A Lifecycle-Oriented Taxonomy of Open-Source Tools for Machine Learning

Hamza Mallam Musa Mohammad[1], Isa Hussain Adam Mohammed[2], Abdullah Bajaber[3],
Syed Abdur Rahman[4], Anas Sani[5], Atif Naseer[6]
University of London, London[1]
Data Science Program, University of Colorado Boulder, Boulder, CO, USA[2]
Department of Computer Science and Artificial Intelligence, Umm Al-Qura University, Makkah, 21421, Saudi Arabia[3]
Jawaharlal Nehru Technological University, Hyderabad, India[4]
Goldsmiths, University of London[5]
Deanship of Postgraduate Studies and Research, Umm Al-Qura University, Makkah, 21955, Saudi Arabia[6]

***Abstract*—Machine learning (ML) offers various tools, frameworks and platforms for resolving complex problems in computational science and engineering. Machine learning frameworks have emerged as the cornerstone of modern research and innovation. It redefines how knowledge is produced, validated, and disseminated. Open-source machine learning frameworks are emerging as a promising way to solve the challenges of large datasets, real-time constraints and heterogeneous system components. This study provides an extensive overview of open source tools based on the ML lifecycle. These tools are evaluated based on their purpose and key features for each stage of lifecycle, assisting researchers and practitioners in making informed decisions according to their requirements. The key challenges are identified and future research directions are also outlined.**

***Keywords*—Open-source; lifecycle; taxonomy; machine learning; challenges**

I. Introduction

Recent developments in AI and ML, especially in their capabilities to automate complex operations, uncover hidden patterns from large amounts of data and help in predictive analysis. This insight is valuable in making decisions and have a significant impact on several sectors like the healthcare, finance, manufacturing and entertainment [1], [2], [3]. The core factors for these advances include open source solutions that ease the creation, development, testing and deployment of AI applications and ML models [4], [5], [6], [7], [8].

Open-source tools help to improve development, as well as make advanced analytics more accessible for users with different experience levels [9], [11], [10]. Nowadays, there is a variety of AI/ML tools available in the market to fulfill diverse needs and preferences of their potential customers [13], [12]. The available resources include fundamental ML libraries such as TensorFlow, PyTorch and high-level frameworks like Keras or Scikit-learn. In addition, complementary components enable structured knowledge representation, rule-based reasoning, uncertainty modeling, sequential decision-making, and integration with real-world systems such as robotics platforms. Furthermore, these tools offer essential functionalities that cover such areas as data pre-processing, features selection, algorithm choice, optimization, inference, model evaluation, deployment and help in achieving high performance in AI models [14], [15]. Despite the variety of available options, choosing an adequate tool is still difficult. It should be noted that the choice of the right tool depends on the type of problems, complexity of data, licensing issues and skills of the practitioner [7], [16], [17], [18].

The adoption of open-source artificial intelligence (AI) and machine learning (ML) frameworks has become fundamental to current computational research and innovation endeavors. In addition to economic benefits that open source technology offers, openness, collaboration and availability are among its most important features. These qualities align well with sound scientific research. Since licensing restrictions are not applicable to open source software frameworks, researchers, entrepreneurs, and other public institutions can use them for collecting data, processing, computations and model development. The open access to source code, model designs and training steps also improves reproducibility, which is the foundation of scientific research integrity.

Collective intelligence of global open-source communities works as an ongoing peer-review system. It is helpful for improvement of tools via prompt updates, bug fixes and new features. It is worth noting that such processes usually occur much faster compared to similar operations in closed proprietary projects. These community efforts promote best practices via providing quality documents, benchmarks and educational resources. The common formats and APIs help integrate various solutions within different organizations. Version history help in detecting early stages problems and facilitates track changes. Researchers and developers can reuse code, pipelines and shared libraries to match specific needs.Public availability of code enables a broad community of experts to identify bugs, detect vulnerabilities, and improve overall software reliability. Open-source tools are also useful in resource-constrained environments for developing local AI expertise. This approach facilitates global participation and provides more diverse ideas. Open-source solutions are very beneficial in improving governance and cyber-security. This aspect is particularly relevant in healthcare, transport, and climate science for continuous research growth in AI and machine learning.

This study presents a detailed review of the open-source ML tools by considering entire lifecycle based on SLR methodology [19], [20]. We focus on the features, advantages,

and limitations of these tools. This study aims to provide useful insights regarding the analysis of such tools and helping individuals in decision-making based on individual needs. The contributions of this research work are as follows:

- A detailed analysis of different stages of the machine learning lifecycle is provided.
- A lifecycle-based taxonomy is introduced for open source machine learning tools, which provide insight to different open source ML tools based on their functionalities. We explore their key features, use cases, and other distinguishing characteristics.
- A detailed discussion and comparative analysis is established on widely used ML tools, emphasizing their unique features and suitability for different types of machine learning applications.
- The key challenges, risks, and future directions are identified, providing insights into potential areas for improvement and innovation.

The remainder of this study is organized as follows: Section II present research methodology and initial analysis. Section III presents an end-to-end view of the machine learning lifecycle establishing the foundation for our analysis. Section IV introduces a lifecycle-oriented taxonomy to organize open-source tools by functionality, summarizes their capabilities, and use cases. Section V provides a comparative analysis and discusses trade-offs and selection guidance for different application contexts. Section VI outlines key challenges, risks, and future research direction. Section VII concludes with a synthesis of key findings and practical implications.

II. Research Methodology and Initial Analysis

This study systematically reviews recent publications in the field of open-source machine learning (ML) tools. We investigate open-source ML tools across different stages of the ML lifecycle, including problem framing, data collection and pre-processing, feature engineering, model development and training, deployment, monitoring, and governance. The investigation draws on sources from prominent academic research platforms and technical repositories, including ACM Digital Library, Elsevier, Google Scholar, IEEE Xplore, SpringerLink, and GitHub. All relevant publications and documentation available up to June 2025 were reviewed, with a primary focus on studies published from 2021 onward.

To ensure comprehensive coverage of the domain, two categories of keywords were used. The first set, including "ML lifecycle", "machine learning lifecycle", "ML lifecycle taxonomy", and "ML system engineering" was used to establish the scope of the overall investigation. The second set, comprising "open source ML tools", "data lifecycle open source tools", "deep learning open source tools", "development open source tools", "model deployment open source tools", "MLOps and workflow open source tools", "model serving and integration open source tools", "monitoring and governance open source tools" was used to refine the search toward specific lifecycle stages and functional tool classes. In the context of production deployment and trustworthiness, additional qualifiers were incorporated, including "privacy in machine learning", "supply-chain security", "fairness assessment", and "regulatory compliance".

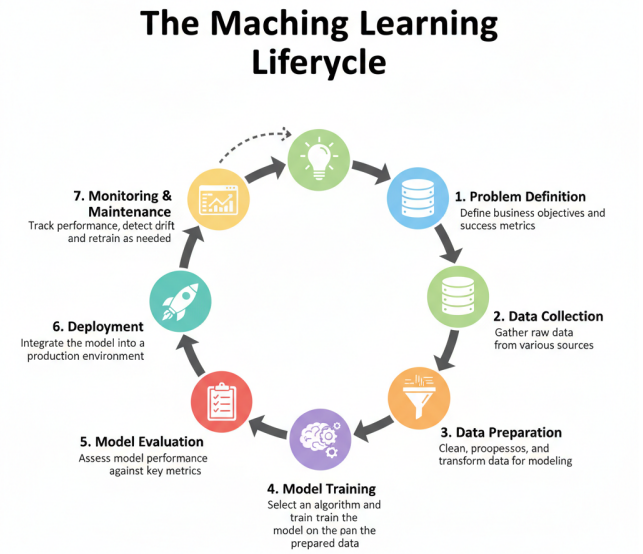

Fig. 1. Machine learning lifecycle.

