Development Individual Project

Student ID: 12696139

Transcript File

Introduction (Slide 1)

Hello, I am **Murthy Kanuri**.

This presentation covers the practical build of my Unit 6 design of an Intelligent Forensics Agent. I will walk through the core problem and objectives, the architecture, a live demonstration of the implemented components, and the test results. The core of this submission is Agent 1, the File Locator Agent, which we fully implemented to manage file system traversal, reliable type identification, and metadata extraction. Agent 2, the learning and archiving component, is scoped as planned future work, and I will touch on it briefly at the end.

Problems & Objectives (Slide 2)

The problems addressed are crucial in digital forensics. Manual triage is slow and often fails, and relying on file extensions might also be misleading. Cross-platform quirks and protected directories lead to errors. To protect privacy and security, we must capture data minimally to ensure every action is auditable (ICO,2024).

The design's multifaceted objectives address these challenges:

- First, safe discovery is achieved via platform-aware exclusions, with all operations restricted to read-only sources (Rodolà, 2025).
- Second, the central objective is to identify by content first using Python-Magic, with file type and extension checks as fallbacks (Hupp, 2022; Aparicio, 2022).
- Third, essential metadata only (like SHA-256, size, and timestamps) is captured and stored in a portable SQLite database, aligning with data minimisation (PSF, 2025c; SQLite Consortium, 2025).

Finally, while the primary focus is on Agent 1, the architecture lays the groundwork for automating Agent 2. The entire process is CLI-driven to ensure reproducible runs.

Architecture (Slide 3)

The system is strong because of its modular and adaptive design, based on a Hybrid Reactive Pipeline. This model ensures predictable actions while remaining responsive to the environment.

The core process flows through five distinct stages: Discover, Identify, Decide, Act, and Audit.

- First, Agent 1 performs the initial phases: Discovery (Traversal), Identification, and Decision. If the decision is yes, it extracts and stores metadata in SQLite. An Audit concludes the process via a CLI summary.
- Second, Agent 2, the File Archiver, is the long-term automation component. It uses machine learning to learn from human feedback and performs predictive archiving.

Separating these responsibilities lets Agent 1 focus entirely on evidential integrity, and Agent 2's predictive power evolves independently.

Components & Code Structure (Slide 4)

Slide 4 shows the actual code structure and components. This structure directly results from applying both Agent-Oriented Programming (AOP) and Object-Oriented Principles (OOP), ensuring the code is modular, maintainable, and highly organised.

The main source directory, src/, contains all the distinct modules:

1. **discovery.py:** Handles the Discovery. It uses psutil to perform safe traversal and implement the platform-aware exclusions we discussed earlier (Rodolà, 2025). This modular approach was chosen to clearly separate traversal logic from processing.
2. **identifier.py:** Manages the Identify phase. Its core justification is reliability: it prioritises content-based MIME-type analysis using Python magic before resorting to filetype and extension checks (Hupp, 2022; Aparicio, 2022).
3. **processor.py:** Manages the Act phase. It minimises data by recording SHA-256, size, and timestamps to meet ethical and evidential integrity standards (PSF, 2025a).
4. **database.py** implements the SQLite persistence layer. The design used upsert logic with indexes to ensure idempotent runs, meaning the agent can safely rerun without duplicating existing entries (PSF, 2025c; SQLite Consortium, 2025).

The dedicated tests/directory also clearly shows my commitment to quality. I created specific unit tests for all critical modules covered in the Demo & Test Evidence slides.

The entire process is orchestrated by main.py (CLI). learner.py is the architectural placeholder for Agent 2.

Key Libraries & Justification (Slide 5)

This slide focuses on the key libraries and the justifications behind their selection.

- As mentioned previously, `psutil` is used for environment-aware discovery. It is explicitly implemented to enable safe exclusions during traversal, directly mitigating the critical risk of system instability (Rodolà, 2025).
- `Python-Magic` was used for identification since it employs content-based MIME detection, which is more accurate than just looking at the file's extension.
- File type was included as a lightweight backup. If both main detectors fail, the Python standard library's `mimetypes` are utilised as a last resort (Hupp, 2022; Aparicio, 2022; PSF, 2025b).
- `SQLite` was used for data handling. The setup was configured for portable, single-file storage and included upsert logic with indexes. This ensures runs are idempotent and auditable (PSF, 2025c; SQLite Consortium, 2025).
- `hashlib` was also used to get a solid SHA-256 hash of each file, which is important proof of integrity (PSF, 2025a).

Finally, for Agent 2's future work, scikit-learn and `joblib` are referenced (Pedregosa et al., 2011; Joblib Developers, 2025). These choices collectively form a robust, reliable, and justifiable architecture.

