Intelligent Agents Individual E-Portfolio Report

Student name: Mariam Almarzooqi

Contents

Introduction and Module Overview

Intelligent Agents digital e-portfolio (hosted at: https://mariamalmarzooqiessex.github.io/portfolio/Intelligent.html)

Through this intelligent agent module, I have experienced important personal and professional growth while working on practical applications of agent based systems. The journey starts with basic concepts of agent architectures and improve through the collaborative discussions, team projects and ended in an individual project that shows the real world applicability of intelligent agent principles. This reflective report shows my learning journey, focusing particularly on the Digital Forensics Agent System that I have developed, the challenges I faced, the skills I acquired and how these experiences have shaped my understanding of intelligent systems.

The module introduced me to various agent architectures which includes simple reactive agents to complex BDI systems and provided the opportunities to apply these concepts through development work. Each unit built based on previous knowledge which creating a complete understanding of how intelligent agents can solve the real world problems. The discussions helped me connect with partners on theoretical concepts while the team and individual projects allowed me to implement theory into practice.

Agent Based Computing and Architecture

When I first encountered agent based computing in Unit 1, the concepts seems abstract and disconnected from practical applications. I understood that agents were autonomous software entities that could see their environment and act upon it, but I struggled to visualize how this would work in the real systems. The early discussions about agent characteristics like autonomy, reactivity, proactiveness and social ability were interesting theoretically, but I needed experience to truly understand their significance.

As I progressed through Units 2 and 3, learning about first order logic and various agent architectures, I start to see patterns and connections. The study of reactive architectures showed me that how agents could respond to environmental without complex reasoning, while architectures shows goal based behavior through planning and reasoning. The BDI architecture particularly helped me because it shows human decision-making processes through viewpoint about the world which representing goals and intentions as committed the plans of action.

The breakthrough came in Unit 4 when we explored hybrid architectures that combined reactive and deliberative components. This made complete sense for practical applications because real-world systems often need both quick reactive responses and thoughtful deliberative planning.

Team Collaboration and Communication Skills

Working as part of Team was an helpfull experience that taught me lessons extending far beyond technical knowledge. Our team which consisted of myself, Majed Alzaabi and Koulthoum Hassan Flamerzi. From the beginning, we established clear communication channels and created a team contract that define our responsibilities and meeting schedules.

The collaborative discussions on agent based systems and agent communication languages shows the different perspectives and interpretations. During the first discussion about agent based systems, I learned to articulate my understanding clearly while remaining open to alternative viewpoints from my peers. Reading and responding to posts from Ali Alhammadi and Koulthoum Flamerzi challenged me to think more about the assumptions representing different agent architectures.

In Unit 5 and 6, we focused on agent communication, learning about speech act theory, KQML and KIF. The exercise of creating agent dialogues between Alice and Bob in warehouse management helped me understand how agents exchange information and coordinate actions. I developed dialogues showing how agents can ask questions, provide information and negotiate using structured communication protocols.

The team throughout the project were generally stayed positive even though we faced challenges. One challenge was coordinating schedules across different time zones and personal commitments. We addressed this by being flexible with meeting times and using common communication tools effectively. Another challenge was make suring equal participation in discussions, as some team members were naturally more vocal than others.

Development of the Digital Forensics Agent System

The individual project to develop the digital forensics agent system became the important point of my learning experience in this module. This project required me to apply everything I had learned about agent architectures, communication and intelligent system design to solve a real problem in digital forensics. The forensics field needs automation because manual evidence collection is time taking, contains error and difficult to examine for legal proceedings.

I chose to implement a BDI based multi-agent architecture for multiple reasons. First, forensic investigation is goal based with clear objectives like finding all files of certain types or verifying evidence integrity. Second, forensic agents need beliefs about the current state of the file system and evidence database. Third, agents must form intentions about which actions to execute based on their goals and current situation. The BDI model provided a natural framework for organizing this complexity.

