Flow with FlorDB: Incremental Context Maintenance for the Machine Learning Lifecycle

Rolando Garcia
UC Berkeley
United States of America
rogarcia@berkeley.edu

Pragya Kallanagoudar
UC Berkeley
United States of America
pkallanagoudar@berkeley.edu

Chithra Anand
UC Berkeley
United States of America
chithra.rajan@berkeley.edu

Sarah E. Chasins
UC Berkeley
United States of America
schasins@berkeley.edu

Joseph M. Hellerstein
UC Berkeley
United States of America
hellerstein@berkeley.edu

Erin Michelle Turner Kerrison
UC Berkeley
United States of America
kerrison@berkeley.edu

Aditya G. Parameswaran
UC Berkeley
United States of America
adityagp@berkeley.edu

ABSTRACT

In this paper we present techniques to incrementally harvest and query arbitrary metadata from machine learning pipelines, without disrupting agile practices. We center our approach on the developer-favored technique for generating metadata — log statements — leveraging the fact that logging creates context. We show how hindsight logging [8] allows such statements to be added and executed post-hoc, without requiring developer foresight. Relational views of incomplete metadata can be queried to dynamically materialize new metadata in bulk and on demand across multiple versions of workflows. This is done in a "metadata later" style, off the critical path of agile development. We realize these ideas in a system called FlorDB and demonstrate how the data context framework covers a range of both ad-hoc metadata as well as special cases treated today by bespoke feature stores and model repositories. Through a usage scenario—including both ML and human feedback—we illustrate how the component techniques come together to resolve classic software engineering trade-offs between agility and discipline.

1 INTRODUCTION

In the rapidly evolving field of Artificial Intelligence (AI) and Machine Learning (ML), the management of metadata has emerged as an enduring challenge [9, 19]. As machine learning models and their applications become increasingly integral to business and technology, the need for accurate and comprehensive metadata management has never been more critical. However, this requirement presents numerous difficulties, primarily due to the highly diverse mix of systems and artifacts involved in MLOps. Compounding this complexity is the necessity for robust feedback loops that span organizational boundaries—such as those between design and deployment teams—and extend over time.

 15th Annual Conference on Innovative Data Systems Research (CIDR '25). January 19-22, Amsterdam, The Netherlands

1.1 A Crisis of Metadata Management

One persistent challenge is balancing agility with the rigor of up-front documentation or "metadata first" approaches. Detailed documentation and metadata are essential for reproducibility, debugging, and collaboration. Unfortunately, insisting on comprehensive metadata from the start can hinder the agility and speed that are often crucial in early development stages.

The tradeoff between discipline and agility is an enduring open problem, evident in various contexts including the dichotomy between schema-first and NoSQL databases, and the contrast between data warehouses and data lakes. Schema-first and warehouse approaches offer discipline and structure but lack the flexibility needed for rapidly evolving requirements. Conversely, NoSQL and data lake approaches provide the necessary agility but can lead to inconsistency and disarray in metadata management.

To address this, two core goals can be established for resolving what has been considered an inherent conflict:

(1) **Goal 1. Agile Development Loop:** Metadata capture should be gradually incorporated into the agile workflows of data scientists and MLEs without interference. For data scientists, metadata capture should fit naturally into their open-source development environment. For MLEs, metadata should fit into standard tools for workflow management and scalability, such as relational databases and CI/CD pipelines, without requiring lock-in on yet another service for metadata management.
(2) **Goal 2. Metadata on Demand:** Metadata generation, like other features, should be able to evolve incrementally, improving organically with project needs. Ideally we want a "metadata later" approach that easily back-propagates metadata discipline into earlier versions of the project — at just the time when someone regrets not having the metadata they need.

1.2 Goals and Contributions

Our work is built on FlorDB[1], the scope of which we significantly expanded to encompass the entire ML lifecycle. FlorDB now captures complete pipelines and their regular execution, rather than just individual tasks. These extensions apply to mechanisms managed by the FlorDB API, initially designed for multiversion hindsight logging but now adapted to include incremental context maintenance over workflows. Despite these extensions, FlorDB maintains API stability and backward compatibility with multiversion hindsight logging.

Key goals and contributions include:

(1) **Incremental Context Maintenance for Agile DevOps:** Our system allows developers to log and analyze metadata in a standard, open, low-friction manner. Metadata can be captured naturally through Python `log` statements as part of the development workflow, without imposing significant overhead. Subsequently, these log statements can be read directly as tabular data using standard Python `dataframes`, queried via Pandas or SQL, without requiring data wrangling.

(2) **Ad-Hoc Metadata on Demand Enabled by Hindsight Logging:** FlorDB enables agile metadata evolution through multiversion hindsight logging, which operates as a record-replay mechanism. While traditional logging systems require predefined schemas, our approach captures sufficient execution state during recording, and allows the extraction of *arbitrary expression values* derivable from that state during replay. This means metadata collection isn't limited to just what was explicitly recorded; developers can compute arbitrary new properties and relationships from the execution record after the fact. Like materializing a view, developers can declare what metadata they want to extract, but unlike views, the source material extends beyond stored data to encompass both the complete execution record and the universe of derivable properties. This powerful mechanism frees developers from "metadata first" constraints while enabling unbounded possibilities for post-hoc metadata computation and analysis.

(3) **Unified, Open Metadata for Machine Learning:** Last, the open, standard approach of FlorDB simplifies and improves the abstraction of metadata, incorporating features from various bespoke ML metadata systems such as feature stores, model registries, and labeling systems into a unified and robust framework.

We demonstrate these contributions through a document intelligence use-case, highlighting the importance of context in streamlining development cycles and improving operational efficiency.

2 MULTIVERSION HINDSIGHT LOGGING

This section provides background on Flor and FlorDB, highlighting their evolution and key features in managing the machine learning lifecycle. Flor, originally designed as a record-replay system for model training [8], offers two main features: i) low-overhead adaptive checkpointing, minimizing computational resources during model training, and ii) low-latency replay from checkpoints, leveraging memoization and parallelism speed ups. Flor's record-replay mechanism introduced the notion of hindsight logging, allowing for post-hoc and on-demand querying of effectively unbounded *context*.

