, allowing the smallholder key real-time insights specific to them, engaging users not only through pull messaging, but also sending them notifications with critical information drawing from behavioural economics principles.

The project also addresses other issues outside information dissemination, namely smallholders not acting on information suggested to them, by presenting a novel user suggestibility concept within an agricultural context, drawing from research into user mirroring, as well as ADSS. Initial results indicate users are on average 1.5 times more likely to implement these tailored responses.

Through the implementation of this project, there were also chances to collate data which will be made into RESTful APIs free for the public to use within their own projects, as well as developing a lightweight queue solution for interacting between microservices, and a Kotlin-Python interoperability module for dual-language projects, which will be published as a Gradle plugin. The project also suggests an extension to the seed rate formula to account for both location and crop, providing more accurate and realistic results than existing solutions.

Overall, this project presents a way forward with AES, with multiple contributions to the field. The ideas presented will increase the quality of farming, and the quality of life across developing countries, bringing the farming community which we all rely on into the 21st century.

16 Areas of Further Research

16.1 Short-Text Authorship Characteristics

In improving the user suggestibility of messages, the short text authorship characteristics problem was set out, stating the difficulty in extracting authorship characteristics from short corpora of texts. This project attempts to mitigate some of this by combining messages together, but this comes with accuracy limitations in how the text is combined and how that affects the overall result. There is much future scope for work in this area, both with authorship matching, and with extracting authorship characteristics such as age and gender. This has many applications, especially given the increase in the use of social media to conduct crime [69]. Being able to accurately determine authorship characteristics of short text may mean finding key suspects. In terms of SMS-based AES, having a more accurate extraction of authorship characteristics would mean messages can be tailored better to each user, and therefore increase their user suggestibility more, thus enhancing the impact of deploying such a system.

16.2 Short-Text User Engagement

This project has discussed the importance of finding ways of increasing user engagement in scenarios where the UI cannot be changed – primarily with textual responses and prompts. This has many applications – from health care, government emergency messages, and SMS marketing campaigns. This project suggests user mirroring as one possible solution, alongside using some principles of behavioural economics, and has implemented this with success, but it would be interesting to consider other possible psychological measures or user engagement techniques, as well as conducting more larger scale rigorous studies. Effective methods in increasing short-text user engagement would increase the user suggestibility of SMS-based AES, and therefore enhance their impact.

16.3 Individual Cultural Rules and Social Biases

In extent to qualitative user tailoring, incorporating cultural rules of different regions, as well as internalised social biases of different regions may help increase user suggestibility. For instance, elders in some societies may be used to be treated with more respect, and therefore would find it more settling to receive more deferent messages, whereas elders from a different society may find this odd and not enjoy engaging with the system. It would be interesting to map different cultural and societal

rules in different regions, determining common groups these apply to. It would also be interesting to see about creating models as to how these will change in the future and how they are impacted by globalisation and migration patterns. Given this information, extra tailoring could be added to the project further enhancing user suggestibility and the potential impact of this project.

16.4 Local Dialect Translation

As discussed in the implementation section of this dissertation, the project supports a handful of languages provided by LibreTranslate. Of course, moving to a paid translation API, such as Googles' Cloud Translation API, would increase the reach of this project, however many local communities have their own dialects and languages. Taking the United Kingdom as an example, we have English, Welsh, Irish and Scots Gaelic as our main languages. These are indeed supported by the Cloud Translation API. However, lower resource languages such as Cornish, Manx, Shelta and Angloromani are not supported. These languages make up about 122,000 speakers within the UK, who may find it more difficult to communicate in the languages supported. Outside of languages, there are more than 40 different dialects [70] each of which may have unique words, word order and spelling. For instance, a farmer in Ulster Scots is a 'culchie' [71] which may not be recognised by the system. The main difficulty with dialect is the lack of corpora available to train models with, making this an active research area, especially with the rise in endangered languages, and the drive to preserve them [72]. Incorporating this research within SMS-based AES would mean projects have a larger reach, and smallholders would feel more comfortable communicating in their native language/dialect, and be able to understand suggestions and advice better, therefore making systems more effective.

