Model Infrastructure Best Practices

Table of Contents

Machine Learning Automation Infrastructure 2

Importance of Machine Learning Infrastructure 3

Data Pipeline Flow 5

Data Pipeline Best Practices 5

Scalability and Performance 7

Importance of Monitoring and Optimization 7

Challenges and Issues If Neglected 8

Recommended Practices 9

Machine Learning Security and Compliance 10

Machine Learning Threats 11

Recommended Practices 12

Documentation and Collaboration 14

Documentation 14

Best Practices 15

Model Cards for Documenting the Model Details 15

Collaboration 18

Challenges that may Arise due to Lack of Collaboration? 18

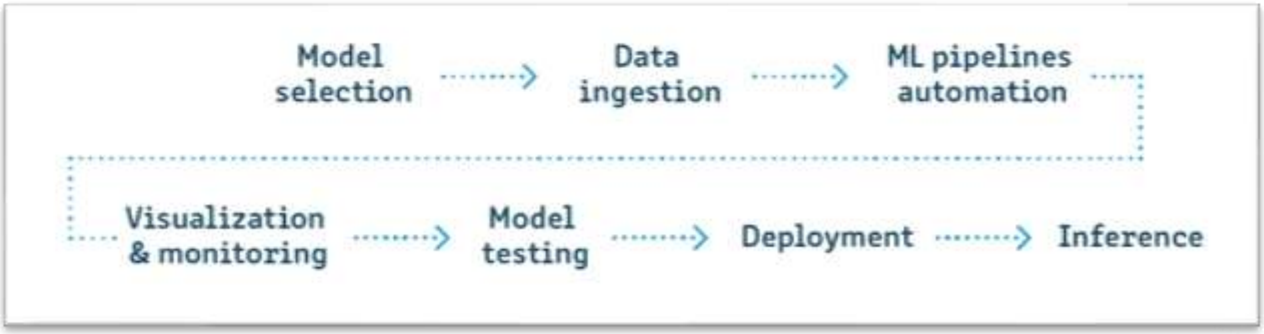
Concerned Issues and Probable Solutions 18

Best Practices 19

Machine Learning Automation Infrastructure

Machine learning infrastructure includes the resources, processes, and tooling needed to develop, train, and operate machine learning models. It is sometimes referred to as AI infrastructure or a component of MLOps.

ML infrastructure supports every stage of machine learning workflows. It enables data scientists, engineers, and DevOps teams to manage and operate the various resources and processes required to train and deploy neural network models. Following images show the Machine Learning infrastructure automation building blocks.



[Source: Link]

1. **Model Selection:** Machine learning model selection is the process of selecting the best fitting model. Based on the selected model, different parameters like training testing split ratio, training dataset format and type changes.

2. **Data Ingestion:** This stage refers to the ETL setup of the Machine Learning pipeline models. Extracting the data, cleaning and processing data for generating a valid input dataset for the models. Enabling the automation with incremental data load.

3. **ML Pipeline Automation:** This stage includes setting up the automation for the data pipelines (ETL) and machine learning model pipelines like training the model, evaluation of the model and deployment of the model via CI/CD.

4. **Visualization and Monitoring:** Visualization and monitoring refers to visualizing and monitoring the working of workflows, model training as well as model results. Model visualization is used to optimize and make the MLOps architecture more efficient.

5. **Model Testing:** Machine learning models are required to be tested and evaluated against the required performance metrics in training and as well as deployment phases. Thorough testing refers to,
    a. Collection and analysis of quantitative and qualitative data.
    b. Multiple training runs in identical environments.
    c. The ability to identify where the errors occurred.

6. **Deployment:** This stage refers to packaging of the model and deploying it to the higher environment for the user consumption. Proper automation should be setup along with validations in-place to validate deployment logs, incoming data to the deployed model, model response for the data and alerts if responses is highly irrelevant/harmful for user or model.

7. **Inference:** During the deployment stage, it is crucial to assess and choose deep learning frameworks that align with the ongoing inference needs of new data. The selected framework must be optimized for performance in production while considering hardware constraints. For instance, a computer vision model in a self-driving car necessitates millisecond-speed inference, accounting for on-board hardware capabilities. The process of transitioning models between frameworks to meet production requirements has been simplified with the advent of universal model file formats like the Open Neural Network eXchange (ONNX), facilitating easier portability across different libraries.

Importance of Machine Learning Infrastructure

Machine learning infrastructure is crucial for several reasons as it forms the backbone that supports the development, deployment, and maintenance of machine learning (ML) models. Here are key reasons why machine learning infrastructure is important:

1. **Scalability and Performance**

    a. Horizontal and Vertical Scaling: ML infrastructure facilitates the scaling of resources both horizontally (across multiple machines) and vertically (increasing resources on a single machine). This is essential for handling larger workloads and datasets efficiently.