We adopt the term "context" from the Ground project [12], as a signifier for an all-encompassing view of metadata that goes beyond traditional relational metadata. Here, "context" includes anything that could be emitted by a log statement in any running process, be it an ML training job, a data wrangling script, a live inference server, a log processor handling usage feedback, or even an orchestration framework that knits these other tools together.

FlorDB extends Flor's capabilities by integrating automatic version control, adding a relational data model for querying logs, and cross-version logging statement propagation for multiversion hindsight logging [7]. Its relational model, accessible via `flor.dataframe`, maps individual logging statements into columns in a pivoted view. This approach facilitates easy tracking of changes over time. In this paper (Section 2.1), we further extend FlorDB to support dataflow pipelines and manage feedback loops. This extension provides a seamless framework for capturing arbitrary context introduced in defining and executing complex, evolving ML pipelines.

To clarify the concept of multiversion hindsight logging, imagine a scenario where a developer has run several versions of a machine learning pipeline but later realizes that certain metadata or context was not captured during those runs. Traditionally, retrieving this missing information would require re-running each version of the pipeline with the new logging statements inserted—a process that is both time-consuming and resource-intensive. FlorDB's multiversion hindsight logging offers a powerful alternative that minimizes both developer effort and compute resources. Developers can add the desired logging statements to the latest version of their code, and FlorDB will (a) inject these statements into the correct locations in all prior versions of the code, and (b) retroactively execute these statements across all those versions via incremental replay, without the need for full re-execution. The former, (a), is made possible via techniques adapted from code diffing [6]; the latter, (b), is made possible through a combination of differential execution and parallelism, allowing FlorDB to efficiently replay only the necessary parts of the pipeline to extract the new metadata. This "magic trick" enables developers to incrementally build up context and metadata after the fact, supporting agile practices by eliminating the need for foresight in logging. It resolves the classic trade-off between starting fast and refining over time by providing the flexibility to log now, and get data from the past. For technical details on how this is achieved, we refer readers to our prior work [7].

2.1 FlorDB Extended API

FlorDB's API captures metadata about the executing file, eliminating the need to restate dataflow dependencies; cross-executable dependencies declared in a typical workflow orchestration tool (Airflow, MLFlow, Make, etc) suffice. By profiling runtime metadata, including the executed file's name, FlorDB remains agnostic to the choice of workflow management system, functioning seamlessly without requiring refactoring of orchestration scripts.

The Flor API, as presented in Garcia et al. (2023) [7], includes:

[1] https://github.com/ucbrise/flor

- `flor.log(name: str, value: T) -> T`: Logs a value with a specified name, constructing a record with `projid`, `tstamp`, `filename`, and nesting dimensions defined by `flor.loop`.
- `flor.arg(name: str, default: T) -> T`: Reads command-line values or uses defaults, retrieving historical values during replay.
- `flor.loop(name:str, vals:Iterable[T]) -> Iterable[T]`: A Python generator maintaining global state between iterations, useful for addressing `flor.log` records and coordinating checkpoints.
- `flor.checkpointing(kwargs: Dict) -> ContextManager`: A Python "context manager" defining objects for adaptive checkpointing at `flor.loop` iteration boundaries.
- `flor.dataframe(*args) -> pd.DataFrame`: Produces a Pandas DataFrame of log information, with columns corresponding to each argument in `*args` plus dimension columns like `projid`, `tstamp`, `filename`.

To support user interactions in long-running web applications, such as "Save & Close" buttons, FlorDB is further extended with the following API call:

- `flor.commit() -> None`: An application-level transaction commit marker supporting visibility control for long-running processes. It writes a log file, commits changes to git, and increments the `tstamp`. This method is automatically invoked (via `atexit`) at the end of a Python execution.

3 INCREMENTAL CONTEXT MAINTENANCE

The ML lifecycle is characterized by numerous fast-changing components, where it is easy to lose track of essential metadata — what we term *context*. Context represents a comprehensive framework that captures the nature, origins, evolution, and functional significance of data and digital artifacts within an organization. It is metadata broadly conceived, extending beyond traditional database metadata to encompass the full spectrum of information necessary for understanding and managing projects.

We base our conceptualization of context on the "ABCs of Context" framework introduced by Hellerstein et al. (2017) [12], which extends traditional database metadata to encompass a broader spectrum of information critical in ML applications. The framework includes:

(1) **Application Context** (the "A"): Captures how raw data is interpreted, including schemas, checkpoints, and parameters. FlorDB captures application context through log statements, allowing developers to record pertinent information during execution.
(2) **Behavioral Context** (the "B"): Tracks how data is created and used over time, relating to lineage or provenance. This context is often contained in build files, capturing dependencies and directed acyclic graphs (DAGs) of tasks. FlorDB captures behavioral context via its relational data model, associating each logging statement with its originating filename. This mapping reveals dataflow pathways through the codebase, enabling analysis of data transformations and lineage within the pipeline.
(3) **Change Context** (the "C"): Manages version histories of both data and code. FlorDB manages change context using Git version control, ensuring that all runs and modifications are tracked and retrievable.

