// First slide

Hello and welcome! Today I will be making a presentation for the development project in the intelligent agents' module at Essex University.

// Index of contents

In this presentation, I will start by giving a quick introduction, then a project overview. After that we'll discuss the different technologies & libraries used to build the system and walk through the decision-making process in the system architecture. Challenges faced will be covered, and the presentation will include pieces of the code used in building the system and a couple of testing demonstrations. After that a short discussion of findings and a summary of future enhancements ideas will be mentioned. References of the stats will be available at the last slide.

// Introduction

The digital forensics field has been highly affected by the recent advances in technology. In the last few years, the data quantity within each storage device has grown significantly. Leaving the manual systems outdated and not able to handle that amount of data. The image on the left demonstrates the statistics of data growth within a range of 15 years. As you can see It is expected that by 2025 it will reach 181 zettabytes.

Automation is one of the proposed solutions. The main motive to use it is to complete entire tasks or processes with minimal or without human intervention at all. It has already been proven to complete the jobs faster and reduce significant costs.

Among the top-used approaches for automating digital forensics is the multi-agent systems. The reason behind this is their ability to breakdown complex forensics into small tasks that are shared between agents. These agents collaborate and communicate with each other to analyze file systems, collect evidence, and produce reports.

On Unit 6 we have decided to go through with the Digital Forensics task to build our Intelligent-agents system. The final decision was to build a multi-agent system capable of identifying specific file types that are text types in a filesystem, producing a report of statistics and archiving the results for future analysis. The project aims to automate the digital forensics tasks such as file type identification, mime type verification and text summarization and reporting.

// Overview

We have decided to use Python as a coding language due to its simplicity and the various intelligent agent libraries that can be used with it.

To achieve our required goals with the system, a couple of techniques and libraries were chosen:

For example, The Spacy library is an open-source python library designed for natural language processing. It is known for being efficient in handling text input from human users in multiple languages. It will be employed to interpret the user requirements from plain English into a set of values passed between the agents.

The Transformers library runs on advanced machine learning models that are pretrained. It is mainly employed in natural language processing models. In this system it will be used to summarize the text inside each file and produce a combined summary at the end.

The “mimeTypes” library provides the possibility to extract the file extension and map it to its corresponding MIME types. It will serve to check the format in this system whether it is a valid text type or not.

Some techniques acquired in the intelligent agents’ module were also employed in this system. For example, to achieve safe and reliable communication between the agents we used the knowledge query and Manipulation language known as KQML. It is a

communication protocol used to exchange information between intelligent agents in a multi-agent system. Natural language processing is used as well as discussed.

// System Design

The system will be composed of a variety of different intelligent agents, some are Symbolic, or reactive and others are hybrid. Each agent will have a specific role in the lifecycle of the task, we have decided to keep each agent assigned to a simple small task so that the maintenance and debugging would be manageable. The agents will run in sequence with the other agents. Each time an agent is executed it does its task then passes the data to the next one.

The agents will communicate with each other using KQML messaging to pass and receive data. The image on the left shows the prototype of the user request we are testing and how it passes the message to the first agent named requirements. The image on the right shows the message being acknowledged by each agent.

// Agents' breakdown

There is a total of 7 agents that construct the system in general:

The first Agent named requirements which is a symbolic agent is the one responsible for receiving the user input in plain English. Then, using the spacy natural language processing, it is able to abstract the necessary information about the directory and filetypes which the user would like to search and summarize. It then passes that data to the next agent via a Json file.

The second agent Search, which is a reactive agent, receives the file types and the folder where to look. It then searches that directory to see if it is found {currently we are limiting the number of folders and filetypes to search for prototype purposes}. Once the search is

finished the total count of files found and their details such as paths will be passed to the next agent using an array of objects. {code demo?}

The next agent in the list is the Types agent, it is a symbolic agent that loops through the list of files found to verify their mime types. Depending on the outcome the counter of right and wrong types will be calculated and passed through to the next agent.

The text agent, which is a hybrid agent, will receive that data and loop through the values by opening each file using its path to get the text inside. It will then produce a summary using the transformer library summarizer and pass that to the two next agents via a JSON file.

The stats agent is a reactive agent that builds a summary of stats made from searching the files. It produces an array containing the total number of files found, their paths, how many had right or wrong types.

Simultaneously, the summary agent uses the preproduced summary from each file, then using that it comes up with a combined summary of all their contents.

Both the stats and the summary agent's outcome will be passed to the Report agent. It uses the transformer's natural language processing module to produce a final report in plain English.