2. **Resource Optimization**

    a. Memory Management*:* Efficient memory management is vital for handling large datasets and complex models. ML infrastructure provides tools for optimizing memory usage during training and inference.

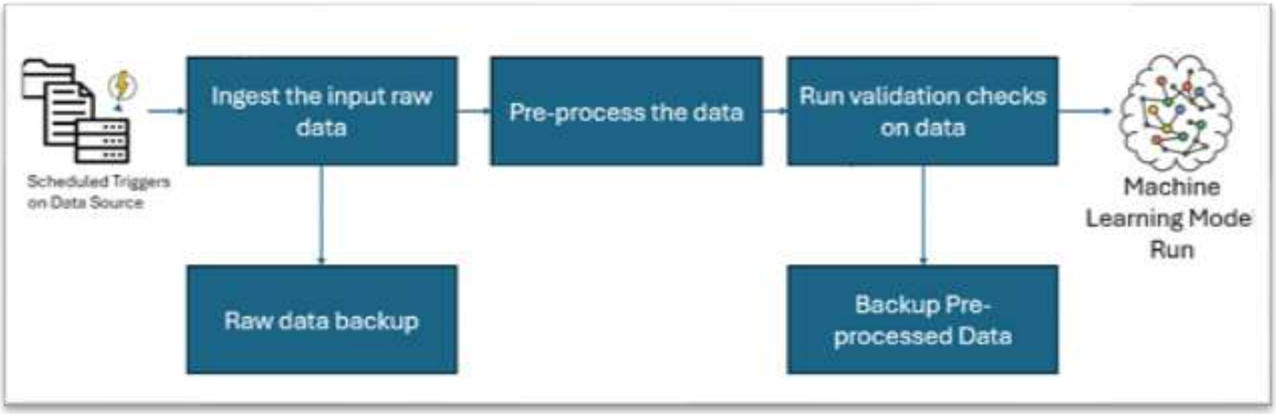
    b. Parallel Processing*:* ML infrastructure supports parallel processing, enabling the concurrent execution of tasks. This is crucial for reducing training time and improving overall performance.

3. **Handling Large Datasets:** Efficient storage and retrieval of large datasets are fundamental in ML workflows. Infrastructure provides solutions for storing, accessing, and preprocessing extensive datasets, ensuring smooth model training and evaluation.

4. **Model Deployment and Serving:** ML infrastructure enables the deployment of trained models into production environments. This involves creating APIs, handling model versioning, and ensuring models can handle real-world requests efficiently.

5. **Experimentation and Model Training:** Infrastructure supports the setup and management of experimentation pipelines. It includes tools for version control, tracking experiments, and managing dependencies, which are essential for reproducibility and collaboration.
6. **Monitoring and Logging:** Infrastructure provides capabilities for monitoring model performance in real-time. This involves tracking metrics, logging, and setting up alerts to detect issues and anomalies during inference or training.
7. **Automation and Orchestration:** ML infrastructure automates repetitive tasks in the ML lifecycle, such as data preprocessing, model training, and deployment. Orchestration tools ensure seamless coordination of these tasks in complex workflows.
8. **Security and Compliance:** Security is paramount in ML, especially when dealing with sensitive data. Infrastructure includes features for securing data, models, and APIs. Compliance requirements are addressed to meet regulatory standards.
9. **Collaboration and Reproducibility:** ML infrastructure facilitates collaboration among team members by providing shared environments, version control, and reproducibility of experiments. This ensures consistency and transparency in the development process.
10. **Cost Optimization:** ML infrastructure allows for efficient resource utilization, leading to cost savings. Auto-scaling mechanisms, resource allocation strategies, and optimization techniques contribute to managing infrastructure costs effectively.

Data Pipeline Flow


Scheduled Triggers on Data Source
Ingest the input raw data
Pre-process the data
Run validation checks on data
Machine Learning Mode Run
Raw data backup
Backup Pre-processed Data

Data Pipeline Best Practices

1. Schedule your pipeline runs to avoid peak demand. ML pipelines can be resource-intensive, so it is important to schedule them to avoid peak demand on your compute resources. This can help to improve performance and reduce costs.

    a. Keep most of the resources in the cloud environment.

    b. Make sure to keep autoscaling enabled for the resources wherever possible.

    c. Enable alerting mechanism once the pipeline runs start via email or other means.

    d. Set up checks in the initial phase of pipeline for latest availability of data and if it passes, then only let the pipeline proceed further. It will save the resources' cost.