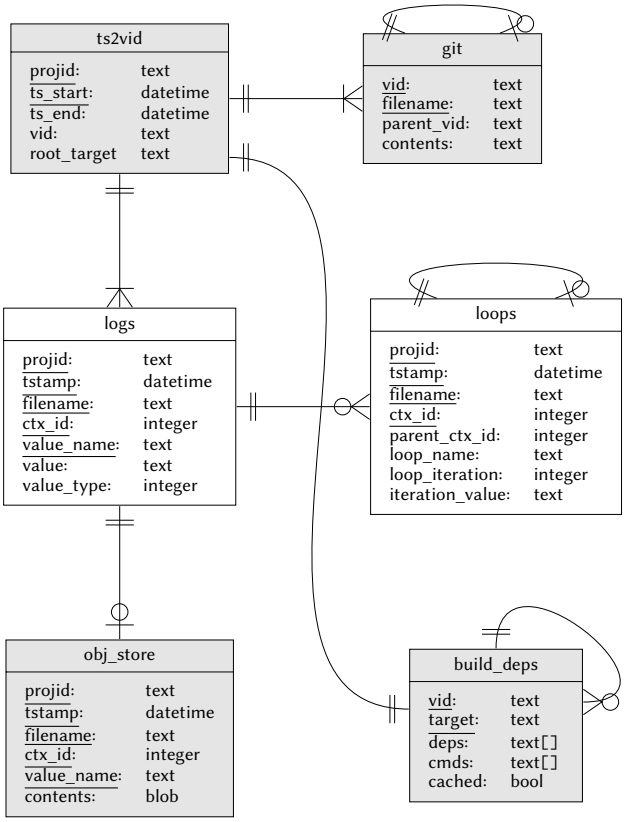


Figure 1: Extended FlorDB data model in Crow's Foot notation. Basic tables denoted in white; virtual tables in gray.

This framework provides a structured approach to understanding and managing the rich tapestry of information that underpins real-world ML applications.

3.1 Application Context

Application Context represents core information describing **what** raw data an application processes and interprets [12]. This includes all information that could be logged, such as the values of arbitrary expressions at runtime. FlorDB can capture this information post-hoc using multiversion hindsight logging (Section 2) and manages it through a unified API. This approach provides a system that supports flexible, NoSQL-like data writes and powerful, SQL-like data reads.

FlorDB provides a straightforward interface for logging via `flor.log(name, value)` statements, ensuring that each log entry is accompanied by crucial structured metadata such as `projid`, `tstamp`, `filename`, and `ctx_id`. This metadata is captured at the time of import and embedded within every log entry, enabling unambiguous identification of the log's origin and context. The `ctx_id` is generated during the initialization of `flor.loop` and indicates the specific loop context a log entry belongs to, providing

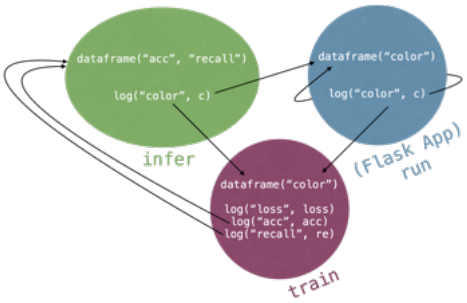
```
# Makefile

prep:
    python prep.py

infer: prep
    python infer.py

run: infer
    flask run

train: prep
    python train.py
```




```
In [1]: import flor
        flor.utils.latest(flor.dataframe("acc","recall"))
```

Out[1]:

	epochs	epochs_value	projid	tstamp	filename	acc	recall
30	1	0	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.0
31	2	1	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.0
32	3	2	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.0
33	4	3	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.0
34	5	4	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.0
35	6	5	pdf_parser	2024-05-08 10:52:03	train.py	0.978723406791687	0.6666666666666666
36	7	6	pdf_parser	2024-05-08 10:52:03	train.py	0.978723406791687	1.0
37	8	7	pdf_parser	2024-05-08 10:52:03	train.py	0.957446813583374	1.0
38	9	8	pdf_parser	2024-05-08 10:52:03	train.py	0.957446813583374	1.0
39	10	9	pdf_parser	2024-05-08 10:52:03	train.py	0.957446813583374	0.6666666666666666
40	11	10	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	1.0
41	12	11	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.3333333333333333
42	13	12	pdf_parser	2024-05-08 10:52:03	train.py	0.936170220375061	0.3333333333333333
43	14	13	pdf_parser	2024-05-08 10:52:03	train.py	0.8936170339584351	0.3333333333333333
44	15	14	pdf_parser	2024-05-08 10:52:03	train.py	0.8936170339584351	0.3333333333333333

Figure 2: ML Pipeline with Feedback: Makefile, Dataflow Diagram, and Flor Dataframe.

```
for doc_name in flor.loop("document", os.listdir(...)):
    N = get_num_pages(doc_name)
    for page in flor.loop("page", range(N)):
        # text_src is "OCR" or "TXT"
        text_src, page_text = read_page(doc_name, page)
        flor.log("text_src", text_src)
        flor.log("page_text", page_text)

        # Run some featurization
        headings, page_numbers = analyze_text(page_text)
        flor.log("headings", headings)
        flor.log("page_numbers", page_numbers)
```

```
In [1]: import flor
        flor.dataframe("headings", "page_numbers", "text_src", "page_text")
```

