

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

Rolando Garcia
UC Berkeley
United States
rogarcia@berkeley.edu

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

Pragya Kallanagoudar
UC Berkeley
United States
pkallanagoudar@berkeley.edu

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

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

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

ABSTRACT

The metadata involved in integrating code, data, configuration, and feedback into predictive models is varied and complex. This complexity is further compounded by the agile development practices favored by data scientists and machine learning engineers. These practices emphasize high experimentation velocity and frequent deployments, which can make it challenging to keep track of all the relevant metadata [5, 12]. The iterative nature of agile methods means that models, datasets, and configurations are constantly evolving. Each experiment might involve tweaks to the data pre-processing steps, changes in model hyperparameters, or updates to the deployment environment. The need for rapid iteration can lead to shortcuts or oversights in documentation and metadata management. Effective metadata management requires robust yet flexible tools and practices that can integrate and organize this information without slowing down the development process. Traditional context management often emphasizes a “metadata first” approach, which can introduce significant friction for developers. FlorDB reduces this friction through multiversion hindsight logging and incremental context maintenance, allowing developers to add and refine metadata after the fact [3]. This “metadata later” approach enables a more flexible and incremental development process, allowing data scientists to focus on model creation and refinement without the burden of documentation upfront. As shown in a demo, FlorDB can be used to build AI/ML applications with integrated *train-infer* pipelines and managed feedback loops. Ultimately, the goal of FlorDB is to ensure that critical metadata is maintained accurately and efficiently, even in fast-paced agile workflows.

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 [5, 11]. 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.

1.1 A Crisis of Metadata Management

One persistent challenge is balancing agility with the rigor of up-front documentation or “metadata first” approaches. While detailed documentation and metadata are essential for reproducibility, debugging, and collaboration, insisting on comprehensive metadata from the start can hinder the agility and speed that are often crucial in early development stages. This balance is critical in enabling teams to innovate rapidly while ensuring long-term project maintainability and scalability.

At the heart of this crisis lies an enduring open problem: the tradeoff between discipline and agility. This dilemma is evident in various contexts, such as the dichotomy between schema-first and NoSQL databases, and the contrast between data warehouses and lakehouses. Schema-first approaches offer discipline and structure but lack the flexibility needed for rapidly evolving requirements. Conversely, NoSQL databases provide the necessary agility but can lead to inconsistency and disarray in metadata management.

To address this, two core goals can be established:

- (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, it should fit naturally into their open-source development environment, including tools like Python, Jupyter, and Git. For MLEs, it should leverage 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 developers regret not having it.

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that you attribute the original work to the authors and CIDR 2025. 15th Annual Conference on Innovative Data Systems Research (CIDR '25). January 19-22, Amsterdam, The Netherlands

1.2 Goals and Contributions

The scope of FlorDB¹ has been significantly expanded to encompass the entire ML lifecycle, covering 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 log statements as part of the development workflow, without imposing significant overhead. Subsequently, these log statements can be read back directly as tabular data using a standard Python `dataframe` library, Pandas, without any need for data wrangling.
- (2) **Multiversion Hindsight Logging for “Metadata on Demand”:** Backward-compatibility maintains FlorDB support for multiversion hindsight logging [3]. Our multiversion hindsight logging mechanism eliminates the need for “metadata first”. Developers can add or refine metadata post-hoc, on demand. This “metadata later” approach ensures that developers can focus on development and rapid iteration while still maintaining the benefits of structured metadata management.

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 this 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 [4], 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 record-replay added support for hindsight logging, allowing for post-hoc and on-demand querying of effectively unbounded context.

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 [3]. 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, we further extend FlorDB to support dataflow pipelines and manage feedback loops. This extension provides a seamless framework for defining and executing complex, evolving ML pipelines.

¹<https://github.com/ucbrise/flor>

2.1 Metadata on Demand

One of the key advantages of Multiversion Hindsight Logging is its support for metadata on demand, offering significant flexibility in capturing and utilizing metadata. This approach allows for the generation and refinement of metadata as needed, rather than necessitating comprehensive documentation upfront. This flexibility is especially valuable in agile environments, where rapid iteration and frequent changes are common.