16.5 Automated IVR solutions

With the increase of feature phones, IVR AES have become more accessible alongside SMS-based solutions. While users cannot see their conversational history, it does overcome a large barrier – literacy. This project does attempt to overcome some literacy barriers by tailoring to a user's literacy level, and the project itself may be converted into an IVR solution by using an out-of-the-box provider such as Twilio; however, IVR may come with its own initial challenges and potential solutions this project has not considered – for instance, dealing with differing accents and how that may affect accuracy. Therefore, more work within this area, utilising findings from this project, may allow a solution capable for people with limited/no literacy, making information

dissemination more accessible, while retaining automation.

17 Legal, Social and Ethical Considerations

The studies in this project have been performed under the statement "If your project is not a research project, and your data from human participants are limited to interviews / questionnaires / consultations with stakeholders (for example, if you ask acquaintances to try out and give you feedback on your software), then you do not need ethical approval." [73]

18 Licence

19 References

19.1 Bibliography

[1] A. Gioia, "Small Farms in Europe: Time for a Re-Definition," 04 2017. [Online]. Available: https://www.accesstoland.eu/IMG/pdf/comparative_analysis_of_small_farms_in_europe.pdf. [Accessed 05 11 2023].

[2] Danjumah, Asiamah, Tham-Agyekum, Ibraham and Mensah, "Dynamics of agricultural extension delivery services to rice farmers in Ghana," *Heliyon,* vol. 10, no. 5, 2024.

[3] J. Anderson, G. Feder, A. Willet and W. Zijp, "Chapter 44," in *Agricultural Extension: Generic Challenges and Some Ingredients for Solutions*, The World Bank, 1999.

[4] PC MAG, "feature phone," PC MAG, [Online]. Available: https://www.pcmag.com/encyclopedia/term/feature-phone. [Accessed 13 02 2024].

[5] A. Supren, N. Mahalik and k. Kim, "A review on application of technology systems, standards and interfaces for agriculture and food sector," *Computer Standards and Interfaces,* vol. 35, no. 4, pp. 355-364, 2013.

[6] K. Mandi and N. M. Patnaik, "Mobile apps in agriclture and allied sector: An extended arm for farmers," *Agriculture Update,* vol. 14, no. 4, pp. 334-342, 2019.

[7] V. Ricciardi, N. Ramankutty, Z. Mehrabi, L. Jarvis and B. Chookolingo, "How much of the world's food do smallholders produce?," *Global Food Security,* vol. 17, pp. 64-72, 2018.

[8] Food and Agriculture Organisation of the United Nations, "The future of food and agriculture: trends and challenges," Rome, 2017.

[9] United Nations, "United Nations: Department of Economic and Social Affairs,"

[Online]. Available: https://sdgs.un.org/goals/goal2. [Accessed 28 03 2024].

[10] M.-E. R. A. L. D. H. Elizabeth Eldridge, "Expanding Perspectives on the Poverty Trap for Smallholder Farmers in Tanzania: The Role of Rural Input Supply Chains," *Sustainability,* vol. 14, no. 9, 2022.

[11] One Acre Fund, "The Life You Can Save," One Acre Fund, End Hunger & Malnutrition, 2 10 2019. [Online]. Available: https://www.thelifeyoucansave.org/charity-stories/putting-smallholder-farmers-first-in-the-fight-to-end-hunger/. [Accessed 1 4 2024].

[12] R. Townsend, "Ending poverty and hunger by 2030 : an agenda for the global food system (English)," 2015. [Online]. Available: http://documents.worldbank.org/curated/en/700061468334490682/Ending-poverty-and-hunger-by-2030-an-agenda-for-the-global-food-system. [Accessed 28 03 2024].

[13] F. M. Aragón, D. Restuccia and J. P. Rud, "Are small farms really more productive than large farms?," *Food Policy,* vol. 106, 2022.