2. Use a trigger to start your pipeline run. A trigger can be based on an event, such as a new file being added to a data store or a notification from an external system. Using a trigger to start your pipeline runs can help to ensure that your data is always processed in a timely manner.

    a. If the data is very frequently getting updated, setup fixed timed interval data pull to keep our system updated with the latest data.

    b. If the data is infrequent, set up the trigger for daily, monthly, or weekly basis.

    c. Azure resources provide the scheduled trigger option in Data factory.

    d. If data is copied/ingested using Power Automate Flows, then multiple triggers are available based on file creation, update, deletion to trigger the flow.

    e. If there is a centralized trigger scheduler available, utilize that to monitor all the triggers at once.

3. Monitor your pipeline runs to ensure that they are completing successfully. You can use monitoring tools to track the status of your pipeline runs, identify any errors, and receive alerts if there are any problems.

    a. Use centralized monitoring tools/mechanisms like Azure Monitor, as it provides monitoring of all resources including data pipelines, Azure Logic Apps along with their metric logs and alerts.

    b. Create a custom dashboard from this with required number of resources to create an ease in monitoring.

    c. If a team is not actively monitoring via monitor, then set up alert emails of pipeline stage completion or failures. Logic apps can be configured for this.

4. Have a retry mechanism in place for failed pipeline runs. In the event of a failure, it is important to have a retry mechanism in place to ensure that your pipeline run is eventually successful. This may involve retrying the run a certain number of times or retrying the run at a later time.

    a. Enable retry for every possible activity in pipelines. Three retries are recommended.

    b. Copy activity, Databricks notebook run, logging activity, should have retries enabled.

    c. A pipeline/activity should have 3 times retry enabled.

5. Use a scheduler that is designed for ML pipelines. ML pipelines can be complex and require specific resources, such as GPUs. There are several schedulers that are designed specifically for ML pipelines, such as Kubeflow Pipelines and MLflow.

    a. GPU requirements are majorly if you are training a deep learning models like CNN, GAN or other very complex architectures. TPUs are also available in Google environment.

    b. CUDA is provided with a high compute of GPU powers by Nvidia, for parallel processing.

    c. Microsoft also provides different virtual machines backed by different processing powered GPUs. For further details of those GPUs, this documentation can be referred to.

6. Check for securities while scheduling for MLOps data pipelines. Use Service Principles or Managed Identity provided by Microsoft to access a resource or to trigger a pipeline.

Following are few necessary practices for scheduling the data pipelines for MLOps.

1. Check for the data availability and schedule the trigger.

    a. If source data has a predefined data source, where file is created or deleted every day, week or other at other time frequencies. Set a trigger on data update of the data source, such that latest data can be ingested.

   b. If pipelines requirement is to run at particular time frequency, schedule a trigger for that by specifying specific date time manner.

2. Opt for scalable resources.
   a. Self Hosted Runtime
      i. If you are converting data to parquet format, it requires Java self hosted run time in the backend to convert and compress the data. Set up auto scalability for runtime to increase the number of nodes (on-prem or virtual machines) based on requirement.
      ii. Setup alerts for high usage of a resource/ or low capacity available for resource. And based on that, setup resource scaling via Power Shell or other methods.
   b. Compute Resources: While doing data transformation or processing using Azure Databricks or Azure ML, keep the autoscaling on for the resources. It will allow the architecture to handle multiple pipelines running simultaneously.
   c. Storage Resource: Azure offers unlimited amount of storage with different pricing tiers. Configure the storage to handle multiple data pipeline simultaneously.
3. Check for reuse of the utilities.
4. Optimize the cost by choosing the best architecture.
   a. Optimized different types of architectures for MLOps are provided by Microsoft and other providers.
   b. Microsoft best architectures for Databricks: Link
   c. Microsoft best architecture for Azure ML Studio: Link

Scalability and Performance

Monitoring and optimizing resource/entities scaling and performance are essential for delivering a reliable, cost-effective, and responsive system while mitigating potential challenges and issues that can arise if neglected.

Importance of Monitoring and Optimization

1. **User Experience**
   a. Ensures a seamless and responsive user experience.
   b. Optimized performance leads to faster response times and better user satisfaction.
2. **Resource Efficiency**

   a. It Maximizes resource utilization, minimizing operational costs.
   b. Efficient scaling adapts to varying workloads, preventing underutilization or overload.

3. **Reliability and Availability**
   a. Monitored systems are more reliable and less prone to unexpected failures.
   b. Optimized scaling enhances system availability, reducing downtime.

4. **Cost-Effectiveness**
   a. Efficient scaling and resource optimization contributes to cost-effective infrastructure management.
   b. Avoids unnecessary expenditure on excess resources or emergency capacity provision.

5. **Proactive Issue Resolution**
   a. Enables early detection of performance bottlenecks and scalability limitations.
   b. Proactive resolution minimizes the impact on users and prevents potential disruptions.