In traditional systems, metadata must often be defined and captured early in the project lifecycle. However, in practice, data scientists and MLEs frequently refine their processes, models, and tools based on new insights and evolving project requirements. Multiversion Hindsight Logging accommodates these changes by enabling the capture of metadata at any stage of the project’s lifecycle. The ability to log and query metadata retrospectively means that developers can revisit and refine metadata after the initial development phases. This is particularly useful for capturing metadata that may have been overlooked or considered unnecessary in the early stages. As the project matures and the need for detailed documentation increases, developers can retroactively add and refine metadata.

FlorDB’s approach to metadata on demand provides a flexible and efficient solution to the challenges of metadata management in agile ML development. It balances the need for rapid iteration with the benefits of thorough documentation and metadata capture, ensuring that critical metadata is both available when needed and manageable over time. The changes and extensions presented in this paper are fully backward-compatible with FlorDB’s multiversion hindsight logging capabilities, extending metadata on demand to entire ML pipelines, not just individual training scripts.

2.2 FlorDB Extended API

FlorDB’s API captures metadata about the executing file, eliminating the need to restate dataflow dependencies; the Makefile suffices. By profiling runtime metadata, including the executed file’s name, FlorDB remains agnostic to the workflow or dataflow management system, functioning seamlessly whether called by Make or Airflow (or others) without requiring refactoring.

The Flor API, as presented in Garcia et al. (2023) [3], includes:

- **`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 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`.

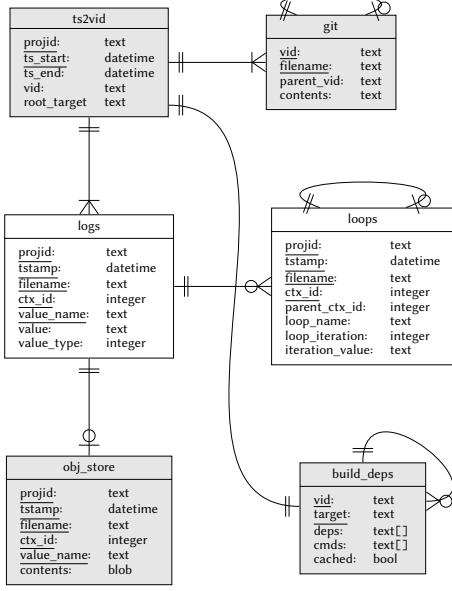


Figure 1: Extended FlorDB data model in Crow’s Foot notation. Basic tables denoted in white; virtual tables in gray.

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.

Our conceptualization of context is drawn from the work of Hellerstein et al. (2017) [8], who proposed it as an extension of traditional database metadata. They introduced the “ABCs of Context” mnemonic, which we find particularly useful for understanding and navigating the complex landscape of ML metadata. This framework provides a structured approach to capturing and managing the rich tapestry of information that underpins real-world ML applications.

3.1 Contextualized Logging

“Application Context” represents core information describing **what** raw data an application processes and interprets [8]. This encompasses all information that *could be* logged, such as the values of arbitrary expressions at runtime, which FlorDB can capture post-hoc

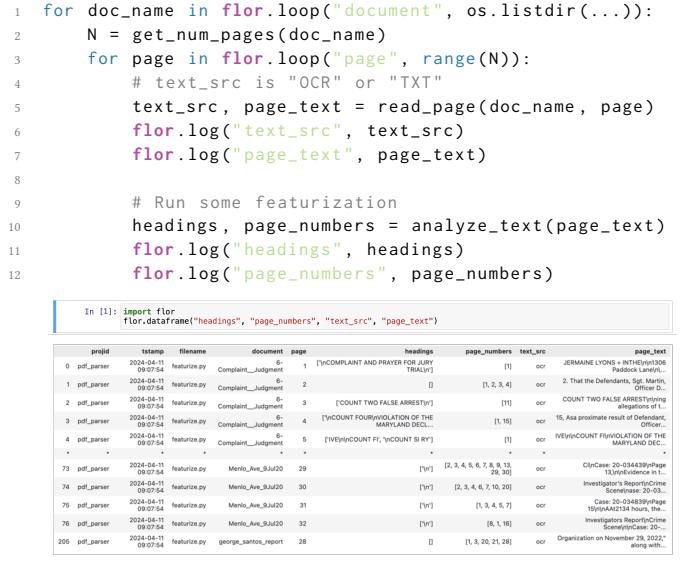


Figure 2: Data featurization with FlorDB

with multiversion hindsight logging (Section 2) and manages using a unified API, providing 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 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 visibility into nested operations and possible cross-iteration dependencies (see `logs` and `loops` in Figure 1).

