



Code Is the Source of Truth: A Static Analysis Tool for Data Flow Documentation of Python Data Pipelines

Omar Mohammed Qassem — 2316700

Driver and Vehicle Licensing Agency (DVLA)

Research context

Public sector bodies process enormous volumes of data

Data Protection Act 2018 - UK GDPR accountability principle

- Orgs. Must demonstrate what happens to personal data as it moves through their systems

Currently written manually

Guiding principle: **The code is the source of truth**

- Documentation generated from code stays accurate as the code changes

Two audiences:

- Data Engineers
- Data Protection Officers/Auditors

Research Problem

“No existing tool generates compliance-grade, code-level data-flow documentation for Python data pipelines.”

- Manual documentation becomes inaccurate as pipelines evolve
- Existing tools don't fill the gap:
 - SQLGlott — SQL only, not Python.
 - Databricks Unity Catalog — sees tables, not the Python logic in between.
 - Security scanners (Bandit, Semgrep, CodeQL, Pysa) — produce warnings, not a map of where data goes.
- The gap: a trustworthy map of where data goes, straight from Python code.

Research Aim & Objectives

Aim

- Design, develop and evaluate a static-analysis tool that parses Python (Pandas & PySpark), maps its data flow, identifies sources and sinks, and presents data-flow documentation via an interactive dashboard, supporting compliance and visibility.

Objectives

- Critically analyse existing static/taint analysis and data-lineage techniques
- Design an architecture that produces a data-flow graph, with a source/sink catalogue and route tracing.
- Develop an MVP tracing data flows source-to-sink, across functions and modules, presented via a dashboard.
- Evaluate detection accuracy and performance against a benchmark of pipelines with known data flows.
- Stretch: track sanitisation steps (masking, hashing, encryption) between sources and sinks.

Literature Review

Foundations well established:

- Data-flow analysis - tracking how data moves through a program (Kildall, 1973).
- AST-based analysis - parse code into an Abstract Syntax Tree, map paths, trace variable origins (Aho et al.).
- IFDS framework - interprocedural analysis - tracing routes between functions (Reps, Horwitz & Sagiv).

Flow security & taint analysis:

- Denning's secure information-flow model.
- Taint analysis follows data from sources to sinks, tracking any sanitiser in between (Schwartz et al.).
 - Very similar to proposed tool



Literature Review

Applying this to Python is hard:

- PoTo (points-to) and PyCG (call-graph) help follow data across function calls.
- Industry tools: CodeQL, Bandit, Semgrep, Meta Pysa, PyT, PyFlow.
- These produce warnings, not documentation of where data goes.

Data lineage (the field related to the aim):

- Provenance theory (Cheney et al.), fine-grained data-science provenance (Chapman et al.), a GDPR provenance model (Ujcich et al.).
- Unity Catalo, SQLGlott

The gap: no tool produces compliance-ready, code-level data-flow documentation for Python pipelines.

Research/Development Methodologies

Research strategy (Saunders' "research onion"):