Challenges and Issues If Neglected

1. **System Bottlenecks**
   a. Inadequate scalability leads to bottlenecks during periods of increased demand.
   b. Results in slower response times and compromised system performance.

2. **Increased Operational Costs**
   a. Unoptimized resource usage escalates operational costs.
   b. Inefficient scaling may necessitate unnecessary resource provisioning.

3. **Memory Leaks and Code Inefficiencies**
   a. Lack of monitoring may allow memory leaks and inefficient code to go unnoticed.
   b. Results in increased memory consumption, potential crashes, and degraded performance.

4. **Data Processing Inefficiency**
   a. Neglecting optimization in handling large datasets leads to sluggish performance.

   b. Hinders the ability to extract meaningful insights from data.

5. **User Dissatisfaction**

   a. Poorly performing systems impact user satisfaction.

   b. Leads to decreased user engagement and potential loss of trust in the platform.

6. **Unplanned Downtime**

   a. Inadequate scalability planning may result in unplanned downtime during traffic spikes.

   b. Disrupts service availability and negatively impacts user experience.

Recommended Practices

1. **Use Distributed Computing**

   a. Utilize frameworks like Apache Spark or TensorFlow on distributed clusters.

   b. Enables parallel processing and distributed training of ML models.

   c. Scales ML workloads across multiple machines, reducing training time and increasing throughput.

2. **Containerize ML Applications**

   a. Employ containerization tools such as Docker for ML applications.

   b. Provides a portable and isolated environment.

   c. Enables easy deployment, scalability, and consistent behavior across different environments.

   d. Allows deployment on cloud-based container orchestration platforms like Kubernetes or Docker Swarm for efficient scaling.

3. **Horizontal Scaling**

   a. Distribute workload across multiple machines or instances.

   b. Achieved by adding compute resources, like virtual machines or containers.

   c. Facilitates parallel processing, improving overall performance and capacity.

   d. Implement load balancing techniques for even distribution of workload among available resources.

4. **Optimize Data Pipelines**

   a. Optimize complex data pipelines for data preprocessing, feature engineering, and model training.

    b. Techniques like data partitioning, caching, and lazy evaluation minimize data movement and unnecessary computations.

    c. Results in faster processing and reduced resource requirements.

5. **Auto-scaling**

    a. Implement auto-scaling mechanisms for dynamic resource adjustments based on workload demands.

    b. Achieved through built-in features on cloud platforms or container orchestration frameworks.

    c. Ensures optimal resource allocation, minimizing costs during low demand and meeting performance requirements during peak times.

6. **Monitoring and Performance Tuning**

    a. Regularly monitor ML workload performance to identify bottlenecks and optimize resource utilization.

    b. Use monitoring tools for metrics related to CPU usage, memory consumption, network traffic, and latency.

    c. Apply performance tuning techniques, such as algorithm optimization and model compression, to enhance scalability and efficiency.

7. **Data Partitioning and Shuffling**

    a. Implement proper data partitioning and shuffling for efficient distributed processing.

    b. Partition data based on key attributes or use techniques like random partitioning.

    c. Ensures even distribution of workload across resources, minimizing data movement, and improving overall performance.

Machine Learning Security and Compliance

The integration of machine learning (ML) technology into various aspects of daily life, such as healthcare, finance, mobile devices, automobiles, and security systems, presents a new attack surface. Although machine learning is not currently a common target for cyberattacks, it is anticipated to become more attractive to attackers as its usage becomes more widespread. This post emphasizes the vulnerability of ML systems to cyber threats and urges teams to proactively address security risks and implement mitigations as part of the development and deployment process. The goal is to raise awareness about potential risks and encourage a proactive approach to security in the evolving landscape of machine learning technology.

Machine Learning Threats

[Reference: Link]

Machine learning solutions can be vulnerable to traditional software application risks as well as risks unique to Machine Learning domain. Types of security threats include:

1. **Data Poisoning:** Data poisoning happens when the data an ML model uses to produce an outcome is tampered with and the model produces an incorrect result.
    - An early example of data poisoning is the attack on the Tay AI chatbot that Microsoft launched in 2016 which was taken offline within 24 hours of launch. The model used twitter interactions to learn and respond to users. Adversaries exploited the model by combining a 'repeat after me' function with racist and offensive language that forced Tay to repeat anything that was said to it [Ref: Learning from Tay's introduction].

2. **Model Theft:** As models represent a significant investment in Intellectual Property, they can be a valuable target for theft. And like other software assets, they are tangible and can be stolen. Model theft happens when a model is taken outright from a storage location or re-created through deliberate query manipulation.
    - An example of this type of attack was demonstrated by a research team at UC Berkeley who used public endpoints to re-create language models with near-production state-of-the-art translation quality. The researchers were then able to degrade the performance and erode the integrity of the original machine learning model using data input techniques to compromise the integrity of the original machine learning model (see Data Poisoning above). Another example of model theft is documented when Brown University researchers replicated the GPT-2 model using information released by OpenAI and open-source ML artifacts.