In Figure 2, 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 efficiently 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 2, the dataframe presents this metadata in a structured layout, allowing users to query and analyze the data more efficiently [7].

3.2 Managing Dataflow and Feedback Loops

As we have discussed throughout this paper, context is something you log, and it’s best when it can be logged post-hoc and on demand. But also, and more importantly, context is something you build and maintain. “Build Context” is information about **how** data is created and used: e.g., dependency management for distribution and building across different machines and by different people; provenance and lineage; routes, pathways, or branches in pipelines; and flow of control and data. “Change Context” is information

```

1 # Makefile
2
3 prep:
4     python prep.py
5
6 infer: prep
7     python infer.py
8
9 run: infer
10    flask run
11
12 train: prep
13     python train.py

```

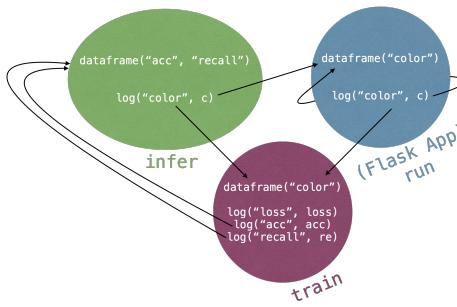


Figure 3: ML Pipeline with Feedback: Makefile, Dataflow Diagram, and Flor Dataframe.

```

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

projid      timestamp   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 x 7 columns