Demo (Slide 6)

We now move to the live demonstration of Agent 1. This slide shows the output capture and run summary, which visually complements the video snippet.
The run is initiated using the CLI, which is the required mechanism for reproducible runs.

- Let us look at the execution's success. The summary confirms five processed files and lists the Counts by MIME Type. The chart on the right visualises these counts, demonstrating the successful identification of the pipeline. This confirms that the `identifier.py` module, leveraging `Python-magic,` correctly classified all files by their content, meeting our objective of using content-based identification.

- The Sample Record confirms the integrity of the data capture. The agent successfully extracted the path, mtime (timestamp), SHA-256 (for evidence integrity via `hashlib`), and the size and mime type. The inclusion of only these fields verifies our commitment to data minimisation. We can see the detector used was explicitly `python-magic` for this file.
- Although not explicitly captured in this screenshot, the run successfully tested the PsutilFilter. Before scanning the sample data, the filter checked and automatically excluded critical directories on my operating system, proving our safe discovery and system stability safeguards are actively implemented.
- Finally, the collected data is stored in the SQLite database. This run confirms that the Discover, Identify, and Act phases of our Agent 1 pipeline function correctly and as expected.

<u>Test Evidence (Slide 7)</u>

Next, I needed reliable testing evidence. I used Python's unit test module and the `coverage.py` tool to measure the code's correctness and quality.

My initial 3-unit tests passed successfully, confirming Agent 1's core functionality. These tests verified the `database.py` upsert logic (for idempotent runs), the identifier.py MIME-type detection, and the `processor.py` extraction and hashing logic.

The coverage report confirms that 36% overall line coverage was achieved across the key modules. While this provides a strong foundation, the report clearly highlights areas that require improvement, such as the missing lines in `database.py` and `identifier.py.`

This transparent output leads directly to our documented remediation activity. I know exactly what I need to do next: to have more coverage, I need to write explicit tests for the fallback logic in file identification and improve how I handle upsert conflicts. This iterative process is based on agile methodology, where test results lead development.

In summary, this slide demonstrates the code's correctness (via passing unit tests) and my commitment to using testing tools to improve code quality continuously.

Learning Component (Slide 8)

We now pivot to Agent 2, the Learning Component. Although this is future work, I have successfully built the necessary foundation. This component shows how I apply my understanding of intelligent agent systems.

The goal of this small 'Toy Classifier' is to flag likely non-text files for archiving, providing guidance only to the user. The model is ethically compliant: it uses only metadata features extracted by Agent 1 and does not read or store file contents.

I chose a small logistic regression model from scikit-learn. The `class_weight='balanced'` setting mitigated the risk of training on imbalanced data (Pedregosa et al., 2011).

The output confirms two key implementation points:

- First, Model Persistence: I saved the trained model using `Joblib`, ensuring it can be reloaded for reproducible recommendations.
- Second, the output format clearly summarises predictions in the 'Predicted Class Counts' chart**.**

In line with my critical evaluation, this model's scope is advisory only; it is explicitly not an autonomous decision-making system. This manages the ethical risk of misclassification, ensuring human oversight remains necessary before any archiving action is taken.

Risks, Ethics & Mitigations (Slide 9)

This final content slide covers my agent's critical evaluation, detailing the significant risks, ethical concerns, and mitigations implemented in my code. This is where I justify why I made specific design choices based on sound academic principles.

- **Operational Mitigations** (Rodolà, 2025; SQLite Consortium, 2025)**:** On the practical side, I do not trust extensions. I check the bytes with `python-magic`, fall back to `filetype`, and only then use the extension as a last resort. I used `psutil` to exclude system paths and do a smoke check before scanning. Writing is limited to a single process to reduce corruption risk, and `SQLite` runs in WAL mode.

- **Ethical and Compliance Mitigations** (ICO, 2024; OWASP Foundation, 2023)**:** The ethical and compliance area is critical. I use data minimisation: path, size, timestamps, MIME and SHA-256; nothing else is stored. The agent runs locally and read-only; it never inspects file contents. To address path and filename sensitivity, I implement code to redact sensitive paths in logs and only share aggregates. The learner is clearly labelled as a demo/guidance tool, ensuring it is not a decision system.
- **Audit and Traceability** (SQLite Consortium, 2025)**:** To ensure audit gaps are closed, I included dry-run mode. Every run prints a CLI summary with a timestamp and hash to provide non-repudiation. Between the database, the logs, and the saved model files, you get a clean, reproducible trail. These layered mitigations demonstrate that the system is functional, reliable, ethical, and forensically sound.

<u>Conclusions & Code (Slide 10)</u>

This final slide summarises the achievements and addresses the Learning Outcomes.