At the beginning of the review process, an initial pool of research articles was identified based on relevance to the broad domain of open-source ML tools and frameworks. Following a rigorous screening process involving duplicate removal, abstract filtering, and full-text assessment, few publications were identified as directly aligned with the scope of this study.

III. Machine Learning Lifecycle

To provide a comprehensive framework for analyzing open-source tools, this section will explain how the machine learning lifecycle works in various stages. The machine learning lifecycle involves several stages which are part of the systematic process in the development of the models [21]. A brief analysis of the machine learning lifecycle along with the evaluation of current applications of machine learning algorithms are described [22]. Fig. 1 presents the machine learning lifecycle.

A. Problem Framing

At the core level, the machine learning lifecycle starts with identifying a problem. In this phase, a problem is identified and its structure is defined [23]. It formulates objectives and performance metrics that ensure consistency between model goal and the company's objectives based on availability of data and resources. This is a joint effort between business experts and data scientists to determine the expected outcomes [24]. Ethical factors such as fairness, harms, and threats are very important and need to be taken into consideration at early stages because they greatly affect data collection, model creation and implementation subsequently [21]. The criteria for success should be more than accuracy and extend to robustness and reliability as well.

B. Data Collection

This stage focuses on acquiring raw data that are used for model training [24]. It involves collecting data samples

through observation and measurement of the real-world system, process, or phenomenon for which an ML model needs to be built [25], [26]. It is critical in the lifecycle of machine learning models, as the quality and relevance of the gathered information directly influence the model performance.

Data can be collected from a range of internal sources, such as organizational databases, logs, and historical records, as well as external sources such as open-source platforms like Kaggle or the UCI Machine Learning Repository, social media streams for sentiment analysis, and market research studies. It is important to ensure that the selected data are relevant to the defined problem. Data may take various forms, including structured formats stored in relational databases, unstructured content such as text, images, or audio, and semi-structured formats such as JSON or XML. It is also important to determine the appropriate amount of data, balancing volume and quality to avoid issues like under- or overfitting while considering computational constraints [21]. The quality and relevance of collected data is equally critical, as these factors directly impact model performance and validity of the predictions.

C. Data Preparation

Data preparation is an important stage in the machine learning lifecycle. At this stage, the raw data is transformed into a clean, organized and suitable format for model training and evaluation [27], [28], [29]. The data cleaning involves handling missing values through deletion or imputation, identifying and managing outliers using statistical techniques and removing duplicates to maintain data integrity. Once the dataset is clean, data transformation techniques are applied. The data transformation techniques may include normalization, standardization to scale numerical features appropriately and encoding methods (one-hot or label encoding) to convert categorical data into machine-readable formats. In data splitting, dataset is partitioned into training, validation and testing sets. It helps in model evaluation and prevents overfitting. The advanced data engineering practices, such as pre-processing have streamlined the exploratory data analysis (EDA) and data preparation stages [30].

D. Feature Engineering

Feature engineering plays a significant role in improving model performance by creating new meaningful features. It may involve mathematical operations and the selection of only the most relevant features using methods such as recursive feature elimination, statistical tests, or feature importance measures [31], [32]. Proper data splitting is also essential, dividing the dataset into training, validation and test sets to support model training, hyperparameter tuning, and unbiased performance evaluation. In cases involving image, text, or other unstructured data, data augmentation techniques can be applied to expand the training data set artificially through modifications such as rotations, noise addition, or text perturbations or to further add data samples to the dataset [34], [33].

Data normalization and feature scaling techniques may also be applied to ensure that data adhere to a uniform scale and facilitate the learning process [24].

E. Model Selection, Training and Optimization

The selection of an appropriate ML model depends on various factors, including the nature of the problem (classification, regression, etc.), the characteristics of the data (structured, unstructured), and the available computational resources. Popular choices include decision trees, support vector machines, neural networks, and ensemble methods [24].

Model training plays a pivotal role in the machine learning lifecycle. In this phase, the selected algorithm learns patterns from the prepared data set by adjusting its parameters to reduce prediction errors. Performance metrics such as loss and accuracy are monitored throughout to ensure proper convergence and detect issues such as overfitting. Hyper-parameters are high-level configuration settings of the ML algorithm that can significantly impact its learning capacity. In addition to this, key hyper-parameters such as the learning rate, the number of training epochs, and the batch size are configured to control the model's learning process. Optimization algorithms such as grid search, random search, bayesian optimization, stochastic gradient descent, or Adam iteratively update the model parameters to minimize loss [24].

F. Model Evaluation and Validation

After training, the model is validated on a separate validation set to evaluate its generalization capability and tune hyper-parameters if required [35], [36]. It ensures that the model generalizes well beyond the training dataset and meets the intended objectives. The process begins with the selection of appropriate evaluation metrics based on the type of problem [24]. Once metrics are defined, the model is assessed using a validation set.

Model evaluation involves checking for overfitting or underfitting by comparing training and validation performance. Overfitting occurs when the model performs well in training data but poorly in validation data, while underfitting indicates that the model is too simple to capture underlying patterns. Sensitivity analysis can be conducted to study how variations in input features influence predictions, revealing the model robustness and identifying critical features. By systematically evaluating the model, practitioners ensure that its performance is reliable, generalizes effectively and is suitable for real-world deployment.

G. Model Deployment

The deployment of the model is the decisive stage of the machine learning lifecycle, where the trained model is integrated into a production environment to generate predictions on new real-world data [24]. This phase ensures that the model operates efficiently, reliably and securely. The process begins with selecting an appropriate deployment method, such as batch processing for scheduled or real-time inference for applications that require immediate responses through APIs. The deployment environment may involve cloud platforms like AWS, Azure, or Google Cloud, chosen for their scalability, services and global reach. The model is then serialized into a suitable format, such as Pickle or TensorFlow, so that it can be easily loaded and executed in production pipeline. If the model needs to be accessed by external applications, an API is developed using frameworks like FastAPI or Flask. Once deployed,

continuous monitoring becomes critical to track performance metrics such as prediction accuracy, latency and resource usage. A structured plan for model updates helps maintain long-term performance by incorporating retraining schedules, version control and user feedback. The post-deployment performance evaluation ensures that the model continues to meet business goals.

H. Monitoring and Maintenance

Monitoring and maintenance are critical stages of the machine learning lifecycle, ensuring that the models remain reliable, accurate and aligned with real-world requirements. Once a model is operational, continuous performance monitoring becomes essential to detect degradation timely. This includes tracking key metrics (accuracy and precision) and system-level indicators (latency and throughput). The detection of data drifts is equally important, as it can significantly impact predictions. Logging and auditing play an important role in maintaining transparency and accountability by recording inputs, outputs, errors and model updates. User feedback also contributes to continuous improvement, offering practical insights into the behavior of the real-world model. In addition, regular reviews help address technical debt by refactoring code, updating dependencies and improving infrastructure reliability. The documentation must be kept up-to-date to reflect model updates, deployment changes and operational guidelines. The techniques such as model explainability, bias mitigation and transparency provide insight into model behavior, allowing stakeholders to assess model performance and compliance with regulatory and ethical standards [30]. Finally, a well-defined incident response plan enables teams to quickly mitigate issues related to performance or security breaches. Together, these monitoring and maintenance practices create a proactive framework that keeps machine learning models reliable and effective throughout their lifecycle.