	projid	tstamp	filename	document	page	headings	page_numbers	text_src	page_text
0	pdf_parser	2024-04-11 09:07:54	featurize.py	6-Complaint__Judgment	1	['\n\nCOMPLAINT AND PRAYER FOR JURY TRIAL\n']	[1]	ocr	JERMAINE LYONS + INTH\t\n\t306 Paddock Lane\n\...
1	pdf_parser	2024-04-11 09:07:54	featurize.py	6-Complaint__Judgment	2	[]	[1, 2, 3, 4]	ocr	2. That the Defendants, Sgt. Martin, Officer D...
2	pdf_parser	2024-04-11 09:07:54	featurize.py	6-Complaint__Judgment	3	['COUNT TWO FALSE ARREST\n']	[11]	ocr	COUNT TWO FALSE ARREST\n\ning allegations of t...
3	pdf_parser	2024-04-11 09:07:54	featurize.py	6-Complaint__Judgment	4	['\nCOUNT FOUR\n\nVIOLATION OF THE MARYLAND DECL...	[1, 15]	ocr	15. Asa proximate result of Defendant, Officer...
4	pdf_parser	2024-04-11 09:07:54	featurize.py	6-Complaint__Judgment	5	['IVE\n\nCOUNT FI\n\nVIOLATION OF THE MARYLAND DEC...	[1]	ocr	IVE\n\nCOUNT FI\n\nVIOLATION OF THE MARYLAND DEC...
*	*	*	*	*	*	*	*	*	*
73	pdf_parser	2024-04-11 09:07:54	featurize.py	Menlo_Ave_9Jul20	29	['\n']	[2, 3, 4, 5, 6, 7, 8, 9, 13, 29, 30]	ocr	Cl\nCase: 20-034439\n\nPage 13;\n\nEvidence in t...
74	pdf_parser	2024-04-11 09:07:54	featurize.py	Menlo_Ave_9Jul20	30	['\n']	[2, 3, 4, 6, 7, 10, 20]	ocr	Investigator's Report\n\nCrime Scene\n\nse: 20-03...
75	pdf_parser	2024-04-11 09:07:54	featurize.py	Menlo_Ave_9Jul20	31	['\n']	[1, 3, 4, 5, 7]	ocr	Case: 20-034839\n\nPage 15\n\n\nAt2134 hours, the...
76	pdf_parser	2024-04-11 09:07:54	featurize.py	Menlo_Ave_9Jul20	32	['\n']	[8, 1, 16]	ocr	Investigators Report\n\nCrime Scene\n\nCase: 20-...
205	pdf_parser	2024-04-11 09:07:54	featurize.py	george_santos_report	28	[]	[1, 3, 20, 21, 28]	ocr	Organization on November 29, 2022," along with...

Figure 3: Data featurization with FlorDB

visibility into nested operations and possible cross-iteration dependencies (see `logs` and `loops` in Figure 1).

In Figure 3, we give an example of how FlorDB's logging mechanism captures data features during a document analysis process. The example illustrates how multiple documents are processed, each consisting of several pages. This example shows how FlorDB captures a wide range of metadata without requiring a predefined schema. The resulting data, including the logs of headings, page numbers, text sources, and page texts, is then accessible through the `flor.dataframe`. As shown in the bottom part of Figure 3, the dataframe presents this metadata in a structured layout, allowing users to query and analyze the data more efficiently [11].

3.2 Behavioral & Change Context

Behavioral and change context are naturally intertwined in ML pipelines: every execution both advances through the dependency graph (behavioral) and creates a new version in time (change). This context emerges naturally in FlorDB via logging executions with relevant file names and timestamps, rather than requiring manual documentation effort integrated into a workflow manager.

Behavioral context captures **how** data is created and used: dependency management, provenance and lineage, pipeline pathways, and dataflow. Change context tracks the version **history** of data, code, configuration parameters, checkpoints, and associated information [12]. Together, they help answer critical development questions: Where was this data defined? Where is it transformed? Who made the last change? Where should new transformations be added?

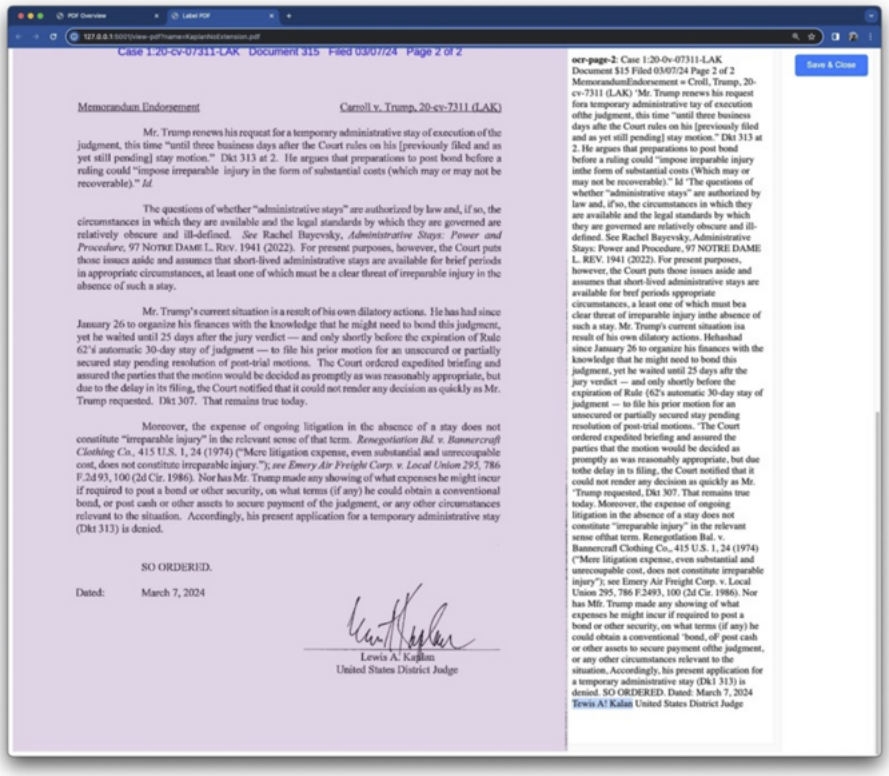
Currently, teams often manage these contexts through face-to-face interactions or communication tools like Slack or email [20]. However, this approach creates friction by relying on colleagues' availability and memory, and doesn't scale as teams change or projects evolve. While some teams try to maintain comprehensive documentation, this conflicts with the rapid iteration typical in MLOps environments.

FlorDB manages both contexts through its logging system. When users query a `flor.dataframe`, they receive not just data but its lineage and version history. To illustrate how FlorDB handles behavioral and change context, let's consider a simplified version of our document intelligence pipeline (we will cover this case in more depth in Section 4).

Our simplified pipeline comprises three main Python scripts that interact with FlorDB:

(1) `train.py`: This script fine-tunes a machine learning model using the preprocessed data. It loads the dataset prepared by `prep.py`, trains the model, and saves a model checkpoint along with performance metrics like accuracy and recall. Throughout the process, it logs the training data, model parameters, and metrics, capturing the change context over time.