4

The system architecture improved through several iterations. Initially, I considered a simpler design where a single program handled all tasks sequentially. However, this approach lacked the scalability and failed to take advantage of concurrent processing. Then we designed a multi agent system with four specialized agents working together. The Orchestrator Agent helps with overall execution and manages the other three agents. The Discovery Agent scans file systems to find potential evidence files based on configurable criteria. The Processing Agent extracts metadata and calculates cryptographic hashes for evidence integrity. The Packaging Agent creates the encrypted evidence packages with multiple report formats.
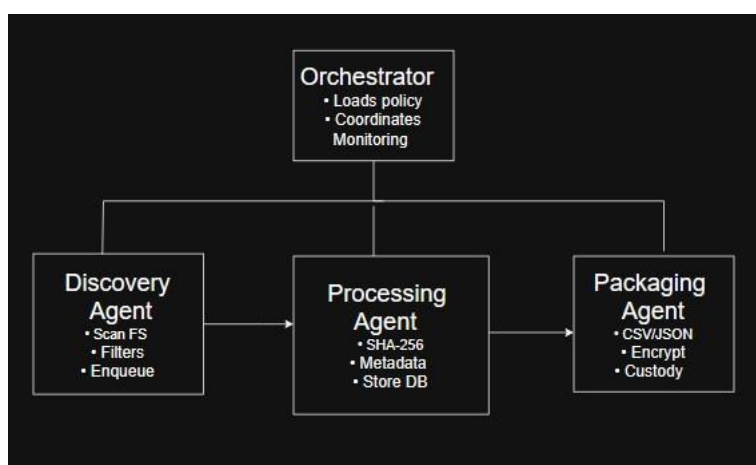

Figure 1: DFAS Multi-Agent Architecture showing the Orchestrator coordinating Discovery, Processing and Packaging agents through queue based communication

The communication between agents uses the python queues which provide thread safe message passing. When the Discovery Agent finds a file matching our criteria, it places the file path into a queue. The Processing Agent monitors this queue, retrieves file paths, and performs its work. This producer consumer pattern allows the agents to work at their own pace without blocking each other. If discovery runs faster than processing, the queue buffers the work. If processing is faster, it waits without wasting CPU resources.

I implemented a streaming approach that reads the files in 4 kilobyte blocks by updating the hash incrementally. This allows the system to process files of any size without consuming the available memory.

File type detection presented another interesting challenge. Initially, I relied on file extensions to determine file types, but this approach is fundamentally insecure because users can easily rename files. An attacker could hide evidence by changing suspicious file extensions to innocuous ones. I implemented libmagic integration, which examines file headers and content to determine actual file types regardless of extension. This required handling cases where libmagic is unavailable, so I created a fallback chain: try

libmagic first, then Python's mimetypes module, and finally a hardcoded extension mapping as a last resort.

I used sqlite because it stores the complete database in a single file, making evidence packages portable and easy to manage. The schema includes an evidence records table storing all file metadata and a chain of custody table documenting every action taken during collection. Each chain of custody entry includes who performed the action, when it occurred, and what changed. The entries are linked through cryptographic hashes, making tampering detectable.

Creating the Packaging Agent involved learning about encryption and evidence packaging standards. Digital forensics requires that evidence packages remain tamper-evident and maintain integrity. I implemented AES-GCM encryption, which provides both confidentiality and authentication. The authentication tag allows anyone with the decryption key to verify that the package has not been modified since creation. The agent exports evidence in multiple formats including CSV for human review in Excel, JSON for programmatic access by other tools, and the original SQLite database for complex queries and analysis.

Technical Challenges and Problem Solving

Developing DFAS presented the numerous technical challenges that required research, experimentation and creative problem solving. These challenges taught me important lessons about software engineering, error handling and the gap between theoretical designs and practical implementations.

Cross platform compatibility proved more difficult than expected. My development environment was windows, but I wanted the system to work on linux and macos as well. File paths, permissions and ownership work differently across operating systems. File ownership on Windows involves security identifiers while Unix uses numeric user IDs. I addressed these issues using python pathlib module which abstracts platform differences and by writing platform specific code wrapped in conditionals that check the operating system.

Performance optimization became important when testing with larger datasets. Processing files took long when dealing with hundreds of files. The queue based multi-agent design naturally supports the concurrent processing because discovery and processing happen together. While the Discovery Agent scans directories, the Processing Agent works on files already in the queue. This pipelining dramatically improved throughput. In my testing, the system processed seventeen files in approximately three seconds, which extrapolates to over 300 files per minute for similar-sized files.