- Agent 1, the File Locator, was successfully developed, deployed, and tested. The core pipeline is fully implemented: It performs safe discovery, executes content-first identification, and stores metadata only in the `SQLite` upsert database.
- It is complemented by the clear evidence of execution and testing, including the demo run, the MIME chart, 3/3 passed unit tests, and the 36% code coverage report.
- All generated artefacts are submitted alongside the code. Overall, the design fits Unit 6 principles of automation, robustness, data minimisation, and reproducibility.

I acknowledge the current limits: Agent 2 is a 'Toy Recommender', and my code coverage is moderate. My next steps are clearly defined: full implementation of Agent 2, including active learning, higher test coverage, and building a purge command for retention policies.

The system, including the technical justifications presented on Slide 9, is based on sound academic principles.

The README file contains detailed execution instructions and confirms all sources, repositories, and libraries are fully cited in the required University of Essex Online (UoEO) format.

References

- **Aparicio, T. (2022)** *filetype: infer file type and MIME*. Available at: https://github.com/h2non/filetype (Accessed: 11 October 2025).
- **Batchelder, N. (2025)** *coverage.py — Code coverage for Python*. Available at: https://coverage.readthedocs.io/ (Accessed: 11 October 2025).
- **Dubettier, A., et al. (2023)** ‘File type identification tools for digital investigations’, Forensic Science International: Digital Investigation, 46, p. 301574. doi:10.1016/j.fsidi.2023.301574.
- **Hupp, A. (2022)** *python-magic documentation*. Available at: https://github.com/ahupp/python-magic (Accessed: 11 October 2025).
- **Information Commissioner’s Office (ICO) (2024)** ‘UK GDPR: Principle (c) Data minimisation’. Available at: https://ico.org.uk/ (Accessed: 11 October 2025).
- **Joblib Developers (2025)** *Joblib documentation*. Available at: https://joblib.readthedocs.io/ (Accessed: 11 October 2025).
- m-kanuri (2025) *ia_agent*. GitHub repository. Available at: https://github.com/m-kanuri/ia_agent (Accessed: 12 October 2025).
- **Kanuri, M., Espag, J., Kirwan, F. and Jittipattanakulchai, G. (2025)** *Intelligent Forensics Agent – Group E Design Report (Unit 6)*. University of Essex Online (unpublished coursework).
- **Kessler, G.C. (2024)** *File Signature Table*. Available at: https://www.garykessler.net/library/file_sigs.html (Accessed: 11 October 2025).
- **MITRE (2024)** *MITRE ATT&CK®: Masquerading (T1036)*. Available at: https://attack.mitre.org/techniques/T1036/ (Accessed: 11 October 2025).
- **National Institute of Standards and Technology (NIST) (2020)** *Security and Privacy Controls for Information Systems and Organizations (SP 800-53 Rev. 5)*. Gaithersburg, MD: NIST.
- **OWASP Foundation (2023)** *Logging Cheat Sheet*. Available at: https://cheatsheetseries.owasp.org/ (Accessed: 11 October 2025).
- **Pedregosa, F., Varoquaux, G., Gramfort, A. et al. (2011)** ‘Scikit-learn: Machine Learning in Python’, *Journal of Machine Learning Research*, 12, pp. 2825–2830.

- **PyPI (2024)** *python-magic-bin (Windows prebuilt)*. Available at: https://pypi.org/project/python-magic-bin/ (Accessed: 11 October 2025).
- **Python Software Foundation (PSF) (2025a)** *hashlib — Secure hashes and message digests*. Available at: https://docs.python.org/3/library/hashlib.html (Accessed: 11 October 2025).
- **Python Software Foundation (PSF) (2025b)** *mimetypes — Map filenames to MIME types*. Available at: https://docs.python.org/3/library/mimetypes.html (Accessed: 11 October 2025).
- **Python Software Foundation (PSF) (2025c)** *sqlite3 — DB-API 2.0 interface*. Available at: https://docs.python.org/3/library/sqlite3.html (Accessed: 11 October 2025).
- **Python Software Foundation (PSF) (2025d)** *unittest — Unit testing framework*. Available at: https://docs.python.org/3/library/unittest.html (Accessed: 11 October 2025).
- **Rodolà, G. (2025)** *psutil documentation*. Available at: https://psutil.readthedocs.io/ (Accessed: 11 October 2025).
- **scikit-learn Developers (2025)** *scikit-learn User Guide*. Available at: https://scikit-learn.org/stable/ (Accessed: 11 October 2025).
- **SQLite Consortium (2025)** *SQLite documentation*. Available at: https://sqlite.org/docs.html (Accessed: 11 October 2025).
- **Shoham, Y. (1993)** ‘Agent-oriented programming’, *Artificial Intelligence*, 60(1), pp. 51–92.