```

```

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 4: Training on labeled data managed by FlorDB

about the version **history** of data, code, configuration parameters, checkpoints, and associated information, including changes over time to both structure and content [8]. Build and Change Context is what allows us to answer important development questions such as the following: Where did you define this data? Where do you transform it? Who was the last person to change this? Where should I make changes to add another transformation or filter? And so on.

Currently, these questions are often answered through face-to-face interactions or via communication tools like Slack or email [12]. However, this approach has several limitations, primarily: i) friction, inefficiency and possible delays, as it relies on the availability

In [1]:	import flor flor.utils.latest(flor.dataframe("acc", "recall"))						
Out[1]:							
epoch	epoch_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	0.97872406791687	0.6666666666666666
36	7	6	pdf_parser	2024-05-08 10:52:03	train.py	0.97872406791687	1.0
37	8	7	pdf_parser	2024-05-08 10:52:03	train.py	0.9744815838374	1.0
38	9	8	pdf_parser	2024-05-08 10:52:03	train.py	0.9544815838374	1.0
39	10	9	pdf_parser	2024-05-08 10:52:03	train.py	0.95744815838374	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	0.8936170336594351	0.3333333333333333
44	15	14	pdf_parser	2024-05-08 10:52:03	train.py	0.8936170336594351	0.3333333333333333

and memory of colleagues, and ii) lack of scalability, especially as teams change or projects evolve. Alternatively, developers may try to anticipate frequently asked questions (FAQs) and maintain comprehensive READMEs, documentation, wikis, or forums. However, as previously noted, this level of discipline and methodology can conflict with the high rate of iteration and agility typical in MLOps environments.

FlorDB addresses these challenges by managing context effectively. When a colleague has questions about the project, they can simply invoke a `flor.dataframe` with the desired data (originating from `flor.log(name, value)`). This dataframe automatically gathers the requisite context alongside the queried data. In essence, FlorDB is a step towards making work self-documenting, a direct consequence of effective context management. By logging with FlorDB, the MLE can build an ML pipeline, and by building that pipeline, a user or colleague can navigate versions, answer questions about provenance or dataflow, identify the files and lines where data is entered and transformed, and so on.

Figure 3 illustrates an ML pipeline for a document intelligence application with feedback loops defined in a Makefile. The pipeline consists of three main components:

- (1) **infer**: Uses `flor.dataframe("acc", "recall")` to select the model checkpoint with the highest recall (or a fallback model if no checkpoint exists). The corresponding dataframe queried by `infer` is shown on the right.
- (2) **run**: Launches a Flask web application to display model predictions, overlaying a color layer (denoting polarity) over each page for human review. Upon confirmation, Flask logs the final color of each page, incorporating feedback into the cycle.
- (3) **train**: Utilizes the human-reviewed data (feedback) to fine-tune the model and improve its performance for future rounds of document analysis (Figure 4).

This process is iterative, alternating between calls to make `run` and make `train`. Throughout this cycle, `flor.log(name, value)` manages writes, while `flor.dataframe(*args)` handles reads, ensuring efficient data flow and context management.

4 PDF PARSER DEMO

This section gives an overview of the PDF Parser Demo, a practical application of FlorDB in document intelligence. We use FlorDB

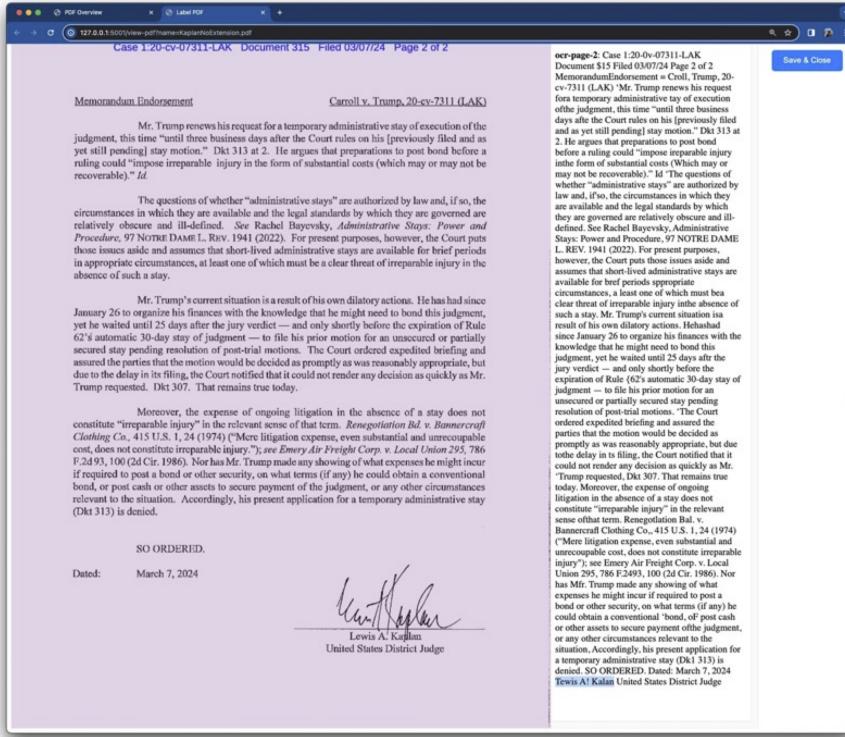


Figure 5: Screenshot of the PDF Parser (left) and its respective Makefile (right).

to manage context and dataflow, and demonstrate how it can be effectively used to implement and manage a real-world AI/ML applications with ML pipelines of moderate complexity. The PDF Parser² 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 5 (left). This interface allows users to navigate PDF documents and select the desired processing options. Beyond demonstrating the practical value of a context-rich approach, we hope it 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 2). 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 2), FlorDB can morph into a feature store.

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

4.2 Inference Pipeline

The inference pipeline automates the processing and analysis of images organized into document-specific folders. The inference pipeline queries FlorDB to select a model checkpoint, and uses it to systematically processes each document's pages and images. For each image, it loads the image file, applies standard pre-processing steps such as resizing and normalization, and converts it into a format suitable for input to the pre-trained model. The pre-trained model is chosen based on user-defined validation metrics. The model then makes predictions on the images, logging the final inferences. **Takeaway:** When used in inference pipelines, FlorDB can morph into a model registry (dataframe in Figure 3) for checkpoint selection.

4.3 Training Pipeline

The training pipeline encapsulates a typical machine learning workflow, tailored for classifying images extracted from PDF pages (Figure 4). This pipeline performs a load of training data (line 1 in Figure 4, 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 morph into a training data store (line 1 in Figure 4), and a model repository (lines 3-21 in Figure 4).