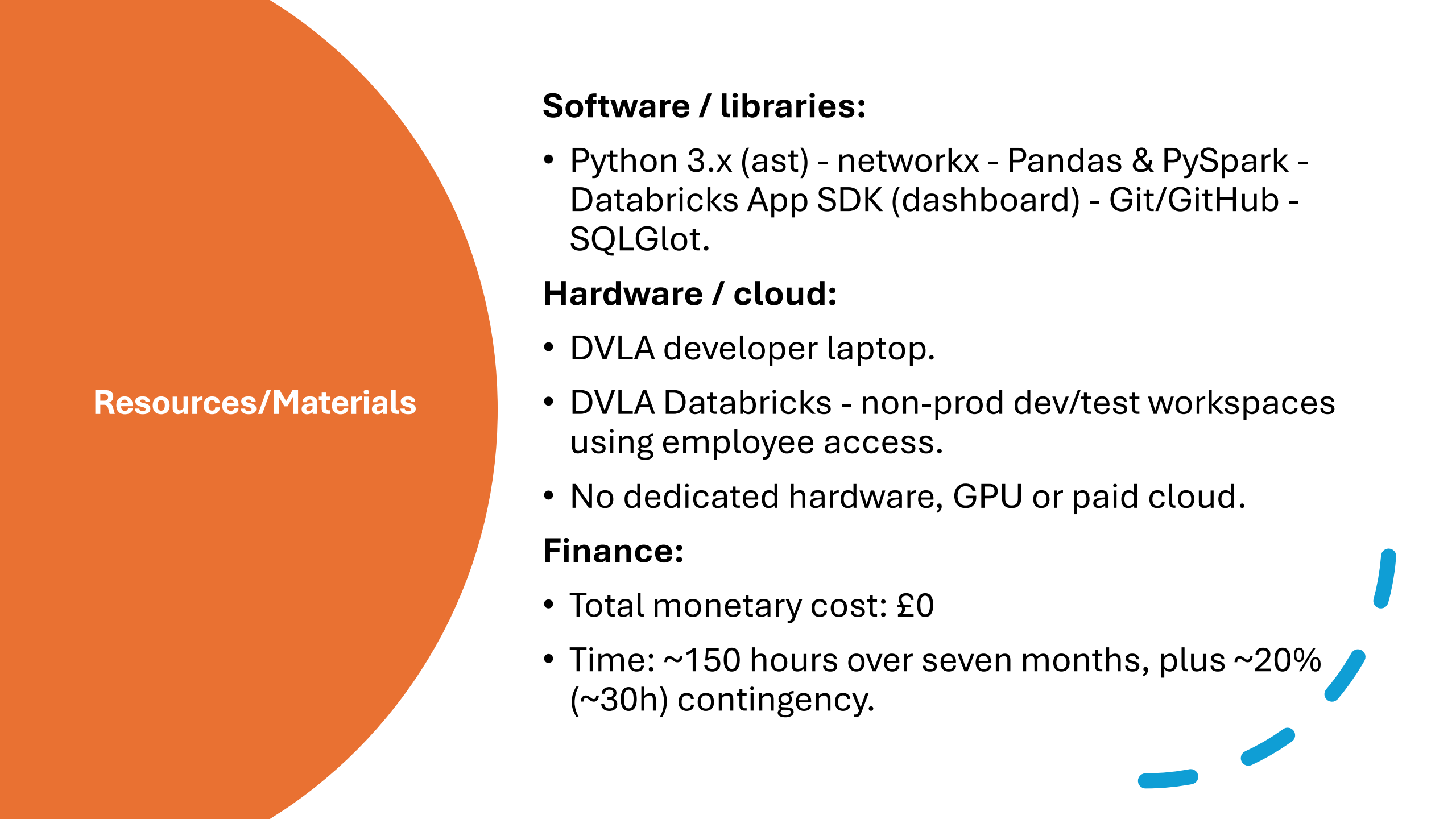
- Software Development strategy - main output is working, evaluated software.
- Positivist / deductive - build on previous techniques and measure objectively.
- Quantitative & cross-sectional - numeric results, evaluated at a point in time.

How it processes code:

- Parse Python > build an AST (built-in ast module).
- Walk AST to find variables, calls, imports, I/O & build a data flow graph with networkx.
- Match against a configurable source/sink catalogue (Pandas & PySpark).
- Trace routes between sources & sinks, within and between functions.

Data & success criteria:

- Real DVLA code & bespoke test pipelines, locked down dev/test, using test data. Correct answers recorded in advance.
- Measured by precision, recall, F1 & timing.



Resources/Materials

Software / libraries:

- Python 3.x (ast) - networkx - Pandas & PySpark - Databricks App SDK (dashboard) - Git/GitHub - SQLGlott.

Hardware / cloud:

- DVLA developer laptop.
- DVLA Databricks - non-prod dev/test workspaces using employee access.
- No dedicated hardware, GPU or paid cloud.

Finance:

- Total monetary cost: £0
- Time: ~150 hours over seven months, plus ~20% (~30h) contingency.

Project Plan

Timeframe: Jan 2027 > July 2027 (~7 months, ~150 hours, part-time).