3. **Inversion:** An inversion attack happens when data is retrieved from a model – or the related workflows - and combined in ways that result in a data privacy leak. This type of attack happens when a model outputs sensitive information or there is a vulnerability in a solution pipeline that allows access to data storage or displays verbose error messages that expose sensitive data.
    - Examples of inversion attacks include an attack where confidential data used to train a model is recovered. Example is: Is Tricking a Robot Hacking? with additional research examples of inversion attacks documented here: Model Inversion Attacks.

4. **Extraction/Evasion:** Referred to as *Adversarial Perturbation*, this type of attack happens when a model is tricked into misclassifying input and returns an incorrect or unintended result.
    - Examples of this type of attack include:
        a. A facial recognition solution is compromised when an attacker gets access to the storage account containing training data and modifies the images to disguise individuals from facial recognition

software: https://atlas.mitre.org/studies/AML.CS0006https://atlas.mitre.org/studies/AML.CS0011/

b. A research project to automate the manipulation of a target image causes an ML model to misclassify, see Microsoft Edge AI Evasion
c. A physical domain attack when self-driving cars were tricked to think a stop sign is a speed limit sign, and

An authentication system is compromised when the confidence output is maximized allowing an authorization check to pass allowing an invalid authorization attempt.

A recommended source for learning about adversarial threats against machine learning solutions is the MITRE ATLAS framework. The tactics and techniques used in AI/ML attacks can be explored by clicking through the matrix published with the framework.

Recommended Practices

1. **Build Awareness**
   a. Identify and inventory machine learning assets.
   b. Understand external dependencies and 3rd party contributions.
   c. Recognize where the business incorporates machine learning components.
2. **Early Security Consideration**
   a. Collaborate with the security team to understand security and compliance requirements.
   b. Stay informed about industry-specific attacks on machine learning through various sources.
   c. Align solutions with responsible AI practices, fairness, privacy, and ethics.
3. **Threat Modeling**
   a. Incorporate machine learning solutions into threat modeling practices.
   b. Acknowledge the potential impact of cyberattacks, even if they are not common.
   c. Follow Microsoft's threat modeling approach and consider AI/ML system-specific guidance.
4. **Protect Data**
   a. Apply security techniques and controls throughout the machine learning pipeline.
   b. Implement recommended practices for access management, data encryption, and monitoring.

    c. Address privacy and compliance requirements, especially with sensitive data.

5. **Security Practices Across Workflows**

    a. Follow well-architected principles for design, development, deployment, and operations.

    b. Implement a landing zone architecture with centralized security capabilities.

    c. Configure role-based access, network isolation, and secure storage for various components.

6. **Security Monitoring and Response**

    a. Invest in security monitoring, detection, and response processes.

    b. Proactively anticipate security compromises by identifying potential threats.

    c. Monitor for behaviors like data exfiltration, unauthorized access, model performance drops, and unusual inferencing patterns.

For detailed explanation on the points, refer to the Microsoft Community Blog [Link].

Documentation and Collaboration

Documentation

Importance of Documenting the ML Code

1. Documenting the ML codes help in understanding the logics, assumptions and results of Machine Learning models.
2. It also enables code reusability for developers.
3. It allows ease in collaboration among different teams to update/extend the code.
4. Documenting the code enhances the reliability of Machine Learning models for future use.

Topics to Document

1. **Project Purpose and Objectives**
    a. Document the problem statement, outlining the challenges the project aims to address.

    b. Specify data sources used in the project and identify the target variable.

    c. Clearly define evaluation metrics that will be used to assess model performance.
2. **Data Processing and Analysis Steps**
    a. Document the data cleaning process, detailing how missing or inconsistent data is handled.

    b. Describe feature engineering techniques applied to enhance the input data.

    c. Include sections on Exploratory Data Analysis (EDA) and data visualization methods utilized.
3. **Machine Learning Models and Algorithms**
    a. Provide documentation on the selection process for machine learning models.

    b. Detail hyperparameter tuning strategies employed to optimize model performance.

    c. Document the entire model lifecycle, covering training, validation, and testing phases.
4. **Results and Insights**
    a. Report on model performance metrics, such as accuracy, precision, recall, etc.

    b. Highlight the importance of features used in the final model.

    c. Include an error analysis section that explores model shortcomings or areas of improvement.

    d. Summarize any actionable recommendations or insights derived from the project.