(2) `infer.py`: This script performs inference using the most effective model checkpoint. `flor.dataframe("acc", "recall")` is queried to retrieve the model checkpoint with the highest recall from the execution history. By accessing the logged performance metrics and version timestamps, it ensures that predictions are made using the best available model. The



```
process_pdfs: $(PDFS) pdf_demux.py
    @echo "Processing PDF files..."
    @python pdf_demux.py
    @touch process_pdfs

featurize: process_pdfs featurize.py
    @echo "Featurizing Data..."
    @python featurize.py
    @touch featurize

train: featurize hand_label train.py
    @echo "Training..."
    @python train.py

model.pth: train export_ckpt.py
    @echo "Generating model..."
    @python export_ckpt.py

infer: model.pth infer.py
    @echo "Inferencing..."
    @python infer.py
    @touch infer

hand_label: label_by_hand.py
    @echo "Labeling by hand"
    @python label_by_hand.py
    @touch hand_label

run: featurize infer
    @echo "Starting Flask..."
    @flask run
```

Figure 4: Screenshot of the PDF Parser (left) and its respective Makefile (right).

script then processes new data and logs the predictions for further analysis.

(3) `run.py`: This script launches a Flask web application that serves the model's predictions to end-users and collects human feedback. Users can review and correct the model's outputs, providing valuable annotations. The script logs these interactions using `flor.log(name, value)`, capturing both the behavioral context (how data flows through the system) and the change context (how feedback updates the dataset).

The dependencies between these scripts are specified in a Makefile, which orchestrates the execution order based on these dependencies (left pane, Figure 2). This dependency management ensures that each script runs in the correct order and that each stage has the necessary data and models from the previous stages. While we've used a Makefile for simplicity, these dependencies can also be specified using other workflow management tools like Airflow or MLflow, depending on the project's complexity and requirements.

In this pipeline, the process cycles between collecting new human-reviewed data with `run.py` and updating the model with `train.py`. The `infer.py` script ensures that the inference stage always utilizes the best-performing model checkpoint.

Figure 2 illustrates the flow of data and transformations in the pipeline, highlighting how FlorDB captures both behavioral and change contexts. By logging with FlorDB, machine learning engineers (MLEs) not only build pipelines but also maintain a comprehensive history of data transformations and model updates. This approach reduces reliance on ad-hoc communication and documentation, allowing projects to evolve without losing crucial context.

4 PDF PARSER DEMO

Real-world AI/ML applications are backed by ML pipelines of non-trivial complexity, often encompassing both computation and human-in-the-loop feedback. This section gives an overview of the PDF Parser Demo, a practical application of FlorDB in document intelligence. We demonstrate how FlorDB can be effectively used to manage context and dataflow that spans multiple asynchronous tasks that can generate a variety of metadata via computation and human feedback.

The PDF Parser[2] is a Flask-based web application designed for efficient PDF document processing, including tasks like splitting PDFs, extracting text, and preparing data for analysis using natural language processing (NLP) techniques. Users interact with the parser through a simple web interface, as shown in Figure 4 (left). This interface allows users to navigate PDF documents and select the desired processing options.

In this demo, we aim to achieve several goals that showcase the capabilities of FlorDB:

[2]https://github.com/ucbepic/pdf_parser

- **Demonstrate Basic Functionality:**
  - *Version Tracking Across Code Changes:* We will show how FlorDB's multiversion hindsight logging operates seamlessly even when the code has been refactored, ensuring that all changes are reviewable.
  - *Flexible Pipeline Modification:* The demo illustrates how new stages can be inserted into an existing pipeline or new pathways can be incorporated into a directed acyclic graph (DAG) within the Flor data model, enabling dynamic workflow evolution.
  - *Incremental Replay Execution:* By using the Flor dataframe, we can drive incremental replay execution, re-running only the parts of the workflow that have been selected by the user, thus saving time and resources.
- **Showcase FlorDB Across Multiple Contexts:**
  - *Feature Storage and Querying After Execution:* We demonstrate how FlorDB can store and allow queries on data features post-execution without prior setup.
  - *Model Registry Functionality Post-Execution:*
    - *Comparative Metrics Across Models:* The demo compares performance metrics across different training runs after they have been executed.
    - *Post-Hoc Governance Enforcement:* We show how to apply governance policies retroactively to identify and handle issues like corrupted or malicious datasets (e.g., detecting a poisoned dataset).
  - *Metric Registry and Visualization After Execution:* FlorDB acts as a repository for metrics post-execution, allowing for visualization of results—similar to generating and augmenting TensorBoard plots—even if this wasn't configured beforehand.

Beyond highlighting the practical value of a context-rich approach, we hope this demo will serve as a reference implementation for those looking to get started with FlorDB.

4.1 PDF Extraction & Text Featurization

Once the PDF is converted into text and image formats, ensuring there is one document per page, the process of featurization begins (Figure 3). This process typically involves text extraction, feature engineering, and vectorization. This featurization process is essential for transforming raw PDF data into a structured form that is amenable to analysis and machine learning applications. The described methodology focuses on maximizing the information extracted from each page, ensuring that both textual and visual data contribute to the inferences made on the document. **Takeaway:** When used in featurization contexts (Figure 3), FlorDB can provide the functionality of a *feature store*.

4.2 Inference Pipeline

The inference pipeline automates the processing and analysis of images organized into document-specific folders. FlorDB's features enhance this pipeline in several key ways. First, FlorDB's comprehensive versioning and metadata tracking enables intelligent model selection. Through queries to FlorDB's unified metadata store (e.g., `flor.dataframe("acc", "recall")`), the pipeline can automatically select the best-performing model checkpoint based on validation metrics tracked across all training runs. This eliminates the need for manual record-keeping or separate model registries.

Second, FlorDB's logging infrastructure streamlines the processing of document pages and images. As each image is processed, pre-processed, and analyzed, FlorDB captures the full context of transformations and model predictions. This includes tracking input parameters, preprocessing steps, and model outputs in a way that maintains clear lineage between the original documents and their derived predictions.