Memory management required careful attention when processing large files. My initial implementation read entire files into memory to calculate hashes, which worked fine for small documents but failed catastrophically with large video files or disk images. I implemented streaming reads that process files in small chunks, updating the hash incrementally. This change reduced memory usage from potentially gigabytes to a constant few kilobytes regardless of file size.

By implementing libmagic integration that examines actual file content, the system detects true file types even when extensions are misleading. Testing this functionality involved creating files with mismatched extensions and verifying that libmagic correctly identified them.

Testing and Validation


```
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,880 - __main__ - INFO - Agent Processing stopped
2025-10-14 01:19:02,880 - __main__ - INFO - Collection complete


================================================
COLLECTION SUMMARY
================================================
2025-10-14 01:19:02,867 - __main__ - INFO - Agent Discovery stopped
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,880 - __main__ - INFO - Agent Processing stopped
2025-10-14 01:19:02,880 - __main__ - INFO - Collection complete


================================================
2025-10-14 01:19:02,867 - __main__ - INFO - Agent Discovery stopped
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,880 - __main__ - INFO - Agent Processing stopped
2025-10-14 01:19:02,880 - __main__ - INFO - Collection complete
2025-10-14 01:19:02,867 - __main__ - INFO - Agent Discovery stopped
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,880 - __main__ - INFO - Agent Processing stopped
2025-10-14 01:19:02,867 - __main__ - INFO - Agent Discovery stopped
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,879 - __main__ - INFO - Processing complete. Processed 17 files
2025-10-14 01:19:02,880 - __main__ - INFO - Agent Processing stopped
2025-10-14 01:19:02,880 - __main__ - INFO - Collection complete


================================================
COLLECTION SUMMARY
================================================
Case ID: a7a9ea4a-e533-47ac-8656-b65f5eac14ea
Evidence Database: evidence_a7a9ea4a-e533-47ac-8656-b65f5eac14ea.db
Evidence Package: evidence_packages\evidence_package_a7a9ea4a-e533-47ac-8656-b65f5eac14ea_20251014_011902.zip
Collection completed at: 2025-10-14 01:19:02.882807
```

Figure 2: Console output showing successful processing of 17 files with complete agent lifecycle from discovery through packaging

Complete testing was important to make sure the system reliability for forensic applications where accuracy and auditability are top. I developed a test suite covering all major system components using python unit test framework. The test execution results showed twelve tests across five test classes with all tests passing successfully.

Testing the database manager involved verifying that tables were created correctly with suitable schemas and that evidence records could be inserted and retrieved accurately. These tests used temporary database files that were created for each test and deleted afterward, and make suring the test isolation. The tests confirmed that the database

initialization created both the evidence records table and the chain of custody table with correct column definitions.

The Processing Agent tests focused on the core forensic functionality. The SHA 256 calculation test created a file with known content, calculated its hash and compared the result against a expected value. This test also verified hash consistency by calculating the hash multiple times and ensuring identical results. The file type detection test created files with various extensions and verified that the system correctly identified their types. The metadata extraction test confirmed that the system captured all required metadata including timestamps, file size, and ownership information.

The Packaging Agent tests validated report generation and package creation. The CSV export test confirmed that evidence records were correctly formatted in comma-separated values with appropriate headers. The JSON export test verified that the data structure was valid JSON and contained all expected fields. The package creation test confirmed that the ZIP file was created, contained all necessary components, and that a chain of custody entry was recorded documenting the package creation.

dfas	10/14/2025 1:19 AM	Text Document	8 KB
evidence_a7a9ea4a-e533-47ac-8656-b65f5eac1...	10/14/2025 1:19 AM	Data Base File	28 KB
evidence_package_a7a9ea4a-e533-47ac-8656-...	10/14/2025 1:19 AM	Compressed (zipped)...	8 KB
evidence_report_a7a9ea4a-e533-47ac-8656-b6...	10/14/2025 1:19 AM	Microsoft Excel Com...	8 KB
evidence_report_a7a9ea4a-e533-47ac-8656-b6...	10/14/2025 1:19 AM	JSON Source File	13 KB

Figure 3: Evidence package contents including database, CSV/JSON reports, and encrypted ZIP package

The integration test validated the complete workflow from start to finish. This test created a temporary directory with multiple files, initialized all agents, ran the complete collection process, and verified that all files were discovered, processed, and packaged correctly. The test confirmed that the evidence database contained the expected number of records and that all generated files existed with non-zero sizes.