Finally, the archive agent will save the report as a .txt file containing data in plain English into a filesystem folder. And after that the process is terminated.

// UML diagram page

The UML diagram illustrates the sequence of the run. It starts with the user input being passed to the requirements agent, then goes through each agent that completes its task, then once the report is produced the process is terminated.

// Implementation

As a prototype the system was developed in a local environment using Visual Studio code.

The libraries used had their module names stored in a text file inside the libraries folder to simplify the set-up process.

We have set up a simple KQML messaging with “ask” and “reply” actions only just for prototyping purposes. However, the general state of the protocol would need to have more actions set up to check the execution status and simulate real simultaneous messaging between the agents. The image on the right showcases the two messaging types we have now which are send and reply.

Then the files' structure consisted of a “taskExecutore” file that configuration setting the order of the run cycle. After that each agent file was stored withing the Agents directory.

The error handling process consisted of trying to catch several errors such as empty folders, invalid types, or empty text files that we needed to summarize. As you can see on the image there are a couple of errors for invalid messages or senders. And how they are being treated.

// Unit Testing

To automate our testing process, we also added some unit tests. The tests cover all agents and their tasks. The unit test library which is a built-in python module was used to test our system. The current tests cover the cases of how a valid run should generate the file and how a run with a folder that does not contain any text files should error. In this case we mocked the agents to amend their expected output to see how the overall run responds.

Two tests were created. One for the “taskExecutor“ that contains all the agents' executions, and the requirements agent on its own due to its importance interpreting the initial user input. The screenshots on the left-hand side illustrate the current unit tests built and their successful run.

// Smoke Testing

A short smoke testing was done on the app to be able to directly visualize the output. The cases were to test getting files in a folder that exists and had 2 text files, and test getting files in a folder that exists but has no text files. The two images on this slide show the output of each test.

// Challenges

There were a couple of challenges faced during the building and the testing phases of the application. For instance, the total run time was large when the number of text files found within the folder was high. The image below shows the total execution time when there are more than two files on the folder.

The second challenge was the bugs faced when corrupted or invalid files were found. That resulted in the specific agent returning invalid data that the next agent was not expecting.

The third challenge was the variety of permissions of some files and folders. Currently the agents are expected to have access permissions to all text files. In advance, they require permission to create the report inside the “savedReports” folder.

The final challenge was the dependency on external libraries. The choice between a number of different libraries to complete the task at hand was based on a couple of points such as public availability, security and performance. However, the total dependency on external libraries might come with several errors in the future and continuous updates will be required.

// Future enhancements

The system in hand can already be described as robust and reliable. However, a couple of points must be considered to ensure proper scalability.

For example, the use of advanced AI modules rather than the ones currently used might come handy in improving the overall performance, and the quality of the outputs.

The system must also be able to handle all directories and filetypes within a filesystem in the future depending on the user's request.

In addition, to prevent unsuspected errors and failures, we can implement locking into the agents. When an agent is executed, it locks the transaction until it is successfully executed. If not, it should terminate the process and output the error. This way we can ensure that first no agent will be executed before the termination of the previous one. And second it would not run unless the expected data is passed to it.

Adding more KQML message types such as "ask if" to check if there's a specific condition that applies which the next agent needs to perform. "Preprocess" could also be a message to check how different data is supposed to be treated by the agent if the sender agent needs special conditioning the that exact data.

Finally, to ensure good improvements in the future. Archiving the errors and alerting the system administrators might help to keep an eye on the users failed action and will facilitate the debugging process. The long-term goal is to achieve complete autonomous automation.

// Conclusion

To conclude, the impact of Intelligent agents is already significant in the field of digital forensics and its applications are several. The development of this multi agent system aimed to address one of the challenges in digital forensics which is file content reading and interpretations.

We utilized different libraries and technologies such as Transformers and spacy for natural language processing. These proved efficient in performing tasks such as text summarization and report generation. Despite some challenges like handling the file permissions or corrupted files or long run times especially with large datasets.

A couple of techniques within artificial intelligence such as KQML approach and natural language processing have been employed in the process.

In the future, several enhancements to this system could help improve scalability and reliability. As I mentioned in previous slides, a few examples such as using more advanced AI models, implementing locking.

Finally, although we have implemented some unit tests, ensuring full test coverage always will help to reduce the number of unexpected bugs.

Ensuring these improvements are implemented. Can play a big part to ensure that the system can scale to large datasets and handle more complex tasks while maintaining performance and reliability.

Thank you for watching this presentation. I am happy to answer any of your questions.