Third, when issues arise in production, FlorDB's hindsight logging capability allows developers to retroactively add logging statements to debug problems without re-running the entire pipeline. The model then makes predictions on the images, with FlorDB logging the final inferences alongside their complete provenance. **Takeaway:** When used in inference pipelines, FlorDB functions as a comprehensive *model registry* (dataframe in Figure 2), managing not only checkpoint selection but also model versioning, metadata tracking, and performance metrics. This enables robust model lifecycle management while maintaining clear provenance of how models evolve through training iterations.

4.3 Training Pipeline

The training pipeline encapsulates a typical machine learning workflow, tailored for classifying images extracted from PDF pages (Figure 5). This pipeline performs a load of training data (line 1 in Figure 5, managed by FlorDB), and data preparation. A model architecture is defined, and the model is trained over a number of epochs. Performance is monitored by flor logging, and context is managed and tracked using FlorDB. **Takeaway:** When used in training pipelines, FlorDB can function as a *training data store* (line 1 in Figure 5), and a *model repository* (lines 3-21 in Figure 5).

4.4 Closing the Loop: Feedback via UI

The main Flask script outlines the core functionalities of a web application designed for handling PDF documents and associated image files. It includes routes for displaying and manipulating PDFs and their converted image previews within a web interface, while also incorporating human-in-the-loop feedback for improving model performance. The core of the application is structured around Flask routes that handle web requests and facilitate expert feedback. Through the UI, domain experts can review model predictions and provide corrective labels, which are managed with the same metadata infrastructure as computational steps. This human feedback loop is crucial for iteratively improving model performance and maintaining data quality.

Helper functions such as `get_colors()` fetch color data associated with the pages of a document, integrating both automated predictions and human-provided labels from the dataset (line 6 in Figure 6). The system maintains provenance for both machine-generated and human-provided labels, allowing developers to track the source and evolution of document annotations (top dataframe in Figure 5). The developer may choose to display labels generated by the model or labels entered manually by an expert end-user, with the metadata system capturing the origin and timestamp of each label.

```
In [1]: import flor
        flor.dataframe("first_page", "page_color")
```

	projid	tstamp	filename	document_value	page	first_page	page_color
0	pdf_parser	2024-04-11 09:38:23	label_by_hand.py	6-Complaint__Judgment	1	1	NaN
1	pdf_parser	2024-04-11 09:38:23	label_by_hand.py	6-Complaint__Judgment	2	0	NaN
2	pdf_parser	2024-04-11 09:38:23	label_by_hand.py	6-Complaint__Judgment	3	0	NaN
3	pdf_parser	2024-04-11 09:38:23	label_by_hand.py	6-Complaint__Judgment	4	0	NaN
4	pdf_parser	2024-04-11 09:38:23	label_by_hand.py	6-Complaint__Judgment	5	0	NaN
...	...	...	...	...	...	...	...
245	pdf_parser	2024-04-11 09:38:25	flask	6-Complaint__Judgment	12	NaN	0
246	pdf_parser	2024-04-11 09:38:25	flask	6-Complaint__Judgment	13	NaN	2
247	pdf_parser	2024-04-11 09:38:25	flask	6-Complaint__Judgment	14	NaN	1
248	pdf_parser	2024-04-11 09:38:25	flask	6-Complaint__Judgment	15	NaN	2
249	pdf_parser	2024-04-11 09:38:25	flask	6-Complaint__Judgment	16	NaN	2

250 rows × 7 columns

```python
labeled_data = flor.dataframe("first_page", "page_color")

hidden_size = flor.arg("hidden", default=500)
num_epochs = flor.arg("epochs", 5)
batch_size = flor.arg("batch_size", 32)
learning_rate = flor.arg("lr", 1e-3)
seed = flor.arg("seed", randint(0, 1e10))
...
with flor.checkpointing(model=net, optimizer=optimizer):
    for epoch in flor.loop("epoch", range(num_epochs)):
        for data in flor.loop("step", trainloader):
            inputs, labels = data
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            flor.log("loss", loss.item())
            optimizer.step()
        acc, recall = eval(net, testloader)
        flor.log("acc", acc)
        flor.log("recall", recall)
```

Figure 5: Training on labeled data managed by FlorDB

- The root route ("/") displays a home page which lists all the PDF files located in a specified directory. Each PDF file is represented with a preview image, and these images are listed on the webpage using rendered HTML templates.
- The "/view-pdf" route handles requests to view a specific PDF. Depending on user interactions and the file's existence, it can display the document in different modes such as labeled text or named entity recognition (NER) views. This route also supports expert annotation interfaces where users can correct model predictions.
- The "/save_colors" route is a POST endpoint that processes user-submitted data concerning color settings associated with a PDF's pages. This route captures this feedback data, logs it with appropriate metadata for tracking, and acknowledges the successful saving of data. The human feedback is stored with the same robust provenance tracking as computational results.

This bidirectional flow between computational processing and human expertise creates a complete feedback loop in the system. When experts provide corrections through the UI, the metadata system captures these annotations alongside the original model predictions. This human feedback can then be incorporated into subsequent model training iterations, with FlorDB maintaining clear provenance of which predictions were machine-generated versus human-corrected. This systematic approach to managing both computational and human feedback allows for continuous improvement of the model while maintaining transparency about the source and evolution of all labels in the system.

```python
@app.route("/")
def home():
    return flask.render_template("index.html")

def get_colors():
    infer = flor.dataframe("first_page", "page_color")
    infer = flor.utils.latest(
        infer[infer.document_value == pdf_names[-1]])
    if infer.page_color.isna().any():
        color = infer["first_page"].astype(int).cumsum()
        infer["page_color"] = color - 1
    return infer["page_color"].to_list()

@app.route("/save_colors", methods=["POST"])
def save_colors():
    colors = request.get_json().get("colors", [])
    pdf_name = pdf_names.pop()
    with flor.iteration("document", None, pdf_name):
        for i in flor.loop("page", range(len(colors))):
            flor.log("page_color", colors[i])
    flor.commit()
    return jsonify({"message": "Colors_saved"}), 200
```

Figure 6: Flask routes demonstrating the human-in-the-loop feedback system: The home route serves the interface, `get_colors()` retrieves existing labels from FlorDB, and `save_colors()` captures expert corrections while maintaining provenance through FlorDB's metadata management.

