Context is All You Need: Closing the Loop in the Machine Learning Lifecycle

Rolando Garcia
UC Berkeley
rogarcia@berkeley.edu

Pragya Kallanagoudar
UC Berkeley
pkallanagoudar@berkeley.edu

Chithra Anand
UC Berkeley
chithra.rajan@berkeley.edu

Sarah E. Chasins
UC Berkeley
schasins@berkeley.edu

Joseph M. Hellerstein
UC Berkeley
hellerstein@berkeley.edu

Aditya G. Parameswaran
UC Berkeley
adityagp@berkeley.edu

ABSTRACT

Effective software engineering in AI/ML remains challenging due to the complexities of integrating code, data, and configuration parameters into predictive models. We extend FlorDB to manage the complete "ABCs" of context [6], creating a comprehensive context management system for the ML lifecycle. FlorDB focuses on essential developer tools: logging, dataframe queries, and automatic version control, enabling efficient workflow management and streamlined operations. Our system enhances scalability and robustness of ML solutions by providing powerful tracking and analysis capabilities without additional overhead. Traditional context management often emphasizes a "metadata first" approach, which can introduce significant friction for developers. FlorDB reduces this friction through hindsight logging, allowing developers to add and refine metadata after the fact. This "metadata later" approach provides flexibility and ease of use, enabling developers to focus on building and improving models without being burdened by the immediate need to meticulously document every step up front. By supporting hindsight logging, FlorDB offers a more intuitive and developer-friendly way to manage contextual metadata throughout the machine learning lifecycle, enabling developers to focus on data processing, model building, and improvement.

ACM Reference Format:
Rolando Garcia, Pragya Kallanagoudar, Chithra Anand, Sarah E. Chasins, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2024. Context is All You Need: Closing the Loop in the Machine Learning Lifecycle. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the rapidly evolving field of Artificial Intelligence (AI) and Machine Learning (ML), the software engineering and maintenance of intelligent applications has emerged as an enduring challenge [10]. The blending of code, data, and configuration parameters into predictive models blurs traditional abstraction boundaries, straining conventional software engineering systems. Furthermore, the complexity introduced by this conglomeration limits the ability of team members to effectively partition and manage distinct components of the project [11]. We argue that these challenges primarily stem from either the absence of context or inadequate context conservation throughout the AI/ML application development lifecycle [4].

1.1 Streamlining the ML Lifecycle

The ML lifecycle involves several interconnected stages, from problem formulation and data collection to model deployment and maintenance. Optimizing this lifecycle requires addressing technical challenges at each stage and integrating processes and tools to reduce friction and speed up development. Key elements include: i) building and maintaining robust data infrastructure; ii) automating model training and evaluation; iii) continuous integration and multi-stage deployment (CI/CD); and iv) monitoring and merging feedback loops.

One persistent challenge is balancing agility with the rigor of "strong typing" or "metadata first" approaches. FlorDB's approach substantially ameliorates this tension in two ways:

(1) **Low-Friction Metadata via Logging and Dataframes:** 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) **Hindsight Logging for "Metadata on Demand":** Our 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.

1.2 FlorDB & The ABCs of Context

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

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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) [6], 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.

The ABCs of Context are as follows:

A. **Application Context (What)**: Core information that describes **what** raw bits an application sees and interprets for its use. Most comprehensively, this involves any information that *could be* logged; i.e., the value of arbitrary expressions at runtime.
B. **Build Context (How)**: 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.
C. **Change Context (When)**: Information about the version **history** of data, code, configuration parameters, and associated information, including changes over time to both structure and content.

FlorDB[1] addresses the lack of a standard mechanism for managing these details by capturing and managing extensive metadata in a familiar, unified API of log statements and dataframe queries.

1.3 Goals and Contributions

In this paper, we aim to expand the management of context beyond continuous training [2] to span the entire workflow, emphasizing not only individual tasks (e.g. *training*) but entire pipelines and their regular execution. Key goals and contributions include:

(1) **Widening the Scope**: We aim to cover the whole workflow, or ML lifecycle, focusing not just on individual tasks or operations but entire pipelines and their regular execution. To support this, we have developed Build extensions for FlorDB that enable the management of the full ABCs of context. This includes mechanisms like `flor.log` (for writing) and `flor.dataframe` (for reading), which were initially designed for multiversion hindsight logging [2] but are now adapted to manage Build context in any workflow.
(2) **Build System Agnosticism**: FlorDB is designed to function seamlessly with any build system (e.g. Make, MLFlow [16], etc.). FlorDB uses Python profiling tools such as `inspect` at import time to probe Build context automatically, such as the name and location of the hosting Git repository, and the name of the Python file being executed. FlorDB intercepts Build context at the Python-script level, making it operable with any build system of choice.
(3) **API Stability Following Build Extensions**: The FlorDB APIs to manage change over time, i.e. with multiversion hindsight logging, remain backward-compatible and fully operational. We have extended FlorDB to passively capture and expose the Build context using the same APIs previously developed for managing Application and Change contexts. The capabilities of FlorDB have been extended to manage the full ABCs of context without affecting the API in [2].
(4) **Simplified yet Powerful Abstraction**: A significant benefit of our open, standard approach is that we simplify and improve the abstraction of metadata, subsuming features from various bespoke ML metadata systems — such as feature stores, model registries, and label registries — into a unified and robust framework. By demonstrating effective methodology by which ML engineers can manage context, we illustrate how FlorDB can be used to build workflows and streamline repeated ML operations, closing the loop in the ML lifecycle.