Best Practices

1. **Document with a purpose**: Before building out documentation, a few questions should be checked such as: Who will consume this documentation? Why do they need this documentation? How would they like to consume documentation? Create various artifacts that best serve each set of stakeholders' needs.
2. **Prioritize deliverables over documentation**: The documentation goal should be to deliver valuable insights and modeling systems that improve your internal stakeholders, end-users, and broader society. For that, Prioritizing deliverables is a key.
3. **Keep it simple**: Keep the language of the document simple and easy to understand. Use proper words which can accurately define the terminologies of machine learning. All professional entities should be able to gain insights from the documented items.
4. **Automate documentation**: Automating documentation can help you save time and reduce errors. Use tools like Sphinx, Doxygen, and Javadoc to automate documentation. For research papers-based documentation, tools like Overleaf which uses LaTex can be used.
5. **Use templates**: Use templates for README files, project plans, and other documentation artifacts. Templates can help to create consistent documentation across projects.
6. **Use diagrams**: Diagrams can help in visualizing complex systems and processes. Use diagrams to explain data flows, system architecture, and other concepts.
7. **Use a wiki**: A wiki is a website that allows users to collaboratively create and edit web pages using a web browser. Use a wiki to create a knowledge base for your team.
8. **Use a style guide**: A style guide is a set of standards for writing and designing documents. Use a style guide to ensure consistency across your documentation.
9. **Use a documentation review process**: A documentation review process can help you ensure that your documentation is accurate, complete, and up-to-date. Use a review process to get feedback from your team and stakeholders.

Model Cards for Documenting the Model Details

[Reference: Link]

1. Engineers from Google proposed a framework called **model cards** to encourage transparent model reporting.
2. Model cards are short documents that provide benchmarked evaluation in a variety of conditions, such as across different cultural, demographic, or phenotypic groups (e.g., race, geographic location, sex, Fitzpatrick skin type) and intersectional groups (e.g., age and race, or sex and Fitzpatrick skin type) that are relevant to the intended application domains.

3. Model cards also disclose the context in which models are intended to be used, details of the performance evaluation procedures, and other relevant information.

4. Released machine learning (ML) models should be accompanied by documentation detailing their performance characteristics to clarify the intended use cases of the models and minimize their usage in contexts for which they are not well-suited.

5. Sections that a model card should contain are,

   a. Model Details: Model background information like model version, developers etc.

   b. Intended Use: This section should contain in-scope use cases, out-scope use case and in-scope users' details.

   c. Factors: Details/features that impact model performance. i.e., for the Smiling face detection model, it can be age, gender, ethnicity etc.

   d. Metrics: Model evaluation metrics details like Accuracy, F1 Score etc.

   e. Evaluation Data: Dataset used for evaluation, reason for selection of mentioned dataset and challenges associated, validation cases.

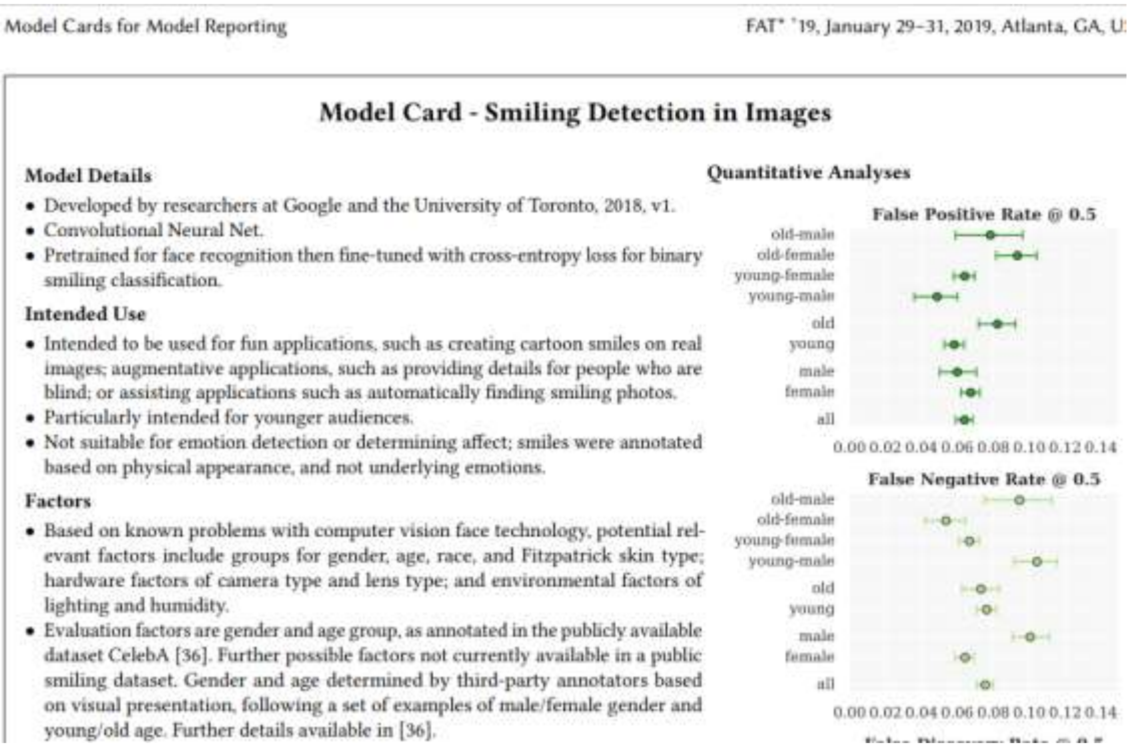
   f. Training Data: Dataset on which model training happened.