Takeaway: When used within human-in-the-loop interfaces, FlorDB functions as a comprehensive *feedback management system* (dataframe in Figure 5), maintaining provenance for both machine-generated and human-provided labels while enabling continuous model improvement through expert corrections.

5 DISCUSSION

This section discusses design features of FlorDB in the context of modern MLOps principles and best practices. We examine how FlorDB embodies the 3Vs of MLOps [20] and discuss its design inspiration from the Ground data context service [12].

First, we assess how FlorDB embodies and extends the 3Vs of MLOps [20]:

- **Velocity**: FlorDB enhances the speed of ML development by getting metadata definition out of the critical path of agile experimentation. Rather than requiring upfront documentation that could slow iteration, FlorDB's hindsight logging capability allows teams to move fast initially while preserving the ability to retroactively capture and analyze any needed metadata on demand. This approach maintains development velocity without sacrificing the rich context needed for effective ML lifecycle management.
- **Visibility**: The system's comprehensive logging and monitoring capabilities increase ML lifecycle transparency, facilitating debugging, optimization, and understanding of model behavior across different phases.

- **Versioning**: FlorDB radically augments version control by providing a clean, simple interface to query metadata across versions, including technology for hindsight logging across those versions. This enables detailed tracking and allows teams to observe and understand arbitrary features of models as they evolve over time.

Building on these MLOps principles, FlorDB's design draws inspiration from Ground's vision for data context services [12]. Like Ground, FlorDB recognizes that effective metadata management must capture not just static pre-declared descriptions but the full context of data usage across multiple dimensions. Specifically, FlorDB adapts Ground's "ABCs of Data Context" - Applications, Behavior, and Change - but reframes them through an MLOps lens. While Ground focused broadly on data context for analytics, FlorDB specializes these concepts for the machine learning lifecycle, treating ML models and pipelines as first-class citizens that require rich contextual tracking.

5.1 Implications for Social Justice Research

FlorDB was developed in the context of the Berkeley EPIC Data Lab[3], whose mission encompasses democratizing data work via no-code and low-code interfaces, informed by applications in the social justice domain including criminal defense and investigative journalism. FlorDB's design principles make it particularly well-suited for supporting social justice research efforts. By simplifying the tracking and organization of diverse datasets that social justice research often requires, FlorDB enables efficient data management even for users without extensive technical backgrounds. This accessibility is crucial for democratizing ML research capabilities across different communities and applications.

The system's "metadata later" approach addresses key challenges faced by resource-constrained organizations, allowing teams to focus on urgent operational demands while maintaining the ability to refine documentation post-hoc as needed. For projects involving multiple data sources, such as public health studies or legal cases, FlorDB simplifies the tracking of data movement between sources, making processes more understandable to non-specialists [5].

FlorDB's focus on transparency and accountability (i.e. visibility) is especially valuable for investigating algorithmic bias and fairness. The system's multiversion hindsight logging enables better tracking of decisions made during model training and development, helping researchers address concerns about the "black box" nature of ML models [18]. Moreover, its flexible metadata platform supports human-in-the-loop workflows, enabling continuous learning based on community feedback—a necessity for ethically deploying ML in dynamic social environments [13]. By lowering technical barriers while enabling rigorous documentation and provenance tracking, FlorDB helps empower marginalized communities to participate more fully in ML research and development. This aligns with broader goals of democratizing access to ML technologies and ensuring their benefits extend equitably across society.

6 RELATED WORK

Managing the lifecycle of ML models and associated data is crucial for efficient, reproducible workflows. FlorDB builds upon existing systems, offering comprehensive context management that integrates logging into a broader ecosystem for streamlined ML lifecycle management.

Experiment Tracking and Version Control: Systems like MLFlow [25], DVC [1], and Weights & Biases focus on managing experiments and ensuring reproducibility. They provide tools for tracking experiments, packaging code, and sharing models. While helpful for tracking model and data evolution, they primarily concentrate on experiment management without deeply integrating hindsight logging capabilities. As such, they do not (currently) address the challenges of context on demand that are the crux of our work.

Model Management and Lineage: ModelDB, Mistique, and Pachyderm emphasize version control and data lineage [14, 22, 23]. They track model lineage, capturing relationships between models, training data, and code. These systems focus on managing provenance and evolution of models and data, offering ways to query and visualize history. However, their focus is more on artifacts and less on process: for example, ModelDB does not automatically version code, and has no way of recovering missing data.

End-to-End ML Workflows: Systems like AWS SageMaker and Kubeflow provide comprehensive solutions for building, training, and deploying ML models [4, 15]. They offer tools for data labeling, model training, model hosting, and support for scalable ML workflows. Both platforms emphasize scalability and operational efficiency but primarily focus on deployment and operational aspects. FlorDB can use either system as a drop-in replacement to Make, the default build system. Other systems in this space include Helix [24] and Motion [21].

Visualization and Monitoring: TensorBoard [10], a visualization toolkit for TensorFlow, allows users to track and visualize metrics, graphs, and other aspects of ML experiments. FlorDB can be used with TensorBoard to visualize training metrics.

Data Catalogs and Metadata Management: Enterprise data catalogs like Collibra and Alation provide comprehensive solutions for organizing and discovering data assets across organizations. Collibra offers advanced governance features and automated workflows for policy management, while emphasizing data quality and observability capabilities [3]. Alation focuses on intelligent data discovery through machine learning-powered search and active metadata management, helping organizations understand data lineage and trustworthiness [2]. In the open-source domain, DataHub and Amundsen represent modern approaches to metadata management. DataHub provides a platform spanning data discovery, observability, and governance [16], while Amundsen employs a modular architecture with dedicated services for metadata, search, and frontend interactions [17]. These systems primarily focus on metadata discovery and governance, helping organizations create and maintain inventories of data assets across their digital landscape. While these platforms excel at data discovery and governance, they lack FlorDB's deep integration with ML workflows and its ability to capture and recover fine-grained execution context.