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

```
# Makefile
prep:
    python prep.py

train: prep
    python train.py
```

```
flor.dataframe("batch_size")
```

	projid	tstamp	filename	batch_size
0	manuscript	2024-06-24 11:01:54	prep.py	64
1	manuscript	2024-06-24 11:03:14	train.py	32

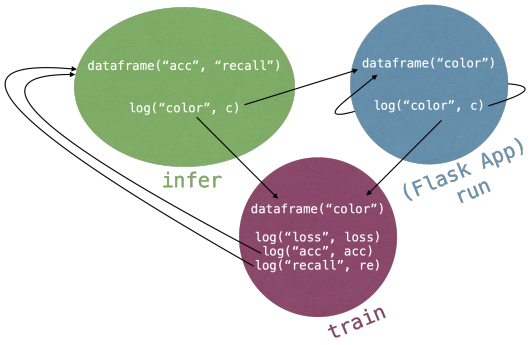
Figure 1: Makefile and Flor Dataframe after one call of `make train`. The logging of `batch_size` is unambiguous, given the `projid`, `tstamp`, and `filename` which are captured automatically and comprise relevant build and change context.

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

2 FLOW WITH FLORDB

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 [3], 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 manages application context through hindsight logging, allowing for post-hoc and on-demand querying or materialization of effectively unbounded context [3].

FlorDB extends Flor's capabilities by combining application context with change context through multiversion hindsight logging [2]. It employs a relational model, accessible via `flor.dataframe`, which 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 build context, thus completing the ABCs of context (Application, Build, Change). This extension provides a seamless framework for defining and executing complex, evolving machine learning pipelines. While FlorDB uses Make by default, it is designed to be agnostic to the build system or workflow manager, and is compatible with alternatives like Airflow, MLFlow, Slurm, or any preferred workflow management system.




```
# Makefile

prep:
    python prep.py

infer: prep
    python infer.py

run: infer
    flask run

train: prep
    python train.py
```

Figure 2: Dataflow diagram and Makefile. `infer` uses `flor.dataframe("acc", "recall")` to selects the model checkpoint with highest recall (or a fallback model if no checkpoint exists) and uses it to segment documents, labeling pages with colors such that contiguous pages with the same color belong to the same segment. `run` launches a Flask web application to display the model predictions, overlaying a color layer over each page for human review. On confirmation, Flask logs the final color of each page. `train` uses the human-reviewed data (or feedback) to tune the model and improve its performance on future rounds of document segmentation. The process is repeated again and again, oscillating between calls to `make run` and `make train`. `flor.log(name, value)` manages writes, and `flor.dataframe(*args)` manages reads.

2.1 Build Context & Unambiguous Logging

The extended capabilities of FlorDB are designed to complement dependency management systems like Make. While Make (or equivalent) handles the execution order of tasks based on defined dependencies, FlorDB extends this process by automatically capturing, tracking, and organizing the associated metadata throughout the execution of a workflow. In a Makefile-defined workflow with Python tasks like `prep.py` and `train.py`, FlorDB activates via `import flor` to capture crucial metadata including:

- Task start and end times, providing execution timeline
- Dependency details from the developer-managed build file (default Makefile)
- Environmental parameters (OS, Python environment, hardware configurations)
- Project context (Git repository directory)
- File-specific metadata (script filenames via Python `inspect` module)

FlorDB captures this data at import time, incorporating it in every `flor.log(name, value)` write, ensuring unambiguous log origins. For instance, if both `prep.py` and `train.py` log batch size, `flor.dataframe(batch_size)` remains unambiguous due to captured `projid` and `filename` (see Figure 1). By leveraging Python profiling tools and system calls, FlorDB gathers comprehensive data without disrupting normal workflow operations.

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) [2], 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`.