Scope (MoSCoW):

- **Must** (MVP): AST parsing, source/sink catalogue, route tracing within & between functions, basic dashboard, benchmarking & evaluation.
- **Should**: fuller Pandas/PySpark coverage, better visualisations, fuller benchmark.
- **Could**: detail protection steps (masking/hashing) before a sink.
- **Won't**: full Python support, dynamic analysis.
- **Stretch**: Track sanitisation steps

Timeline & milestones:

- **Jan to Feb**: setup & literature review.
- **Feb**: requirements & design.
- **Mar to Apr**: build MVP (all Musts & hopefully Shoulds).
- **Apr to May**: benchmark & evaluation.
- **June**: stretch features (if time allows).
- **July**: write-up & submission of report



Impact

Automated compliance evidence

Visibility for engineers

Risk-catching

Transferable beyond DVLA

Limitations



Python is dynamic

Imports/variables/datatypes etc. that are decided at runtime



Interprocedural analysis is hard

Time limitations



Benchmark dependence

Test and DVLA-specific pipelines only

References

- - [1] G. A. Kildall, "A Unified Approach to Global Program Optimization," POPL, 1973, pp. 194–206. doi:10.1145/512927.512945.
- - [2] D. E. Denning, "A Lattice Model of Secure Information Flow," ACM, 19(5):236–243, 1976. doi:10.1145/360051.360056.
- - [3] D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow," ACM, 20(7):504–513, 1977. doi:10.1145/359636.359712.
- - [4] T. Reps, S. Horwitz, and M. Sagiv, "Precise Interprocedural Dataflow Analysis via Graph Reachability", 1995, pp. 49–61. doi:10.1145/199448.199462.
- - [5] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, "Compilers: Principles, Techniques, and Tools", 2nd ed. Pearson/Addison-Wesley, 2006. ISBN 978-0321486813.
- - [6] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution...", IEEE S&P, 2010, pp. 317–331.
- - [7] A. Sabelfeld and A. C. Myers, "Language-Based Information-Flow Security," IEEE J. Sel. Areas Commun., 21(1):5–19, 2003. DOI 10.1109/JSAC.2002.806121.
- - [8] T. Mao et al., "SQLGlot," GitHub. <https://github.com/tobymao/sqlglot>
- - [9] I. Rak-amnouykit, A. Milanova, G. Baudart, M. Hirzel, and J. Dolby, "PoTo: A Hybrid Andersen's Points-To Analysis for Python," ECOOP, LIPIcs 333, pp. 27:1–27:29, 2025. doi: 10.48550/arXiv.2409.03918.
- - [10] V. Salis, T. Sotiropoulos, P. Louridas, D. Spinellis, and D. Mitropoulos, "PyCG: Practical Call Graph Generation in Python," ICSE, 2021, pp. 1646–1657. doi:10.1109/ICSE43902.2021.00146.
- - [11] P. Avgustinov, O. de Moor, M. Peyton Jones, and M. Schäfer, "QL: Object-oriented Queries on Relational Data," ECOOP, LIPIcs 56, pp. 2:1–2:25, 2016. doi:10.4230/LIPIcs.ECOOP.2016.2.
- - [12] UK Government, Data Protection Act 2018, c. 12. <https://www.legislation.gov.uk/ukpga/2018/12>
- - [13] PyCQA, "Bandit," GitHub. <https://github.com/PyCQA/bandit>
- - [14] Semgrep, "Taint Mode — Overview," docs. <https://semgrep.dev/docs/writing-rules/data-flow/taint-mode/overview>
- - [15] Meta/Pyre, "Pysa," docs. <https://pyre-check.org/docs/pysa-basics/>
- - [16] S. Micheelsen and B. Thalmann, "PyT — A Static Analysis Tool...", MSc thesis, Aalborg Univ., 2016. <https://projekter.aau.dk/projekter/files/239563289/final.pdf>
- - [17] Z. Gu, H. Yan, and P. Yao, "PyFlow: An Inter-procedural Static Analysis Framework for Python," arXiv:2608.07026, 2026. <https://arxiv.org/abs/2608.07026>
- - [18] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in Databases: Why, How, and Where," Found. Trends Databases, 1(4):379–474, 2009. doi:10.1561/19000000006.
- - [19] A. Chapman et al., "Capturing and Querying Fine-grained Provenance of Preprocessing Pipelines in Data Science," PVLDB, 14(4):507–520, 2021.
- - [20] B. E. Ujcich, A. Bates, and W. H. Sanders, "A Provenance Model for the EU GDPR," IPAW, LNCS 11017, pp. 45–57, 2018. doi:10.1007/978-3-319-98379-0_4.
- - [21] Databricks, "Data Lineage in Unity Catalog," docs. <https://docs.databricks.com/aws/en/data-governance/unity-catalog/data-lineage.html>