[14] syngenta, "Agricultural Extension," [Online]. Available: https://www.syngentafoundation.org/agricultural-extension. [Accessed 05 4 2024].

[15] F. van Rijn, E. Nkonya and A. Adekunle, "The impact of agricultural extension services on social capital: an application to the Sub-Saharan African Challenge Program in Lake Kivu region," *Agric Hum Values,* vol. 32, pp. 597-615, 2015.

[16] Wikipedia, "Wikipedia," Wikipedia, 15 10 2023. [Online]. Available: https://en.wikipedia.org/wiki/Agricultural_extension. [Accessed 1 4 2024].

[17] F. N. Che, K. D. Strang and N. R. Vajjhala, "Introduction to Agricultural Information Systems," in *Opportunities and Strategic Use of Agribusiness Information Systems*, IGI Global, 2020, p. 12.

[18] C. Ifejika Speranza, B. Kiteme and M. Opondo, "Adapting public agricultural

extension services to climate change: Insights from Kenya," in *Amsterdam Conference on the Human Dimensions of Global Environmental Change*, Amsterdam, 2009.

[19] A. E. Agwu, M. Suvedi, C. Chanza, K. Davis, A. Oywaya-Nkurumwa, M. N. Mangheni and P. Sasidhar, "Agricultural Extension and Advisory Services in Nigeria, Malawi, South Africa, Uganda, and Kenya," Alliance for African Partnership, Michigan, 2023.

[20] A. AL-Sharafat, M. Altarawneh and E. Altahat, "Effectiveness of Agricultural Extension Activities," *American Journal of Agricultural and Biological Sciences,* vol. 7, no. 2, pp. 194-200, 2012.

[21] A. W. v. d. Ban, "Different ways of financing agricultural extension," *Agricultural Research and Extension Network,* vol. 106, 2000.

[22] D. C. Parker, K. H. Jacobsen and M. K. Komwa, "A Qualitative Study of the Impact of HIV/AIDS on Agricultural Households in Southeastern Uganda," *International Journal of Environmental Research and Public Health,* vol. 6, no. 8, pp. 2113-2138, 2009.

[23] GFRAS, "NOTE 18: Using Radio in Agricultural Extension," GFRAS, [Online]. Available: https://www.g-fras.org/en/good-practice-notes/using-radio-in-agricultural-extension.html?showall=1. [Accessed 01 04 2024].

[24] GFRAS, "NOTE 22: Edutainment TV for disseminating information about agriculture," GFRAS, [Online]. Available: https://www.g-fras.org/en/good-practice-notes/22-edutainment-tv-for-disseminating-information-about-agriculture.html?showall=1. [Accessed 01 04 2024].

[25] S. M. N. Khalid and S. Sherxad, "AGRICULTURAL EXTENSION MANUAL FOR EXTENSION WORKERS," FAO, 2019.

[26] L. Hollis, "Plantwise," CABI, 30 05 2022. [Online]. Available: https://blog.plantwise.org/2022/05/30/agricultural-mobile-apps-strengthening-agricultural-extension/. [Accessed 08 04 2024].

[27] H. S. Kassem, B. A. Alotaibi, M. Muddassir and A. Herab, "Factors influencing farmers' satisfaction with the quality of agricultural extension services," *Evaluation and Program Planning,* vol. 85, 2021.

[28] N. T. Krell, S. A. Giroux, Z. Guido, C. Hannah, S. E. Lopus, K. K. Caylor and T. P. Evans, "Smallholder farmers' use of mobile phone services in central Kenya," *Climate and Development,* vol. 13, no. 3, pp. 215-227, 2020.

[29] L. K. Narine, A. Harder and G. Roberts, "Farmers' intention to use text messaging for extension services in Trinidad," *The Journal of Agricultural Education and Extension,* vol. 25, pp. 1-14, 2019.

[30] PxD, "What We Do," PxD: Precision Development, [Online]. Available: https://precisiondev.org/what-we-do/our-motivation/. [Accessed 18 03 2024].

[31] GatewayAPI, "Pricing," GatewayAPI, [Online]. Available: https://gatewayapi.com/pricing/. [Accessed 10 04 2024].