IV. Lifecycle-Oriented Taxonomy of Open-Source Machine Learning Tools

The existence of a variety of open source machine learning tools enables a reliable, scalable, and reproducible data workflow throughout the machine learning pipeline. A structured taxonomy for machine learning experiment tracking is introduced and applied to widely used tools [37]. A comprehensive taxonomy of open source machine learning tools is presented in Fig. 2. It includes open source machine learning tools for data engineering, deep learning, model deployment and optimization, workflow orchestration, model serving and system integration, monitoring and governance, development, visualization and knowledge management.

A. Data Lifecycle

Data operations in machine learning pipelines include ingestion, distributed computing, quality and validation, lineage and governance, labeling and annotation, feature engineering, data augmentation, and privacy and compliance. Table I provides a summary of these operations.

Open source tools for data ingestion include Pandas, Dask, and Apache Spark. Pandas, for instance, is highly effective for in-memory processing of small to medium-sized datasets, while Dask and Apache Spark are designed for larger distributed datasets. These tools help to perform data preprocessing, loading, and transforming in large datasets. For computationally intensive ML operations, data are needed. can be computed in distributed or parallel fashion using tools e.g. Ray, Spark, and Dask. To ensure data quality by automating data profiling, validation and anomaly detection tools such as Great Expectations and Deepchecks can be helpful. For metadata management and lineage tracking tools, such as DVC, lakeFS, Pachyderm, OpenLineage/Marquez, and DataHub, enhance reproducibility and data control. Some supervised learning tasks require data labeling and annotation. For this purpose, the Label Studio and CVAT tools streamline the dataset labeling process. To automate feature extraction and centralized feature management, Featuretools and Feast can help in feature operationalization. In some scenarios, the data set available is insufficient for model training, data augmentation is used to create synthetic datasets to improve model robustness. For this purpose, tools such as SDV and Albumentations can be helpful. For data privacy and compliance libraries including Diffprivlib, Opacus, and TensorFlow Privacy, the library supports modern security and compliance standards. Together, these tools form a comprehensive foundation for handling data in machine learning pipelines.

B. Deep Learning

In the machine learning pipeline, a wide range of open-source libraries and frameworks are available to support deep learning development. The general purpose deep learning pipelines include scikit-learn, XGBoost, LightGBM, and CatBoost are helpful in ensemble methods, gradient boosting and supervised or unsupervised learning. These libraries are reliable for an efficient and scalable implementation of various applications. Neural networks are the primary foundation for deep learning. There are open source frameworks to facilitate their design, training, and deployment by supporting flexible model architectures, GPU acceleration, and automatic differentiation. It includes PyTorch, TensorFlow, JAX, and Keras. Similarly, for probabilistic models, libraries like Pyro, NumPyro, Stan, and TensorFlow Probability help in advanced statistical modeling and inference. There are many toolkits available for specialized domain specific tasks. The libraries such as PyTorch Geometric and DGL supports graph based frameworks. For time series libraries, darts, GluonTS, and Kats are helpful for sequential data analysis. To automate feature selection and optimize hyper-parameters(HPO) tools such as Auto-sklearn, AutoGluon, H2O-3 AutoML, TPOT, Optuna, and Ray Tune are helpful in feature engineering and parameter tuning. Collectively, using these resources, efficient models can be implemented across diverse ML workflows. Table II provides a summary of these tools.

C. Development and Visualization

The development and communication in machine learning workflows rely on versatile tools for coding, visualization, and documentation. Open source tools such as interactive notebooks and integrated development environments provide flexible platforms for experimentation, debugging, and iterative model development. Open source visualization libraries provide clear and insightful representation of data and model

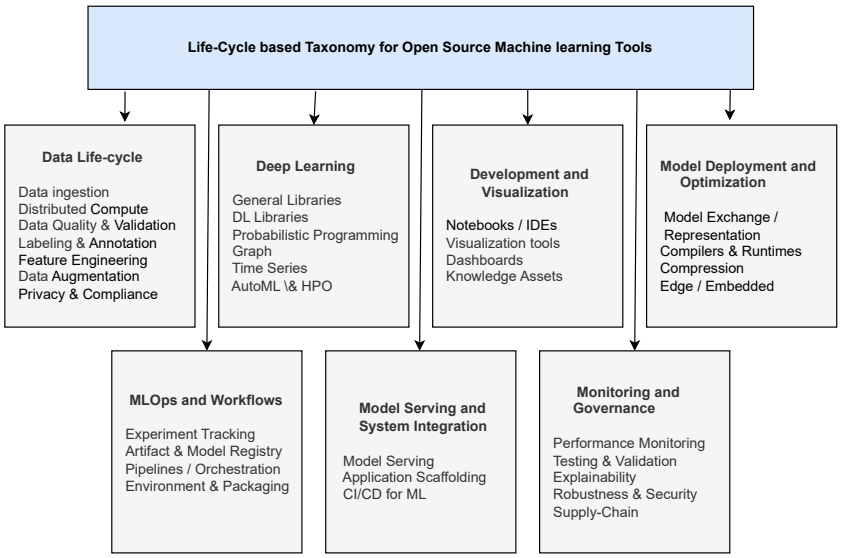

Fig. 2. Machine learning lifecycle-based taxonomy for open source ML tools.

TABLE I. OPEN-SOURCE ML TOOLS FOR DATA LIFECYCLE

Category	Tools	Languages	Purpose	Key Features
Data ingestion	Pandas [38], Dask [39], Apache Spark [40]	Python; Python; Python/Scala/Java	Data loading, preprocessing, transformation	Scalable ETL, distributed dataframes
Distributed Compute	Ray [41], Spark, Dask	Python; Python/Scala/Java; Python	Parallel and distributed execution for ML pipelines	Task scheduling, cluster management, scalable computation
Data Quality & Validation	Great Expectations [42], Deepchecks [43]	Python	Data profiling, validation, integrity checks	Automated tests, pipeline integration, anomaly detection
Data Lineage & Governance	DVC [44], lakeFS [45], Pachyderm [46], Marquez [47], DataHub [48]	Python; Python/Go; Go/Python; Python/Java; Java/Python	Dataset versioning, lineage tracking, metadata management	Git-like data control, governance
Labeling & Annotation	Label Studio [49], CVAT [50]	Python/JS	Creating and managing labeled datasets	Multi-format annotation, collaborative tools, automation support
Feature Engineering	Featuretools [51], Feast [52]	Python; Python/Go	Feature creation, storage, and serving	Automated feature engineering, centralized feature repository
Data Augmentation	SDV [53], Albumentations [54]	Python	Generating synthetic datasets	Advanced image augmentation
Privacy & Compliance	Diffprivlib [55], Opacus [56], TensorFlow Privacy [57]	Python	Differential privacy and secure ML training	Privacy-preserving training, noise mechanisms

outputs. JupyterLab and the open-source VS Code are examples of such platforms, whose visualization libraries include Matplotlib, Seaborn, Altair, and the open-source core of Plotly.

To share results with stakeholders, frameworks like Streamlit, Panel, and Dash help in the deployment of user-friendly interfaces, and rapid prototyping. Additionally, knowledge asset frameworks facilitate systematic documentation of models and datasets, promoting reproducibility and informed decision-making throughout the machine learning lifecycle. Such frameworks include Model Cards Toolkit and Datasheet tooling. Table III provides a summary of these tools.