```

1 process_pdffs: $(PDFS) pdf_demux.py
2   @echo "Processing PDF files..."
3   @python pdf_demux.py
4   @touch process_pdffs
5
6 featurize: process_pdffs featurize.py
7   @echo "Featurizing Data..."
8   @python featurize.py
9   @touch featurize
10
11 train: featurize hand_label train.py
12   @echo "Training..."
13   @python train.py
14
15 model.pth: train export_ckpt.py
16   @echo "Generating model..."
17   @python export_ckpt.py
18
19 infer: model.pth infer.py
20   @echo "Inferencing..."
21   @python infer.py
22   @touch infer
23
24 hand_label: label_by_hand.py
25   @echo "Labeling by hand"
26   @python label_by_hand.py
27   @touch hand_label
28
29 run: featurize infer
30   @echo "Starting Flask..."
31   @flask run

```

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 [12] and discuss its design inspiration from the Ground data context service [8].

We assess FlorDB through the lens of the 3Vs of MLOps [12]:

- **Velocity:** FlorDB enhances the speed of ML model development, testing, and deployment by streamlining the integration of new data and models, enabling rapid iteration and adaptation.
- **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 leverages version control for managing data, models, and code changes, supporting detailed tracking and allowing teams to understand model evolution over time.

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 [17], DVC [1], and Weights & Biases focus on managing experiments and ensuring reproducibility. They provide tools for tracking experiments, packaging code, and sharing models. While crucial for tracking model and data evolution, they primarily concentrate on experiment management without deeply integrating hindsight logging capabilities.

Model Management and Lineage: ModelDB, Mistique, and Pachyderm emphasize version control and data lineage [9, 14, 15]. 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: AWS SageMaker and Kubeflow provide comprehensive solutions for building, training, and deploying ML models [2, 10]. They offer tools for data labeling, model training, and hosting, supporting 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 [16] and Motion [13].

Visualization and Monitoring: TensorBoard [6], 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.

7 CONCLUSION

FlorDB represents a step toward managing the complex metadata and context associated with the machine learning lifecycle. By addressing the enduring challenges of metadata management in

AI/ML development, FlorDB offers a flexible yet powerful solution aligning with modern MLOps agile development practices.

The key contributions of FlorDB include several 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 **Multiversioin Hindsight Logging** allowing for a “metadata later” approach, which enables the addition and refinement of metadata post-hoc. This capability supports rapid iteration and adaptation without compromising the long-term maintainability of projects. Last, FlorDB offers a **Unified Framework & Versatility** by integrating features from bespoke machine learning metadata systems. It provides a cohesive solution for managing the entire ML lifecycle, from data preparation to model deployment and feedback integration. For instance, as demonstrated through the PDF Parser application, FlorDB can take on multiple roles within the ML pipeline, such as acting as a feature store, model registry, training data store, and experiment record.

REFERENCES

- [1] 2021. DVC: Data Version Control. <https://dvc.org/>. Accessed: 2024-05-31.
- [2] Kubeflow Community. 2021. Kubeflow: A Composable, Portable, Scalable ML Platform. <https://www.kubeflow.org/>. Accessed: 2024-05-31.
- [3] Rolando Garcia, Anusha Dandamudi, Gabriel Matute, Lehan Wan, Joseph Gonzalez, Joseph M Hellerstein, and Koushik Sen. 2023. Multiversioin Hindsight Logging for Continuous Training. *arXiv preprint arXiv:2310.07898* (2023).
- [4] 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.
- [5] 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.
- [6] Google. 2020. TensorBoard. tensorflow.org/tensorboard.
- [7] Venky Harinarayanan, Anand Rajaraman, and Jeffrey D Ullman. 1996. Implementing data cubes efficiently. *AcM Sigmod Record* 25, 2 (1996), 205–216.
- [8] 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*.
- [9] Pachyderm Inc. 2021. Pachyderm: Data Versioning, Data Pipelines, and Data Lineage. <https://www.pachyderm.com/>. Accessed: 2024-05-31.
- [10] 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.
- [11] 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).
- [12] 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).
- [13] 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.
- [14] Manasi Vartak. 2016. ModelDB: a system for machine learning model management. In *HILDA '16*.
- [15] 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.
- [16] 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.
- [17] M. Zaharia et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41 (2018), 39–45.