[32] D. Mhlanga and E. Ndhlovu, "Digital Technology Adoption in the Agriculture Sector: Challenges and Complexities in Africa," *Human Behaviour and Emerging Technologies,* 2023.

[33] PxD, "Key Metrics," PxD: Precision Development, [Online]. Available: https://precisiondev.org/pxd-global-dashboard/. [Accessed 19 02 2024].

[34] PxD, "Paddy – PxD's User Communications Platform," PxD: Precision Development, [Online]. Available: https://precisiondev.org/pxds-user-communications-platform-paddy/. [Accessed 19 04 2024].

[35] PxD, "MoA-INFO – taking digital agriculture to Kenya's heartland," PxD: Precision Development, [Online]. Available: https://precisiondev.org/project/moa-info/. [Accessed 19 04 2024].

[36] O. S. Sowolw, "Leveraging Large Language Models for Improving Agricultural Extension in Nigeria," *DeepLearningIndaba 2023 Conference,* 3 7 2023.

[37] A. Tzachor, M. Devare, C. Richards, P. Pypers, A. Ghosh, J. Koo, S. Johal and

B. King, "Large language models and agricultural extension services," *Nature food,* vol. 4, pp. 941-948, 2023.

[38] A. Balaguer, V. Benara, R. L. d. F. Cunha, R. d. M. E. Filho, T. Hendry, D. Holstein, J. Marsman, N. Mecklenburg, S. Malvar, L. O. Nunes, R. Padilha, M. Sharp, B. Silva, S. Sharma and V. As, "RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture," *Computation and Language,* 2024.

[39] F. N. Ford, "Decision Support Systems and Expert Systems: A comparison," *Information & Managment,* vol. 8, no. 1, pp. 21-26, 1985.

[40] B. Fazlollah, M. Parikh and S. Verma, "Adaptive Decision Support Systems," *Decision Support Systems,* vol. 20, no. 4, pp. 297-315, 1997.

[41] Sitaram, K. Ahuja, H. Diddee, R. Hada, M. Ochieng, K. Ramesh, P. Jain, A. Nambi, T. Ganu, S. Segal, M. Axmed, K. Bali and Sunayana, "MEGA: Multilingual Evaluation of Generative AI," 2023.

[42] Commoditic, "Agricultural Commodities Prices API," Commoditic, [Online]. Available: https://commoditic.com/agricultural-commodities-prices-api/. [Accessed 05 04 2024].

[43] IR-4, "Crop Group Tables," IR-4, [Online]. Available: https://www.ir4project.org/fc/crop-grouping/crop-group-tables/. [Accessed 20 03 2024].

[44] N. Richards, "Tamar Organics Growers Price List 2023," Tamar Organics, [Online]. Available: https://tamarorganics.co.uk/wp-content/uploads/2022/11/Tamar-Organics-Price-List-2023.pdf . [Accessed 17 02 2024].

[45] M. Hasanuzzaman. [Online]. Available: https://hasanuzzaman.weebly.com/uploads/9/3/4/0/934025/seed_rate_calculation.pdf. [Accessed 20 02 2024].

[46] SER; INSR; RBGK, "Seed Information Database (SID)," SER; INSR; RBGK,

2023. [Online]. Available: https://ser-sid.org/about. [Accessed 04 03 2024].

[47] SID, "Zea mays L.," [Online]. Available: https://ser-sid.org/species/e3f1b34a-7182-4310-a93c-6e8074b0814a. [Accessed 28 04 2024].

[48] R. Cima, M. D'Antuono and W. Anderson, "The effects of soil type and seasonal rainfall on the optimum seed rate for wheat in Western Australia," *Australian Journal of Experimental Agriculture,* vol. 44, no. 6, pp. 585-594, 2004.

[49] Trefle API, "Trefle," [Online]. Available: https://trefle.io/. [Accessed 10 03 2024].