D. Model Deployment and Optimization

The deployment and optimization of machine learning models have been facilitated especially in model representation, compilation, run-time , compression, and deployment. Models trained in PyTorch, TensorFlow, or other platforms can be seamlessly executed in other environments. For this purpose, ONNX, enable interoperability across different frameworks. Machine learning models can be compiled and optimized using hardware acceleration. For this purpose, open source tools include TVM, IREE, XLA, OpenVINO, and ONNX Runtime. For model compression, Intel Neural compression, NNI, and SparseML help reduce model size and latency using pruning, quantization, and other techniques. The embedded applications require low latency execution on smart devices and microcontrollers. Lightweight inference engines such as TensorFlow Lite, TFLite Micro, and ONNX Runtime Mobile are a big support for this purpose. These tools help to secure, portable, and scalable model deployment in the ML pipeline. Table IV provides a summary of these tools.

TABLE II. OPEN-SOURCE ML TOOLS FOR DEEP LEARNING

Category	Tools	Languages	Purpose	Key Features
General Libraries	scikit-learn [58], XGBoost[59], LightGBM [60], CatBoost [61]	Python	Supervised/unsupervised ML, boosting gradient, ensemble methods	Easy-to-use APIs, high performance, scalable
DL Libraries	PyTorch [62], TensorFlow [57], JAX [63], Keras [64]	Python	Neural network design and training	GPU acceleration, automatic differentiation, flexible model building
Probabilistic Programming	Pyro [65], NumPyro [66], Stan [67], TensorFlow Probability [68]	Python; Python; Stan; Python	Bayesian modeling, probabilistic inference	Efficient sampling, variational inference, uncertainty quantification
Graph	PyTorch Geometric [69], DGL [70]	Python	Graph neural networks and structured data modeling	Message passing, graph convolutions, scalable GNN
Time Series	darts [71], GluonTS [72], Kats [73]	Python	Forecasting, time-series analysis	Forecasting, anomaly detection
AutoML & HPO	Auto-sklearn [74], AutoGluon [75], H2O-3 AutoML [76], TPOT [77], Optuna [78], Ray Tune [79]	Python; Python; Java/Python; Python; Python; Python	Model selection, hyperparameter optimization	Feature selection, ensemble building, distributed optimization

TABLE III. OPEN-SOURCE ML TOOLS FOR DEVELOPMENT AND VISUALIZATION

Category	Tools	Languages	Purpose	Key Features
Notebooks / IDEs	JupyterLab [80], VS Code (OSS) [81]	Python; Multiple	Interactive development and code editing	Rich IDE, debugging, plugins, notebook
Visualization	Matplotlib [82], Seaborn [83], Altair [84], Plotly (OSS core) [85]	Python	Create static and interactive data visualizations	Plotting, interactive charts, statistical visualization, figures
Dashboards	Streamlit [86], Panel [87], Dash [88]	Python	Build interactive dashboards and data-driven applications	Rapid prototyping, Visualization, low-code development
Knowledge Assets	Model Cards Toolkit [89], Datasheets tooling [89]	Python	Document model details and dataset characteristics	Standardized metadata capture, transparency, governance support

TABLE IV. OPEN-SOURCE MODEL DEPLOYMENT AND OPTIMIZATION TOOLS

Category	Tools	Languages	Purpose	Key Features
Model Exchange / Representation	ONNX [90]	Python, C++	Cross-platform deployment	Interoperability, model format standardization, framework-agnostic conversion
Compilers & Runtimes	TVM [91], IREE [92], XLA [93], OpenVINO [94], ONNX Runtime [90]	Python, C++, MLIR	Optimized compilation and runtime execution	Graph-level optimizations, target-specific compilation, hardware acceleration
Compression	Intel Neural Compressor [90], NNI [95], SparseML [96]	Python	Quantization, pruning, and compression	Reduced model size and latency, hardware-aware optimization
Edge / Embedded	TensorFlow Lite [97], TFLite Micro [98], ONNX Runtime Mobile [90]	Python, C++	Deployment on resource-constrained devices	Low-latency execution, mobile and microcontroller support

E. MLOps and Workflows

In the deployment of machine learning models, the use of robust tools and frameworks is of primary importance to ensure efficient management of data, models, and computational workflows. The open source experiment tracking platforms include MLflow, Aim, Sacred, and Guild AI. These platforms enable systematic comparison and iterative improvements. It also provides comprehensive mechanisms to log hyperparameters, metrics, and results. To facilitate collaboration, artifact and model registries can store different versions of trained models, datasets, and other resources. Open source tools include MLflow Model Registry, ModelDB, and DVC in this domain. Efficient environment management and packaging are very helpful in enabling reproducible computational environments and dependency management across different systems. For this purpose, tools such as Conda, Mamba, Poetry, pip-tools, Docker, and Nix can be a game changer. Automated execution of sequential and parallel tasks can be streamlined with orchestration. Such frameworks include Apache Airflow, Argo Workflows, Kubeflow Pipelines, Dagster, and Metaflow. These tools enhance the scalability and reproducibility of machine learning production pipelines. Table V provides a summary of these tools.

F. Model Serving and Integration

The reliable model serving and integration frameworks are important for translating models into commercial applications. There are many open source tools to support high performance solutions to expose trained models as APIs to support real-time and batch inference. Such platforms include KServe, Seldon

TABLE V. OPEN-SOURCE MLOPS AND WORKFLOW MANAGEMENT TOOLS

Category	Tools	Languages	Purpose	Key Features
Experiment Tracking	MLflow [99], Aim [100], Sacred [101], Guild AI [102]	Python	Track experiments, log metrics and parameters	Versioned results, reproducibility, visualization
Artifact & Model Registry	MLflow Model Registry [99], ModelDB [103], DVC artifacts [104]	Python	Store and manage models	Model versioning, metadata management, pipeline integration
Environment & Packaging	Conda/Mamba [105], Poetry/pip-tools [106], Docker [107], Nix [108]	Python, Shell	Manage dependencies and reproducible environments	Dependency resolution, environment isolation
Pipelines / Orchestration	Apache Airflow [109], Argo Workflows [110], Kubeflow Pipelines [111], Dagster [112], Metaflow [113]	Python, YAML, Shell	Design, schedule, and orchestrate ML pipelines	Workflow management, task scheduling, scalable execution

TABLE VI. OPEN-SOURCE MODEL SERVING AND SYSTEM INTEGRATION TOOLS

Category	Tools	Languages	Purpose	Key Features
Model Serving	KServe [114], Seldon Core [115], TorchServe [116], BentoML [117], NVIDIA Triton Inference Server [118]	Python, C++	Deploy trained ML models for production inference	Scalable serving, standardized APIs, GPU acceleration
Application Scaffolding	FastAPI [119], gRPC [120], Streamlit [86], Dash [88]	Python	Build lightweight applications and APIs	Interactive dashboards, rapid prototyping, low-latency interfaces
CI/CD for ML	Tekton [121], Argo CD [122], Jenkins [123]	YAML, Python, Shell	Automate testing, integration, and deployment	Pipeline automation, reproducibility, cloud-native integration