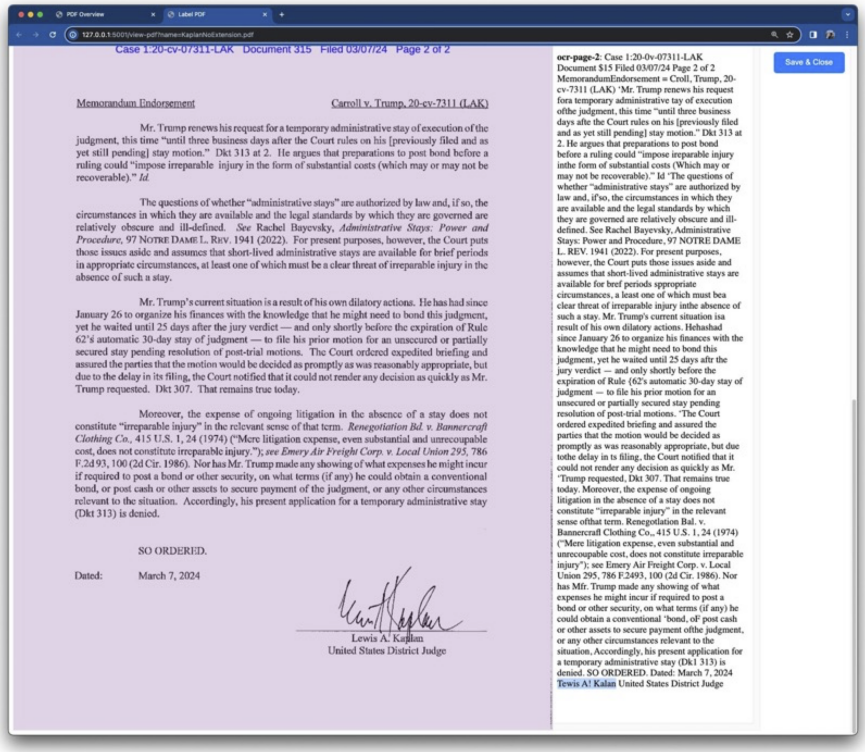
This chapter introduces an extension to the API:

- `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 CLOSING THE LOOP IN THE ML LIFECYCLE

In a decoupled architecture of multiple applications and backend services, a unified view of overall context can serve as a narrow gateway — a single point of access for the basic information about data, metadata and their usage. We envision FlorDB streamlining ML engineer's workflow by integrating fragmented metadata from scattered sources within `flor.dataframe`: a single, unified solution.