[50] FAO, "Welcome to ECOCROP," [Online]. Available: https://ecocrop.review.fao.org/ecocrop/srv/en/home. [Accessed 11 02 2024].

[51] J. Hedley, " jsoup: Java HTML Parser," [Online]. Available: https://jsoup.org/. [Accessed 13 03 2024].

[52] H. M. Cathey, "USDA Plant Hardiness Zone Map," United States Department of Agriculture.

[53] M. C. Pee, B. L. Finlayson and T. A. Mcmahon, "Updated world map of the Köppen-Geiger climate classification," *Hydrology and Earth System Sciences Discussions,* vol. 4, no. 2, pp. 439-473, 2007.

[54] LlamaIndex, "Turn your enterprise data into production-ready LLM applications," [Online]. Available: https://www.llamaindex.ai/. [Accessed 21 01 2024].

[55] Ramanathan and Muzammil, "RAG Wars – Llama Index vs. Langchain Showdown," USEReady, 20 02 2024. [Online]. Available: https://www.useready.com/blog/rag-wars-llama-index-vs-langchain-showdown. [Accessed 01 04 2024].

[56] J. Ip, "Quick Introduction," DeepEval, 25 04 2024. [Online]. Available: https://docs.confident-ai.com/docs/getting-started. [Accessed 26 04 2024].

[57] B. Adolph, M. Allen, E. Beyuo, D. Banuoku, B. S., T. Bourgou, N. Bwanausi, F. Dakyaga, E. Derbile, P. Gubbels, B. Hie, C. Kachamba, G. Naazie, E. Niber, I.

Nyirengo, S. Tampulu and A. Zongo, "Supporting smallholders' decision making: managing trade-offs and synergies for sustainable agricultural intensification," *International Journal or Agricultural Sustainability,* pp. 456-473, 2020.

[58] G. L. Kreps and L. Neuhauser, "Artificial intelligence and immediacy: Designing health communication to personally engage consumers and providers," *Patient Education and Counseling,* vol. 92, no. 2, pp. 205-210, 2023.

[59] D. Bilal and J. K. Barfield, "Hey There! What Do You Look Like? User Voice Switching and Interface Mirroring in Voice-Enabled Digital Assistants (VDAs)," *Proceedings of the Association for Information Science and Technology,* vol. 58, no. 1, pp. 1-12, 2021.

[60] Y. Liao and J. He, "The racial mirroring effects on human-agent in psychotheraeuthic conversation," *IUI'20: Proceedings of the 25th International Conference in Intelligent User Interfaces,* pp. 430-442, 2020.

[61] M. Koppel, J. Schler and S. Argamon, "Authorship Attribution: What's Easy and What's Hard?," *Journal of Law and Policy,* vol. 21, no. 2, 2013.

[62] Unicef, "Primary Education," Unicef, [Online]. Available: https://www.unicef.org/education/primary-education. [Accessed 22 04 2024].

[63] SME, "How many words do we text in a lifetime?," Text Anywhere, 5 12 2021. [Online]. Available: https://www.textanywhere.com/resources/blog/technology/how-many-words-do-we-text-in-a-lifetime/. [Accessed 23 10 2023].

[64] Wikipedia, Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests. [Accessed 06 12 2023].

[65] Mbo'o-Tchouawou, M. Colverson and K. E, Increasing access to agricultural extension and advisory services: How effective are new approaches in reaching women farmers in rural areas?, ILRI (aka ILCA and ILRAD), 2014.

[66] D. Lee, "LLMs + fraud: How criminals use large language models to commit fraud," Persona, 2023. [Online]. Available: https://withpersona.com/blog/llm-fraud. [Accessed 29 03 2024].

[67] Here After, "Your stories and voice. Forever.," Here Afer, [Online]. Available: https://www.hereafter.ai. [Accessed 28 04 2024].

[68] StoryFile, "StoryFile," StoryFile, [Online]. Available: https://storyfile.com/#. [Accessed 28 04 2024].

[69] Press Association, "Social media-related crime reports up 780% in four years," The Guardian, 27 12 2012. [Online]. Available: https://www.theguardian.com/media/2012/dec/27/social-media-crime-facebook-twitter. [Accessed 26 04 2024].