TABLE VII. OPEN-SOURCE MONITORING AND GOVERNANCE TOOLS

Category	Tools	Languages	Purpose	Key Features
Performance Monitoring	Evidently AI [124], whylogs [125], Prometheus [126] + Grafana [127]	Python; Python; Python/Go	Track performance and detect data drift	Real-time monitoring, visualization, anomaly detection
Testing & Validation	Great Expectations [42], Deepchecks [43]	Python	Validate datasets and models for correctness	Data profiling, model checks, pipeline integration, anomaly detection
Explainability	SHAP [128], LIME [129], Captum [130], Fairlearn [131], AIF360 [132]	Python	Model interpretability and fairness assessment	Feature attribution, bias detection, fairness metrics
Robustness & Security	Adversarial Robustness Toolbox (ART) [133]	Python	Evaluate and improve model robustness	Attack simulation, defense mechanisms, threat modeling
Supply-Chain	Sigstore/cosign [134], Syft/Grype [135], [136], pip-audit [137], in-toto [138], SBOMs	Python; Go; Shell	Ensure security and integrity	Artifact verification, provenance tracking, vulnerability scanning

Core, TorchServe, BentoML, and NVIDIA Triton Inference Server. The application sacffolding tools help in the rapid development of interactive applications that integrate seamlessly with different ML models. The application scaffolding tools include FastAPI and gRPC. The open source lightweight user interface frameworks such as Streamlit or Dash support low latency. For smooth and robust delivery in production pipelines, CI/CD tools facilitate reduced operational overhead and reliable updates. These tools include Tekton, Argo CD, and Jenkins to automate testing, deployment, and monitoring processes. Together, these tools form a cohesive stack for efficient model operations in various production environments. Table VI provides a summary of these tools.

G. Monitoring and Governance

A comprehensive set of monitoring and governance tools is needed to ensure fairness in machine learning systems deployed. To monitor the performance of the model, it is necessary to keep track of the model output, data distribution, and system health to facilitate early detection of performance degradation. The frameworks include Evidently AI, whylogs, and the Prometheus–Grafana. For rigorous testing and validation, platforms like Great Expectations and Deepchecks automate checks on model behavior and data quality. To evaluate model predictions and mitigate bias, ensure ethical and transparent AI, tools such as SHAP, LIME, Captum, Fairlearn, and AIF360 are very helpful. To ensure protection and security of machine learning systems, the Adversarial Robustness Toolbox

(ART) allows the assessment of vulnerabilities to adversarial attacks.

Finally, supply-chain integrity is maintained by verifying dependencies, packaging, and traceability throughout the ML lifecycle. the tools for this phase include Sigstore, Cosign, Syft, Grype, pip-audit, in-toto, and Software Bill of Materials (SBOMs). These tools establish a strong foundation for trustworthy and secure AI deployments. Table VII provides a summary of these tools.

V. Comparative Analysis and Discussion

The landscape of tools and frameworks supporting machine learning is expanding rapidly, new frameworks are emerging, and existing ones are evolving rapidly. This constant growth requires researchers and developers to stay upto date on these resources and use them to efficiently execute their tasks.

An effective data lifecycle forms the backbone of machine learning pipelines. The Pandas library works well for small to medium datasets due to its in-memory processing. Dask extends the Pandas interface to distributed environments and parallel computations. It may require tuning for large heterogeneous clusters. In contrast, Apache Spark is suitable for large-scale workloads, but it is less flexible for parallel execution compared to Dask. Ray also helps scale workloads for reinforcement learning by offering distributed computation capabilities that handle complex and iterative tasks. DVC and lakeFS are ideal for reproducible experiments because of their Git-like versioning, whereas Pachyderm offers enterprise-grade reproducible pipelines. Label Studio supports multi-modal annotation for general ML applications, whereas CVAT excels with advanced video annotation capabilities. Feast serves as a feature store in production systems. The features can be retrieved consistently. Featuretools is more research-oriented than Feast. Opacus integrates privacy concerns for training machine learning algorithms, and TensorFlow Privacy supports similar capabilities for TensorFlow models. Great Expectations validates with automated documentation, whereas Deepchecks extends validation to for drift and model performance using its error analysis.

For deep learning, general-purpose libraries such as scikit-learn provide broad algorithmic coverage, whereas XGBoost, LightGBM, and CatBoost perform well on structured data. Auto-sklearn and TPOT are valuable for automating feature engineering and model selection within scikit-learn pipelines, while AutoGluon and H2O-3 AutoML extend this automation to deep learning frameworks. The dynamic interface of PyTorch makes it preferred for research and experimentation, while TensorFlow supports production-grade. Domain-specific libraries provide efficient message-passing primitives and extensive model repositories (PyG, DGL). Time-series libraries offer unified interfaces for anomaly detection and probabilistic predictions using tools such as Pyro and NumPyro. Stan is well- known for statistical inference but it is less suited for deep neural architectures. Keras emphasizes rapid prototyping and is often used as an accessible front-end for TensorFlow-backed training.

For development and visualization, JupyterLab excels in exploratory data analysis with a user-friendly interface, whereas VS Code provides a more integrated development environment with advanced debugging, version control, and multi-language support. Visualization libraries such as Matplotlib offer fine-grained control for static plots but require more boilerplate, while Seaborn provides higher-level statistical visualizations with minimal code. Dashboards like Panel offer greater flexibility by supporting multiple plotting backends, while Dash provides support for enterprise level analytics for web applications.

Model deployment and optimization tools perform low-level graph optimization, kernel fusion, and hardware code generation. These tools include TVM and IREE which excel in automated operator tuning and broad hardware support, while XLA offers tight integration with TensorFlow and JAX for ahead-of-time compilation. OpenVINO is optimized for Intel architectures, providing high-performance inference on CPU and VPUs, while ONNX Runtime delivers portable high-speed inference across GPU, CPU, and edge devices through execution providers.

Experiment tracking tools emphasize lightweight metrics tracking and support reproducibility during model development. These tools include MLflow, Aim, Sacred, and Guild AI. In contrast, artifact and model registries extend this functionality by enabling version control, lineage tracking, and governance of models and datasets. Environment and packaging tools such as Conda/Mamba, Poetry, pip-tools, Docker, and Nix address key sources of inconsistencies between development and production systems. For instance, MLflow offers a comprehensive platform for both experiment tracking and model registry, making it suitable for end-to-end MLOps, whereas tools like DVC focus specifically on data and model versioning for reproducibility.

Model serving frameworks are primarily designed to support low-latency inference with hardware acceleration. TorchServe and BentoML also offer strong integration with PyTorch-centric workflows, Triton is optimized for high-throughput and multi-framework GPU inference for large-scale productions. Application scaffolding tools, including FastAPI, gRPC, Streamlit, and Dash, focus on exposing trained models through interactive interfaces for rapid prototyping and visualization. CI/CD tools for machine learning, such as Tekton, Argo CD, and Jenkins, automate testing, integration, and pipeline deployments. However, these tools differ in their native support for GitOps practices, Kubernetes orchestration, and ML-specific versioning. The toolchain must be carefully composed to achieve reproducible end-to-end deployments.

Performance monitoring tools enable continuous monitoring of system behavior with early detection of model degradation. Clearly, AI, whylogs, and Prometheus with Grafana are examples of such tools. Testing and validation frameworks, including Great Expectations and Deepchecks, emphasize pre-deployment, distributional consistency, and failure diagnostics. Explainability tools such as SHAP, LIME, Captum, and Fairlearn address the growing demand for interpretable and fair AI by providing bias assessment across diverse model architectures. Finally, supply-chain security tools such as Sigstore/cosign, Syft/Grype, pip-audit, in-toto, and SBOMs extend governance to software integrity and dependency transparency.