FlorDB promotes an agile approach that emphasizes tight-loop development. This approach can involve an iterative multi-phase development process (or workflow) in which an initial phase (e.g.




```
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 3: Screenshot of the PDF Parser (left) and its respective Makefile (right). A scanned document appears on the left pane, it can be scrolled vertically from page to page. A colored layer is overlaid over each page for document segmentation: contiguous pages with the same color belong to the same segment. Extracted text and other features are displayed on the right page, grouped by page. The end-user may correct segmentation errors by clicking on the page overlay to cycle over colors; they may correct OCR errors by editing the text in the right pane directly. Workflow consists of *train* and *infer* pipelines. On *run*, PDF Parser extracts and featurizes documents and feeds them through the inference pipeline to display document segmentation plan to the end-user. Following any corrections and confirmation, the newly labeled data becomes training data for future rounds of training, improving future runs of the *infer* pipeline.

`infer` in Figure 2) leverages inference pipelines to generate predictions based on existing models. These predictions are then presented to users in a subsequent phase (e.g. `run` in Figure 2), where human feedback is incorporated to refine and train subsequent models. This creates a continuous feedback loop where user-verified labels are ingested by *train* pipelines to create checkpoints for the *infer* pipelines. In Figure 2 we show metadata being written via `flor.log` statements, and queried via the `flor.dataframe` API.

While FlorDB supports generic Python pipelines, workflows, and build jobs, it is particularly helpful in managing closed loops, which are prevalent in production environments. Effective management of these closed loops is often challenging and can be a point of failure, making them a primary focus of our case study. The three cases presented next demonstrate instances of this agile development framework. By successfully managing context in the ML lifecycle, ML engineers can close the loop, integrating feedback from later stages at each step and providing each participant with a comprehensive view of the entire lifecycle.

4 PDF PARSER: A FLORDB USE-CASE

This section explores a practical application of FlorDB in document intelligence through the PDF Parser project. We demonstrate how FlorDB can be effectively used to implement and manage a real-world document intelligence applications.

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 3 (left). This interface allows users to navigate PDF documents and select the desired processing options. The focus of this use-case is simplicity, serving as a practical guide for those new to FlorDB implementations. By examining how FlorDB is used to implement the PDF Parser, we provide a clear example of how to leverage FlorDB's capabilities in real-world scenarios.

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

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 (see `featurize` target in Figure 3). This process typically involves the following steps:

- **Text Extraction**: The `pdf_demux.py` script is used to demultiplex each page of the PDF into separate text and image files. This is crucial for handling documents where text and visual data are intermixed. The text extraction component employs Optical Character Recognition (OCR) to convert any text found in images into machine-readable format.
- **Feature Engineering**: The `featurize.py` script takes the cleaned text and applies various natural language processing (NLP) techniques to extract meaningful features from the text. This might include tokenization, stemming, and the creation of n-grams. For images, feature engineering could involve extracting color histograms, texture patterns, or edge statistics which are useful for computer vision models.
- **Vectorization**: The text features are then vectorized into a format suitable for machine learning models. This often involves transforming categorical data into numerical data through techniques like neural embedding, one-hot encoding or the use of TF-IDF (Term Frequency-Inverse Document Frequency) for text.

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.

4.2 Inference Pipeline

The inference pipeline automates the processing and analysis of images organized into document-specific folders. The pipeline, contained in `infer.py`, 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 model then makes predictions on the images, logging the final inferences.

4.3 UI: Web Application on Flask

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. The core of the application is structured around Flask routes that handle web requests:

- 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 handles dynamic user settings that can influence how the document is processed and displayed.
- 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 data, logs it for tracking, and acknowledges the successful saving of data.

Helper functions such as `get_colors()` fetch color data associated with the pages of a document, integrating latest updates from a dataset. The developer may choose to display labels generated by the model or labels entered manually by an expert end-user.

4.4 Training Pipeline

The training pipeline encapsulates a typical machine learning workflow, tailored for classifying images extracted from PDF pages. Key components include:

- **Model Architecture:** A modified ResNet-18 network, adapted for binary classification of first pages vs. other pages. The model's earlier layers are frozen, with only the final layer trained to optimize performance.
- **Data Processing:** Images undergo standardization and augmentation, including resizing, cropping, tensor conversion, and pixel normalization, preparing them for neural network processing.
- **Training Process:** The model is trained over multiple epochs, each comprising a training phase and validation phase.
- **Performance Tracking:** FlorDB is utilized to log crucial metrics such as loss, accuracy, and recall throughout the training and validation processes.

This streamlined pipeline effectively combines data preparation, model architecture, training dynamics, and performance monitoring, all managed and tracked using FlorDB.

5 DISCUSSION

This section evaluates FlorDB in the context of modern MLOps principles and best practices. We examine how FlorDB embodies the 3Vs of MLOps [11] and discuss its design inspiration from the Ground data context service [6] and Postel's law of robustness.

5.1 MLOps Validation Criteria

We assess FlorDB through the lens of the 3Vs of MLOps [11]:

- **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.

5.2 Postel's Law of Robustness

Inspired by the Ground data context service [6], FlorDB adopts Postel's law of robustness:

> "Be conservative in what you do, be liberal in what you accept from others."

This principle is crucial in the complex ML lifecycle, where diverse systems with different APIs interact. FlorDB implements this principle through a logging metaphor for writing data and a dataframe metaphor for reading it. This approach makes FlorDB as easy to write to as NoSQL and as easy to read from as SQL. Contextual information is organized into a Flor Dataframe, serving as a common layer for managing metadata from various sources, enhancing accessibility and manageability for users.

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 [16], DVC [8], 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 [7, 13, 14]. 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 [1, 9]. 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 [15] and Motion [12].

Visualization and Monitoring: TensorBoard [5], 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.

FlorDB distinguishes itself through comprehensive context management across the ML lifecycle. By focusing on the ABCs of context (Application, Build, and Change), it captures and integrates metadata spanning data and model management. This includes tracking code execution, data flow, dependencies, and version history throughout the development process.

7 CONCLUSION

This paper introduces an extended version of FlorDB, a system that manages context throughout the machine learning lifecycle by capturing and integrating metadata across Application, Build, and Change contexts (the ABCs of context). FlorDB offers a unified solution for the complex landscape of AI/ML development. We demonstrated FlorDB's capabilities through a document intelligence use case, showing its ability to streamline the ML lifecycle end-to-end. This use case highlighted how FlorDB facilitates high-velocity development cycles, enables reliable experimentation, and enhances collaboration by preserving and sharing essential context across all stages of the ML workflow.

FlorDB represents a significant advancement in AI/ML development, addressing the critical need for context preservation and accessibility. Its flexible architecture supports integration with various build and workflow management tools, offering a versatile solution for managing complex ML workflows. FlorDB has the potential to become an indispensable tool in the MLOps ecosystem, and ultimately helping teams drive innovation in the field.

REFERENCES

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

[2] 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).

[3] Rolando Garcia, Eric Liu, Vikram Sreekanti, Bobby Yan, Anusha Dandamudi, Joseph E Gonzalez, Joseph M Hellerstein, and Koushik Sen. 2020. Hindsight logging for model training. *arXiv preprint arXiv:2006.07357* (2020).

[4] 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.

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

[6] 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*. Citeseer.

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

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

[9] 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.

[10] 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).

[11] 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).

[12] 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.

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

[14] 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.

[15] 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.

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