[3] https://epic.berkeley.edu

7 CONCLUSION

FlorDB provides a new, general-purpose methodology and Python-based system for managing the complex metadata—the context—associated with the machine learning lifecycle. By resolving the longstanding friction between discipline and agility in metadata management, FlorDB offers AI/ML groups a flexible yet powerful solution aligning with modern MLOps agile development practices.

The key contributions of FlorDB include features that enhance the developer experience and streamline machine learning workflows. First, incremental context maintenance allows developers to gradually build out the metadata and project structure within their existing workflows. Second, the extensions presented to support pipelines, dataflow, and feedback are backward compatible with multiversion hindsight logging, allowing for a "metadata later" approach that enables the addition and refinement of metadata post-hoc. This capability supports rapid iteration and adaptation without compromising the long-term maintainability of projects. As demonstrated through the PDF Parser demo, FlorDB can take on multiple roles within the ML pipeline, acting as a feature store, model registry, training data store, and experiment record. This versatility eliminates the need for multiple specialized tools while maintaining robust metadata management throughout the project lifecycle.

The development of FlorDB was informed by an interview study of ML Engineers to understand how they operationalize the machine learning lifecycle [20], ensuring our design decisions were grounded in real-world practices and needs. While this evidence-based design approach helped shape FlorDB's features and interface, a rigorous validation of its usability and other human factors through controlled user studies remains as important future work.

By providing a systematic way to balance agility and rigor, FlorDB creates new possibilities for scaling ML operations, improving collaboration between teams, and ensuring long-term maintainability of ML systems. As the field continues to evolve, FlorDB's flexible architecture positions it to adapt to emerging needs while maintaining its core promise: enabling teams to move fast without sacrificing reproducibility and rigor.

REFERENCES

[1] 2021. DVC: Data Version Control. https://dvc.org/. Accessed: 2024-05-31.

[2] Alation, Inc. 2024. Alation Data Catalog. https://www.alation.com/product/data-catalog/ Data catalog software for data discovery and governance. Accessed: 2024-11-11.

[3] Collibra, Inc. 2024. Collibra Data Catalog. https://www.collibra.com/us/en/products/data-catalog Data catalog software for data governance and management. Accessed: 2024-11-11.

[4] Kubeflow Community. 2021. Kubeflow: A Composable, Portable, Scalable ML Platform. https://www.kubeflow.org/. Accessed: 2024-05-31.

[5] A. Epstein and S. O'Brien. 2019. The role of social media in changing power dynamics and data transparency in public health. *Global Public Health* 14, 5 (2019), 562–571. https://doi.org/10.1080/17441692.2019.1584845

[6] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 313–324.

[7] Rolando Garcia, Anusha Dandamudi, Gabriel Matute, Lehan Wan, Joseph Gonzalez, Joseph M Hellerstein, and Koushik Sen. 2023. Multiversion Hindsight Logging for Continuous Training. *arXiv preprint arXiv:2310.07898* (2023).

[8] Rolando Garcia, Eric Liu, Vikram Sreekanti, Bobby Yan, Anusha Dandamudi, Joseph E Gonzalez, Joseph M Hellerstein, and Koushik Sen. 2021. Hindsight logging for model training. *PVLDB* 14, 4 (2021), 682–693.

[9] Rolando Garcia, Vikram Sreekanti, Neeraja Yadwadkar, Daniel Crankshaw, Joseph E Gonzalez, and Joseph M Hellerstein. 2018. Context: The missing piece in the machine learning lifecycle. In *KDD CMI Workshop*, Vol. 114. 1–4.

[10] Google. 2020. TensorBoard. tensorflow.org/tensorboard.

[11] Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. 1996. Implementing data cubes efficiently. *Acm Sigmod Record* 25, 2 (1996), 205–216.

[12] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, et al. 2017. Ground: A Data Context Service.. In *CIDR*.

[13] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–16.

[14] Pachyderm Inc. 2021. Pachyderm: Data Versioning, Data Pipelines, and Data Lineage. https://www.pachyderm.com/. Accessed: 2024-05-31.

[15] Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, et al. 2020. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 731–737.

[16] LinkedIn, Inc. 2024. DataHub: A Metadata Platform for the Modern Data Stack. https://datahubproject.io/ Open-source metadata platform for data discovery and observability. Accessed: 2024-11-11.

[17] Lyft, Inc. 2024. Amundsen: Data Discovery and Metadata Engine. https://www.amundsen.io/ Open-source data discovery and metadata engine. Accessed: 2024-11-11.

[18] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high-stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

[19] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. (2014).

[20] Shreya Shankar, Rolando Garcia, Joseph M Hellerstein, and Aditya G Parameswaran. 2024. "We have no idea how models will behave in production until production": How engineers operationalize machine learning. *Proceedings of the ACM on Human-Computer Interaction (CSCW)* (2024).

[21] Shreya Shankar and Aditya G Parameswaran. 2024. Building Reactive Large Language Model Pipelines with Motion. In *Companion of the 2024 International Conference on Management of Data*. 520–523.

[22] Manasi Vartak. 2016. ModelDB: a system for machine learning model management. In *HILDA '16*.

[23] Manasi Vartak, Joana M F. da Trindade, Samuel Madden, and Matei Zaharia. 2018. Mistique: A system to store and query model intermediates for model diagnosis. In *Proceedings of the 2018 International Conference on Management of Data*. 1285–1300.

[24] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya Parameswaran. 2018. Helix: Holistic optimization for accelerating iterative machine learning. *Proceedings of the VLDB Endowment* 12, 4 (2018), 446–460.

[25] M. Zaharia et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41 (2018), 39–45.