VI. Key Challenges and Future Perspectives

The machine learning lifecycle presents several key challenges related to each stage. Large or heterogeneous datasets present scalability and performance constraints, often due to limitations in memory management, efficient distributed processing, or data transfer speeds within open-source frameworks. Distributed computing frameworks often differ in execution paradigms, which complicate overall workflow. Feature consistency and low-latency service between research and production environments presents a real challenge. Furthermore, effective integration of privacy preservation mechanisms and adherence to regulatory requirements remains a struggle due to the heterogeneity of compliance standards.

Deep learning and machine learning workflows face several interrelated challenges. The frameworks optimized for flexibility often lack seamless integration into scalable production systems. The support of high-performance computing requires careful consideration of computational graphs, compilation strategies, and parallelism. Domain-specific modeling, such as graph-based or temporal data, introduces additional complexity in the efficient processing of structured and sequential data. Automated machine learning and feature engineering also face limitations in generalization and robustness when working with diverse data types.

Development and visualization workflows require support for interactive, iterative, and integrated workflows. Visualization tools should balance control and ease of use for enterprise-ready deployment. These factors highlight the difficulty of designing tools that are user-friendly for experimentation and robust for production analytics.

High performance inference requires careful low-level optimization, hardware-aware code generation, and efficient execution across diverse platforms. Balancing portability, speed, and compatibility with different hardware architectures (CPUs, GPUs, and edge devices) adds additional complexity. Integrating these optimizations into existing workflows while maintaining reproducibility and scalability remains a challenge in model deployment.

Despite the capabilities offered by the MLOps, several critical challenges persist due to the fragmented and heterogeneous nature of the machine learning architectures. The lack of a unified end-to-end platform requires complex integration across experiment tracking, model registries, and orchestration tools. Increase system complexity and operational overhead by demanding extensive configuration. The scalability and deployment issues in cloud interoperability also complicate the transition from research to production. The synchronization and governance of metadata across loosely coupled components often leads to gaps in traceability and compliance. Additionally, while packaging tools improve reproducibility, inconsistencies across operating systems and hardware accelerators introduce risks in large-scale deployments.

The heterogeneous protocol and hardware acceleration in diverse frameworks introduce deployment complexity and hamper portability across infrastructures. production-grade frameworks and rapid prototyping tools create friction when transitioning experimental prototypes to scalable services. CI/CD platforms such as Tekton, Argo CD, and Jenkins enable automated deployment, but their limited support for versioning, model validation, and data testing restricts full lifecycle automation. In addition, the absence of standardized interfaces across application layers and pipelines complicates rollback and reproducibility, for example, requiring manual intervention for data schema changes or model version rollbacks.

Fragmented coverage in performance monitoring, data validation, explainability, robustness, and supply-chain security requires the integration of multiple specialized tools, substantially increasing system complexity and maintenance overhead. The heterogeneous coverage of performance monitoring, data validation, and explainability requires the integration of multiple tools. Consequently, it increases system complexity, operational burden, and long-term maintenance overhead. The limited standardization of explainability and fairness assessments severely restricts cross-tool interoperability across the MLOps toolchain. Large-scale productions remain computationally expensive and often conflict with real-time latency requirements. Additionally, while supply-chain security tools enhance software integrity, their integration into ML-specific pipelines is still emerging, leaving critical gaps in compliance enforcement.

VII. Conclusion

This study bridges the gap between tool-centric surveys and practical requirements of end-to-end ML system engineering by providing a systematic, lifecycle-oriented taxonomy of open-source tools and discussing their practical implications. In this study, we present a comprehensive, lifecycle-oriented taxonomy of open-source machine learning tools. The proposed taxonomy systematically organizes the evolving ML tools across all major stages of the machine learning lifecycle, including data collection and preparation, feature engineering, model development, training and evaluation, deployment, monitoring, and governance. It enables researchers to systematically analyze emerging tools while providing a guideline for designers to develop scalable, reliable, and trustworthy ML systems.

Open source tools offer strong specialization in areas such as experiment tracking, orchestration, model serving, and monitoring. However, achieving fully automated, reproducible, and compliant ML pipelines still continues to require substantial cross-tool integration and engineering effort. There are persistent challenges related to interoperability, standardization, production, real-time monitoring, explainability, and supply-chain security. There is a need for standardization of the interface, integration between lifecycle stages, and security mechanisms in ML pipelines. Addressing these challenges will require collaborative efforts towards developing common APIs, robust data contracts, and standardized security protocols across the open-source ecosystem. Future research directions include the development of unified ML lifecycle platforms for seamless integration across all stages, benchmark-driven comparative evaluations of tool performance to provide evidence for tool selection, and empirical studies on large-scale industrial deployments, or validate theoretical frameworks against real-world challenges.

REFERENCES

[1] Nasir, V., and Sassani, F., "A Review on Deep Learning in Machining and Tool Monitoring: Methods, Opportunities, and Challenges," *The International Journal of Advanced Manufacturing Technology*, vol. 115, no. 9, pp. 2683–2709, 2021.

[2] Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., and Cortez, P., "A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost," in *Proc. International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[3] Sharifani, K., and Amini, M., "Machine Learning and Deep Learning: A Review of Methods and Applications," *World Information Technology and Engineering Journal*, vol. 10, no. 07, pp. 3897–3904, 2023.

[4] Chiche, A., and Yitagesu, B., "Part of Speech Tagging: A Systematic Review of Deep Learning and Machine Learning Approaches," *Journal of Big Data*, vol. 9, no. 1, pp. 10, 2022.

[5] Taye, M. M., "Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions," *Computers*, vol. 12, no. 5, pp. 91, 2023.

[6] Geetha, R., and Thilagam, T., "A Review on the Effectiveness of Machine Learning and Deep Learning Algorithms for Cyber Security," *Archives of Computational Methods in Engineering*, vol. 28, no. 4, pp. 2861–2879, 2021.

[7] Kaluarachchi, T., Reis, A., and Nanayakkara, S., "A Review of Recent Deep Learning Approaches in Human-Centered Machine Learning," *Sensors*, vol. 21, no. 7, pp. 2514, 2021.

[8] Soori, M., Arezoo, B., and Dastres, R., "Artificial Intelligence, Machine Learning and Deep Learning in Advanced Robotics: A Review," *Cognitive Robotics*, 2023.

[9] Fregoso-Aparicio, L., Noguez, J., Montesinos, L., and García-García, J. A., "Machine Learning and Deep Learning Predictive Models for Type 2 Diabetes: A Systematic Review," *Diabetology & Metabolic Syndrome*, vol. 13, no. 1, pp. 148, 2021.

[10] Hopkins, E., "Machine Learning Tools, Algorithms, and Techniques," *Journal of Self-Governance and Management Economics*, vol. 10, no. 1, pp. 43–55, 2022.

[11] Xu, Y., Zhou, Y., Sekula, P., and Ding, L., "Machine Learning in Construction: From Shallow to Deep Learning," *Developments in the Built Environment*, vol. 6, pp. 100045, 2021.

[12] Mijwil, M., Salem, I. E., and Ismaeel, M. M., "The Significance of Machine Learning and Deep Learning Techniques in Cybersecurity: A Comprehensive Review," *Iraqi Journal for Computer Science and Mathematics*, vol. 4, no. 1, pp. 87–101, 2023.

[13] Janiesch, C., Zschech, P., and Heinrich, K., "Machine Learning and Deep Learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.

[14] Al-amri, R., Murugesan, R. K., Man, M., Abdulateef, A. F., Al-Sharafi, M. A., and Alkahtani, A. A., "A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data," *Applied Sciences*, vol. 11, no. 12, pp. 5320, 2021.