   g. Quantitative Analysis: Categorized evaluation of performance. i.e., performance measured for Female category in gender and age group below 30.

   h. Ethical Consideration: Details about any sensitive data used, or model's implication on human life.

   i. Recommendations: Notable points which were not covered in earlier times.

Model Card Example [Reference: Link]



Model Cards for Model Reporting FAT* '19, January 29–31, 2019, Atlanta, GA, U

Model Card - Smiling Detection in Images

Model Details

- Developed by researchers at Google and the University of Toronto, 2018, v1.
- Convolutional Neural Net.
- Pretrained for face recognition then fine-tuned with cross-entropy loss for binary smiling classification.

Intended Use

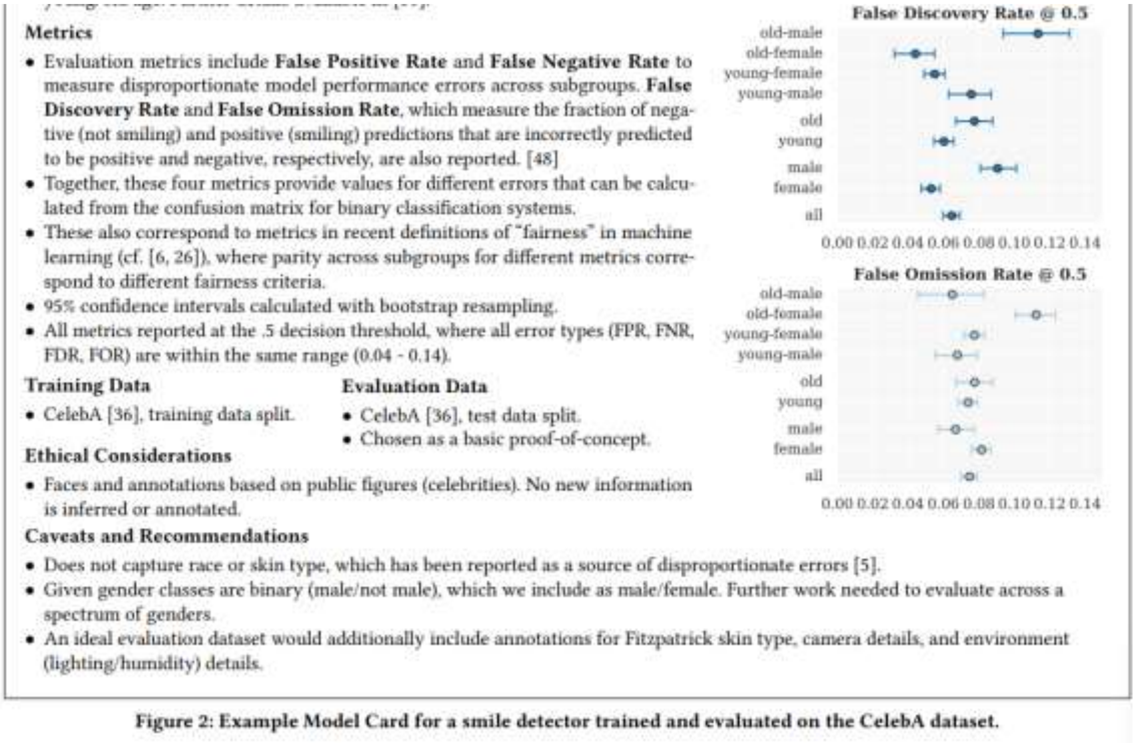
- Intended to be used for fun applications, such as creating cartoon smiles on real images; augmentative applications, such as providing details for people who are blind; or assisting applications such as automatically finding smiling photos.
- Particularly intended for younger audiences.
- Not suitable for emotion detection or determining affect; smiles were annotated based on physical appearance, and not underlying emotions.

Factors

- Based on known problems with computer vision face technology, potential relevant factors include groups for gender, age, race, and Fitzpatrick skin type; hardware factors of camera type and lens type; and environmental factors of lighting and humidity.
- Evaluation factors are gender and age group, as annotated in the publicly available dataset CelebA [36]. Further possible factors not currently available in a public smiling dataset. Gender and age determined by third-party annotators based on visual presentation, following a set of examples of male/female gender and young/old age. Further details available in [36].