Learning Outcomes and Skill Development

This module transformed my understanding of intelligent systems from abstract theory to practical implementation. Several key learning outcomes stand out as particularly significant for my professional development.

My understanding of agent architectures deepened substantially through practical application. Initially, I could define reactive, deliberative, and hybrid architectures in theoretical terms, but implementing a BDI-inspired system revealed nuances and trade-

offs that theory alone cannot convey. I learned that the BDI model provides excellent structure for goal-oriented systems but requires careful design to avoid performance bottlenecks when belief sets grow large or when reasoning about complex goals.

Technical skills and problem solving abilities improved through hands-on development. My Python programming ability strengthened particularly in areas like concurrent programming with threads and queues, database interactions using SQLite, cryptographic operations with the cryptography library, and file system operations with pathlib. These are practical skills directly applicable to future projects in various domains beyond forensics.

Understanding of standards and compliance grew through research and implementation. Digital forensics operates under strict legal and regulatory frameworks. I learned about ISO 27037 guidelines for evidence handling, NIST SP 800-86 forensic techniques, and ACPO principles for digital evidence. Implementing a system that complies with these standards taught me the importance of documentation, audit trails, and cryptographic integrity verification in regulated domains.

Communication skills improved through collaborative discussions and presenting technical work. Explaining agent concepts to peers in discussion forums required translating technical jargon into clear explanations.These communication skills complement technical abilities and are essential for professional success.

Reflection on the Development Process

Looking back on the entire development process, I recognize both successes and areas where I could have performed better. Understanding these aspects helps me plan for improvement in future projects.

The phased development approach worked well. I started with the simplest component, the database manager, and validated it thoroughly before moving to more complex agents. Each agent was developed and tested independently before integration. This incremental approach allowed me to catch and fix issues early when they were easier to debug. If I had attempted to build everything simultaneously, tracking down bugs would have been far more difficult.

Documentation could have been more comprehensive throughout development. While the final code includes detailed comments and the README file provides good user documentation, I did not maintain detailed design documents during development. In several instances, I made architectural decisions that seemed obvious at the time but later required significant mental effort to remember the reasoning behind them. For future projects, I will maintain a design journal documenting major decisions and the rationale behind them.

Time management was generally effective but imperfect. I created a development timeline with milestones for each major component. This plan kept the project on track overall, but I underestimated the time required for testing and documentation. The actual coding took less time than expected while testing and documentation took more. In retrospect, I should have allocated more time for these crucial but less exciting aspects of software development.

The testing strategy proved its value repeatedly. Each time I added significant new functionality, running the test suite caught regression bugs where new code broke previously working features. Without automated testing, these bugs might have gone unnoticed until much later when they would have been harder to diagnose and fix. This experience convinced me of the importance of automated testing for any substantial software project.

Error handling improved throughout development as I encountered various failure modes. Early versions of the code assumed ideal conditions and crashed when facing unexpected situations like missing files or invalid permissions. As I tested more thoroughly and encountered these edge cases, I added defensive checks and graceful error handling. The final version rarely crashes, instead logging errors and continuing with remaining work. This robustness is essential for forensic tools that must work reliably even when examining damaged or suspicious systems.

Application to Real-World Contexts

The Digital Forensics Agent System addresses genuine needs in the digital forensics field. Currently, many forensic investigators manually collect evidence, which is slow, prone to human error, and difficult to audit. Automated collection tools exist but are often expensive commercial products or specialized tools that lack flexibility. DFAS demonstrates that intelligent agent principles can create flexible, auditable, and standards-compliant forensic collection systems.

The multi-agent architecture provides advantages over monolithic tools. Each agent specializes in a specific forensic task, making the system easier to understand, maintain, and extend. If new requirements emerge, such as integrating with specific forensic analysis tools or adding support for cloud storage evidence collection, developers can modify or extend individual agents without rewriting the entire system. This modularity is valuable for tools that must evolve to meet changing needs.

The configurable policy-based approach allows investigators to adapt the system to different investigation scenarios without modifying code. The YAML configuration file specifies what to collect, where to look, and what to exclude. An investigator can create multiple configuration profiles for different investigation types, such as fraud investigations

focused on financial documents or intellectual property theft investigations focused on source code and emails. This flexibility makes the tool more widely applicable than hard-coded collection tools.