[70] C. Rumsey, "A brief guide to British accents and dialects," Studio Cambridge, 14 11 2023. [Online]. Available: https://www.studiocambridge.co.uk/a-brief-guide-to-british-accents-and-dialects/. [Accessed 13 04 2024].

[71] Wikipedia, "Ulster English," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Ulster_English. [Accessed 16 04 2024].

[72] "International Decade of Indigenous Languages," [Online]. Available: https://idil2022-2032.org/. [Accessed 18 04 2024].

[73] Department of Computer Science, "Ethical Consent for Undergraduate Projects," University of Warwick, [Online]. Available: https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs310/ethics/. [Accessed 29 04 2024].

[74] Vechev, R. Staab, M. Vero, M. Balunović and Martin, "Beyond Memorization: Violating Privacy Via Inference with Large Language Models," 2023.

[75] Olaniyi, Olumuyiwa, Ogunkunle and Tajudeen, "Agricultural and Nutritional Information Needs of Arable Crop Farmers in Ondo State, Nigeria," *Journal of Agricultural Extension,* vol. 22, p. 9, 2018.

[76] D. Power, "The Global Development Research Center," [Online]. Available: https://www.gdrc.org/decision/dss-types.html. [Accessed 1 11 2023].

[77] "Talking Tech Health," 15 10 2020. [Online]. Available: https://www.talkinghealthtech.com/glossary/decision-support-system. . [Accessed 1 11 2023].

[78] M. Fairlie, "Business.com," 21 02 2023. [Online]. Available: https://www.business.com/articles/decision-support-systems-dss-applications-and-uses/. [Accessed 1 11 2023].

[79] M. Deveci, A. R. Mishra, I. Gokasar, P. Rani, D. Pamucar and Ö. E., "A Decision Support System for Assessing and Prioritizing Sustainable Urban Transportation in Metaverse," *IEEE Transactions on Fuzzy Systems,* vol. 31, no. 2, pp. 475-484, 2023.

[80] D. H. Galeon, P. G. J. Garcia and T. D. Palaoag, "SMS-Based ICT Tool for Knowledge Sharing in Agriculture," *International Journal on Advanced Science Engineering Information Technology,* vol. 9, no. 1, 2019.

[81] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kutter, M. Lewis, W. Yih, T. Rocktaschel, S. Riedel and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Computation and Language,* 2021.

[82] M. Faheem, K. Wassif, H. Bayomi and S. Abdou, "Improving neural machine translation for low resource languages through non-parallel corpora: a case study of Egyptian dialect to modern standard Arabic translation," *Scientific Reports,* vol. 14, 2024.

20 Appendix A

1) Write me 10 sentences. These should have some of the following information in them: SMALLHOLDING_SIZE, LOCATION_CITY, LOCATION_COUNTRY, NAME, MAIN_CROP. An example message could be: "My name is John, I live in Wales, Swansea, and I have a smallholding of 5 acres. I mainly grow wheat". The main crop should be specified by name, and NOT be livestock.
2) Write me another 10 sentences. The messages should have some irrelevant information in as well.
3) Write me another 10 messages, capitalisations and other grammar rules should not be strictly followed.
4) Write me another 10 messages focusing on a wide range of crops.
5) Write me another 10 messages, with different units of measurement in them for smallholding size such as meters and yards.
6) Write me another 10 messages with various lengths of names in various formats. For example: John Smith, John Doe-Smith, John Fred Bob Charles Doe-Smith Jr.
7) Write me another 10 messages, which have an agricultural question in the middle of the sentence.
8) Write me another 10 messages, including general conversation and system feedback.
9) Write me another 10 messages, including a refer a friend sentence in each of them and mentioning at least one information category in them. For example: “My smallholding is 1 km2. I’d like to grow wheat on half of it. Can you please also send messages to my friend Jose, his smallholding is 3 acres.”.
10) Write me another random collection of 10 messages using all the above.