Quantitative Analyses

[Image 1: Model Details, Use and Factors]



Metrics

- Evaluation metrics include **False Positive Rate** and **False Negative Rate** to measure disproportionate model performance errors across subgroups. **False Discovery Rate** and **False Omission Rate**, which measure the fraction of negative (not smiling) and positive (smiling) predictions that are incorrectly predicted to be positive and negative, respectively, are also reported. [48]
- Together, these four metrics provide values for different errors that can be calculated from the confusion matrix for binary classification systems.
- These also correspond to metrics in recent definitions of "fairness" in machine learning (cf. [6, 26]), where parity across subgroups for different metrics correspond to different fairness criteria.
- 95% confidence intervals calculated with bootstrap resampling.
- All metrics reported at the .5 decision threshold, where all error types (FPR, FNR, FDR, FOR) are within the same range (0.04 - 0.14).

Training Data

- CelebA [36], training data split.

Evaluation Data

- CelebA [36], test data split.
- Chosen as a basic proof-of-concept.

Ethical Considerations

- Faces and annotations based on public figures (celebrities). No new information is inferred or annotated.

Caveats and Recommendations

- Does not capture race or skin type, which has been reported as a source of disproportionate errors [5].
- Given gender classes are binary (male/not male), which we include as male/female. Further work needed to evaluate across a spectrum of genders.
- An ideal evaluation dataset would additionally include annotations for Fitzpatrick skin type, camera details, and environment (lighting/humidity) details.

Figure 2: Example Model Card for a smile detector trained and evaluated on the CelebA dataset.

[Image 2: Detailed Model]

Collaboration

Collaboration basically puts emphasis on ease in communication, support among different teams working together to make a Machine Learning project complete and setting up proper harmony among them to increase efficiency as well as reducing latency or irrelevant delays that may arise due to lack of domain or another knowledge. Majorly to complete a one Machine Learning project, Data Engineers, Data Scientists, Infrastructure team, Finance team, Business team and Software engineering teams are required to gather the requirements, initiate a project, set up development environment, setup resources required, estimate and approve the budget amount and setting up the CI/CD pipeline to deploy and deliver the builds to production.

Challenges that may Arise due to Lack of Collaboration?

Three steps where Machine Learning teams majorly face challenges are [Reference: Link]:

1. Requirement and project planning
2. Getting proper training data
3. Product-model integration

Concerned Issues and Probable Solutions

Communication

1. Miscommunication is a common issue in interdisciplinary collaboration, especially between participants with different backgrounds.
2. ML literacy for software engineers, managers, and even customers is essential.
3. Training data scientists to understand software engineering concerns is crucial for effective collaboration.
4. The concept of T-shaped professionals, having deep expertise in one area and broad knowledge of others, is recommended for hiring and training.

Documentation

1. Clearly documenting expectations between teams is vital.
2. Traditional interface documentation is a starting point, but practices for documenting ML model requirements, data expectations, and assured model qualities are not well established.
3. Suggestions like model cards and FactSheets are recommended for standardized documentation of ML components.
4. Documentation must be understood by all involved, and the use of boundary objects may improve interface description mechanisms.

Engineering

1. Organizations often underestimate the engineering effort required to turn an ML model into a reliable product.
2. Adopting machine learning increases software complexity, making engineering practices such as data quality checks, deployment automation, and testing in production more important.
3. Project managers should ensure sufficient engineering capabilities for both ML and non-ML parts of the project from the beginning.

Process

1. ML challenges traditional software process life cycles, requiring a more science-like approach.
2. Involving data scientists for model prototyping is essential in establishing product requirements.
3. Adopting a model-first trajectory may reduce risk, but balancing focus on the product and overall process is crucial.
4. More research into integrated process life cycles for ML-enabled systems is needed, covering software engineering and data science aspects.

Best Practices

Communication Frequently and Effectively

1. **Importance of Communication**
    a. Communication is crucial for effective collaboration in machine learning projects.

    b. Understanding the problem, data, methods, and results relies on clear communication among team members.
2. **Clarity and Frequency**
    a. Communicate clearly and frequently with team members to ensure everyone is on the same page.

    b. Use common terms, avoid jargon, and explain assumptions and goals to enhance understanding.
3. **Utilizing Appropriate Channels**
    a. Choose appropriate communication channels based on the purpose and urgency of the message.

    b. Channels may include email, chat, video calls, or documentation, depending on the context.

4. **Feedback and Input**

    a. Actively seek feedback and input from team members to foster collaboration and improvement.

    b. Be open to different perspectives and suggestions, creating a more dynamic and innovative team environment.