The emphasis on standards compliance and audit trails addresses legal requirements for digital evidence. Courts require establishing chain of custody showing that evidence has not been tampered with from collection through presentation. The cryptographic hashing at multiple levels provides mathematical proof of integrity. The chain of custody table documents every action taken during collection. The encrypted packaging prevents unauthorized modification. These features make evidence collected with DFAS more likely to be admissible in legal proceedings.

Beyond digital forensics, the architectural patterns and implementation techniques from DFAS apply to other domains requiring automated data collection and processing. Environmental monitoring systems could use similar agent architectures where discovery agents find sensor data, processing agents validate and aggregate measurements, and packaging agents create reports for environmental agencies. Compliance auditing systems could use analogous designs where agents collect configuration data from IT systems, analyze compliance against policies, and generate audit reports. The fundamental pattern of specialized agents coordinating through message passing has broad applicability.

Future Improvements and Research Directions

While DFAS successfully demonstrates intelligent agent principles applied to digital forensics, numerous opportunities exist for enhancement and extension. Identifying these opportunities helps me understand the current system's limitations and plan future learning.

Advanced file analysis could integrate additional forensic techniques. Currently, the system calculates hashes and extracts basic metadata, but forensic investigations often require deeper analysis. Integrating YARA rule scanning would allow detecting known malware patterns or indicators of compromise. Entropy analysis could identify encrypted or compressed data that might contain hidden evidence. Optical character recognition could extract text from images and scanned documents. These capabilities would make the system more powerful for complex investigations.

Machine learning integration could increase the evidence prioritization. Not all collected files are equally relevant to an investigation. Training classifiers to score files based on relevance could help investigators focus on the most promising evidence first. For example, in a fraud investigation, financial documents would score higher than vacation

photos. Implementing this would require collecting training data and experimenting with various classification algorithms.

Conclusion

This Intelligent Agents module has been a transformative learning experience that connected theoretical concepts with practical application. The journey from understanding basic agent properties to implementing a functional multi-agent forensic system demonstrated both the power of intelligent agent principles and the challenges of real-world software development.

The Digital Forensics Agent System successfully applies BDI based architecture to solve genuine problems in evidence collection. The specialized agents working together through queue-based communication create a system that is performant, maintainable, and extensible. The emphasis on standards compliance, cryptographic integrity, and audit trails makes the tool potentially valuable for actual forensic work while demonstrating key concepts from the module.

The most important lesson from this experience is that intelligent agent principles are not merely academic abstractions but powerful tools for building real systems that solve complex problems. By carefully choosing appropriate architectures, designing clean communication protocols, handling errors gracefully, and maintaining focus on user needs, we can create intelligent systems that are both theoretically sound and practically useful. This insight will guide my future work in technology, whatever specific form that work takes.

References

1. Association of Chief Police Officers (2012) *ACPO Good Practice Guide for Digital Evidence*. Available at: https://www.digital-detective.net/digital-forensics-documents/ACPO_Good_Practice_Guide_for_Digital_Evidence_v5.pdf (Accessed: 14 October 2025).
2. Bratman, M.E. (1987) *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press.
3. International Organization for Standardization (2012) *ISO/IEC 27037:2012 Information technology — Security techniques — Guidelines for identification, collection, acquisition and preservation of digital evidence*. Geneva: ISO.
4. International Organization for Standardization (2015) *ISO/IEC 27042:2015 Information technology — Security techniques — Guidelines for the analysis and interpretation of digital evidence*. Geneva: ISO.
5. National Institute of Standards and Technology (2015) *FIPS PUB 180-4: Secure Hash Standard (SHS)*. Gaithersburg, MD: U.S. Department of Commerce. Available at:

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (Accessed: 14 October 2025).

6. Polleres, A. (2007) 'From SPARQL to rules (and back)', in Williamson, C.L., Zurko, M.E., Patel-Schneider, P.F. and Shenoy, P.J. (eds.) *Proceedings of the 16th International Conference on World Wide Web*. New York: ACM, pp. 787-796.

7. Searle, J.R. (1969) *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press.

8. The University of Edinburgh (no date) *Reflection Toolkit*. Available at: https://www.ed.ac.uk/reflection/reflectors-toolkit (Accessed: 14 October 2025).