[15] Forootan, M. M., Larki, I., Zahedi, R., and Ahmadi, A., "Machine Learning and Deep Learning in Energy Systems: A Review," *Sustainability*, vol. 14, no. 8, pp. 4832, 2022.

[16] Avci, O., Abdeljaber, O., Kiranyaz, S., Hussein, M., Gabbouj, M., and Inman, D. J., "A Review of Vibration-Based Damage Detection in Civil Structures: From Traditional Methods to Machine Learning and Deep Learning Applications," *Mechanical Systems and Signal Processing*, vol. 147, pp. 107077, 2021.

[17] Moein, M. M., Saradar, A., Rahmati, K., Mousavinejad, S. H. G., Bristow, J., Aramali, V., and Karakouzian, M., "Predictive Models for Concrete Properties Using Machine Learning and Deep Learning Approaches: A Review," *Journal of Building Engineering*, vol. 63, pp. 105444, 2023.

[18] Jayatilake, S. M. D. A. C., and Ganegoda, G. U., "Involvement of Machine Learning Tools in Healthcare Decision Making," *Journal of Healthcare Engineering*, vol. 2021, no. 1, pp. 6679512, 2021.

[19] Kitchenham, B., Budgen, D., and Brereton, O., "Using Mapping Studies as the Basis for Further Research – A Participant-Observer Case Study," *Information and Software Technology*, vol. 53, no. 6, pp. 638–651, 2011.

[20] Petersen, K., Vakkalanka, S., and Kuzniarz, L., "Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[21] Ok, E., Winston, O., and James, L., "Lifecycle of Machine Learning Models," 2022.

[22] Devi, A., and Pavithra, K., "Machine Learning: Life Cycle and Its Techniques," *SSRN Electronic Journal*, 2022. DOI: 10.2139/ssrn.4140255.

[23] Hoerl, R., Kuonen, D., and Redman, T., "Problem Framing: Essential to Successful Statistical Engineering Applications," *Quality Engineering*, vol. 34, pp. 1–9, 2022. DOI: 10.1080/08982112.2022.2113098.

[24] Saini, V., Reddy, A. K., Venkataramanan, S., Chitta, S., and Bo, S., "Improving Algorithm Performance: Managing the Full Machine Learning Model Lifecycle," *Journal of Informatics Education and Research*, vol. 2, pp. 29–43, 2022.

[25] Géron, A., "Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems," O'Reilly Media, Inc., 2017.

[26] Wagstaff, K., "Machine Learning That Matters," arXiv:1206.4656, 2012. [Online]. Available: https://arxiv.org/abs/1206.4656

[27] Varshney, K. R., "Trustworthy Machine Learning and Artificial Intelligence," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 25, no. 3, pp. 26–29, 2019.

[28] Kotsiantis, S. B., Kanellopoulos, D., and Pintelas, P. E., "Data Preprocessing for Supervised Learning," *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[29] Zhang, S., Zhang, C., and Yang, Q., "Data Preparation for Data Mining," *Applied Artificial Intelligence*, vol. 17, no. 5–6, pp. 375–381, 2003.

[30] Jayabalan, D., and Indra, S., "Enhancing Machine Learning Life Cycle through Advanced Data Engineering," *International Journal of Computer Trends and Technology*, vol. 72, pp. 136–139, 2024. DOI: 10.14445/22312803/IJCTT-V72I4P117.

[31] Heaton, J., "An Empirical Analysis of Feature Engineering for Predictive Modeling," in *Proc. SoutheastCon*, IEEE, pp. 1–6, 2016.

[32] Khurana, U., Samulowitz, H., and Turaga, D., "Feature Engineering for Predictive Modeling Using Reinforcement Learning," in *Proc. AAAI Conference on Artificial Intelligence*, pp. 3407–3414, 2018.

[33] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M., "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3234–3243, 2016.

[34] Wong, S. C., Gatt, A., Stamatescu, V., and McDonnell, M. D., "Understanding Data Augmentation for Classification: When to Warp?" in *Proc. International Conference on Digital Image Computing: Techniques and Applications*, IEEE, pp. 1–6, 2016.

[35] Goodfellow, I., Bengio, Y., and Courville, A., "Deep Learning," MIT Press, 2016.

[36] Murphy, K. P., "Machine Learning: A Probabilistic Perspective," MIT Press, 2012.

[37] Quaranta, L., Calefato, F., and Lanubile, F., "A Taxonomy of Tools for Reproducible Machine Learning Experiments," in *Proc. AIxIA Discussion Papers Workshop (AIxIA DP)*, pp. 65–76, 2021. [Online]. Available: CEUR-WS.org/Vol-3078/paper-81.pdf

[38] Pandas-Python Data Analysis Library. (2026, Jan 21). https://pandas.pydata.org/

[39] Dask — Scale the Python tools you love. (n.d.). https://www.dask.org/

[40] PySpark Overview - PySpark 4.1.0 documentation. (2025, Dec 11). https://spark.apache.org/docs/latest/api/python/index.html

[41] ray. (2025, December 20). PyPI. https://pypi.org/project/ray/

[42] great-expectations. (2026, January 29). PyPI. https://pypi.org/project/great-expectations/

[43] Deepchecks AI. (2025b, February 5). DeepChecks Open-Source-Validating your ML models & data — DeepChecks. https://www.deepchecks.com/open-source/

[44] dvc. (2026, January 8). PyPI. https://pypi.org/project/dvc/

[45] lakefs. (2026, January 14). PyPI. https://pypi.org/project/lakefs/

[46] Pachyderm. (2024, Februrary 23). https://pypi.org/project/pachyderm/

[47] Marquez Project — Marquez Project. (n.d.). https://marquezproject.ai/

[48] DataHub. (2026, January 29). DataHub — Modern Data Catalog & Metadata Platform. https://datahub.com/

[49] Open Source data Labeling — Label Studio. (n.d.). Label Studio. https://labelstud.io/

[50] CVAT.ai. (n.d.). Leading data annotation platform for images, videos and 3D — CVAT. https://www.cvat.ai/

[51] What is Featuretools? - Featuretools 1.31.0 documentation. (n.d.). https://featuretools.alteryx.com/en/stable/

[52] Feast-the open source feature store for machine learning. (n.d.). Feast. https://feast.dev/

[53] Welcome to the SDV! — Synthetic Data Vault. (n.d.). Synthetic Data Vault. https://docs.sdv.dev/sdv

[54] Albumentations. (n.d.). Albumentations: fast and flexible image augmentations. https://albumentations.ai/

[55] diffprivlib. (2025, April 10). PyPI. https://pypi.org/project/diffprivlib/

[56] Opacus · Train PyTorch models with Differential Privacy. (n.d.). https://opacus.ai/

[57] TensorFlow. (n.d.). TensorFlow. https://www.tensorflow.org/

[58] scikit-learn: machine learning in Python - scikit-learn 0.16.1 documentation. (n.d.). https://scikit-learn.org/

[59] XGBoost. (n.d.). https://xgboost.ai/

[60] Welcome to LightGBM's documentation!-LightGBM 4.6.0 documentation. (n.d.). https://lightgbm.readthedocs.io/

[61] CatBoost-state-of-the-art open-source gradient boosting library with categorical features support. (n.d.). https://catboost.ai/

[62] PyTorch. (n.d.). PyTorch. https://pytorch.org/

[63] JAX: High performance array computing - JAX documentation. (n.d.). https://jax.readthedocs.io/

[64] Team, K. (n.d.). Keras: Deep Learning for humans. https://keras.io/

[65] Pyro. (n.d.). https://pyro.ai/

[66] NumPyro documentation- NumPyro documentation. (n.d.). https://num.pyro.ai/

[67] Stan. (n.d.). Stan. https://mc-stan.org/

[68] TensorFlow Probability. (n.d.). TensorFlow. https://www.tensorflow.org/probability

[69] PyG Documentation- pytorch_geometric documentation. (n.d.). https://pytorch-geometric.readthedocs.io/

[70] Themeix. (n.d.). Deep Graph Library. https://www.dgl.ai/

[71] Time Series Made Easy in Python- darts documentation. (n.d.). https://unit8co.github.io/darts

[72] GluonTS documentation. (n.d.). https://ts.gluon.ai/

[73] Kats — Kats. (n.d.). https://facebookresearch.github.io/Kats

[74] auto-sklearn (n.d.). https://pypi.org/project/auto-sklearn/

[75] AutoGluon: AutoML for text, image, time series, and tabular data - AutoGluon Documentation. (n.d.). https://auto.gluon.ai/

[76] H2O.ai — Convergence of the World's Best Predictive and Generative AI for Private, Protected Data. (n.d.). https://www.h2o.ai/

[77] Redirecting. (n.d.). https://epistasislab.github.io/tpot

[78] Optuna-A hyperparameter optimization framework. (n.d.). Optuna. https://optuna.org/

[79] Ray Tune: Hyperparameter Tuning - Ray 2.53.0. (n.d.). https://docs.ray.io/en/latest/tune

[80] Project Jupyter. (n.d.). Home. https://jupyter.org/

[81] Visual Studio Code-The open source AI code editor. (2021, November 3). https://code.visualstudio.com/

[82] Matplotlib - Visualization with Python. (n.d.). https://matplotlib.org/

[83] seaborn: statistical data visualization - seaborn 0.13.2 documentation. (n.d.). https://seaborn.pydata.org/

[84] Vega-Altair: Declarative Visualization in Python - Vega-Altair 6.0.0 documentation. (n.d.). https://altair-viz.github.io/

[85] Plotly. (n.d.). https://plotly.com/python

[86] Streamlit-A faster way to build and share data apps. (n.d.). https://streamlit.io/

[87] Overview - Panel v1.8.7. (n.d.). https://panel.holoviz.org/

[88] DASH Documentation & User Guide — Plotly. (n.d.). https://dash.plotly.com/

[89] Google Model Cards. (n.d.). https://modelcards.withgoogle.com/

[90] ONNX — Home. (n.d.). https://onnx.ai/

[91] Apache. (n.d.). GitHub-apache/tvm: Open Machine Learning Compiler Framework. GitHub. https://github.com/apache/tvm/

[92] IREE. (n.d.). https://iree.dev/

[93] XLA. (n.d.). OpenXLA Project. https://www.tensorflow.org/xla

[94] OpenVINO— Home. (n.d.) https://www.intel.com/openvino

[95] NNI Documentation - Neural Network Intelligence. (n.d.). https://nni.readthedocs.io/

[96] Artificial intelligence (AI) at Red Hat. (n.d.). https://neuralmagic.com/sparseml

[97] Google AI Edge — Google AI for Developers. (n.d.). Google AI for Developers. https://www.tensorflow.org/lite

[98] LiteRT for microcontrollers. (n.d.). Google AI for Developers. https://www.tensorflow.org/lite/microcontrollers

[99] MLFlow. (n.d.). https://mlflow.org/

[100] Home — AIMStack. (n.d.). AimStack. https://aimstack.io/

[101] Welcome to Sacred's documentation! - Sacred 0.8.4 documentation. (n.d.-b). https://sacred.readthedocs.io/

[102] AI Developer Productivity & Agentic Development — Guild.ai. (n.d.). https://guild.ai/

[103] Client challenge. (n.d.-c). https://pypi.org/project/modeldb/

[104] DVC. (2025, December 11). Home – DVC. https://dvc.org/

[105] Conda Documentation- conda-docs documentation. (n.d.). https://docs.conda.io/

[106] Poetry-Python dependency management and packaging made easy. (n.d.). https://python-poetry.org/

[107] Ratliff, S. (2026, January 29). Docker: Accelerated Container Application Development. Docker. https://www.docker.com/

[108] Nix & NixOS — Declarative builds and deployments. (n.d.). Nix & NixOS. https://nixos.org/

[109] Home. (2026, January 22). Apache Airflow. https://airflow.apache.org/

[110] Argo workflows. (n.d.). Argo. https://argoproj.github.io/workflows

[111] Kubeflow. (n.d.). Kubeflow. https://www.kubeflow.org/

[112] Dagster. (n.d.). Modern Data Orchestrator Platform — Dagster. https://dagster.io/

[113] Welcome to Metaflow — Metaflow Docs. (n.d.). https://docs.metaflow.org/

[114] Redirecting to KServe documentation. (n.d.). https://kserve.github.io/

[115] Seldon. (2026, January 21). Seldon-take control of ML and AI complexity. Take Control of ML and AI Complexity. https://www.seldon.io/

[116] TorchServe - PyTorch/Serve master documentation. (n.d.). https://pytorch.org/serve

[117] Bento: Run Inference at scale. (n.d.). https://bentoml.com/

[118] NVIDIA Developer. (n.d.). NVIDIA Dynamo-Triton. https://developer.nvidia.com/nvidia-triton-inference-server

[119] FastAPI. (n.d.). https://fastapi.tiangolo.com/

[120] GRPC. (n.d.). gRPC. https://grpc.io/

[121] Tekton. (n.d.). Tekton. https://tekton.dev/

[122] ARGO CD-Declarative GITOPS CD for kubernetes. (n.d.). https://argo-cd.readthedocs.io/

[123] Jenkins. (n.d.). Jenkins. https://www.jenkins.io/

[124] Evidently AI-AI Evaluation & LLM Observability platform. (n.d.). https://www.evidentlyai.com/

[125] Whylogs. (n.d.). https://pypi.org/project/whylogs/

[126] Prometheus-Monitoring system & time series database. (n.d.). https://prometheus.io/

[127] Grafana Labs. (n.d.). Grafana: The open and composable observability platform — Grafana Labs. https://grafana.com/

[128] Welcome to the SHAP documentation - SHAP latest documentation. (n.d.). https://shap.readthedocs.io/en/latest/

[129] lime. (2020, June 26). PyPI. https://pypi.org/project/lime/

[130] Captum · Model Interpretability for PyTorch. (n.d.). https://captum.ai/

[131] Fairlearn. (n.d.). https://fairlearn.org/

[132] aif360. (2024, April 8). PyPI. https://pypi.org/project/aif360/

[133] adversarial-robustness-toolbox. (2025, July 7). PyPI. https://pypi.org/project/adversarial-robustness-toolbox/

[134] sigstore. (n.d.). Sigstore. https://www.sigstore.dev/

[135] Anchore. (n.d.). GitHub-anchore/syft: CLI tool and library for generating a Software Bill of Materials from container images and filesystems. GitHub. https://github.com/anchore/syft

[136] Anchore. (n.d.-a). GitHub-anchore/grype: A vulnerability scanner for container images and filesystems. GitHub. https://github.com/anchore/grype

[137] Client challenge. (n.d.). https://pypi.org/project/pip-audit

[138] in-toto. (n.d.). In-toto. https://in-toto.io/