Principles of AI Engineering Chapter 6: Model deployment and software architecture

Prof. Dr. Steffen Herbold

Credit:

Based on contents from Christian Kästner (https://github.com/ckaestne/seai)

Contents

- Simple deployments
- Software architecture
- Design decisions
- Design pattern for systems with ML components
- Documenting model inference interfaces

Simple deployments

Deploying a model is easy!

1. Define a REST API

```python
from flask import Flask, escape, request
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = '/tmp/uploads'
detector_model = … # load model…

# inference API that returns JSON with classes
# found in an image
@app.route('/get_objects', methods=['POST'])
def pred():
    uploaded_img = request.files["images"]
    coverted_img = … # feature encoding of uploaded img
    result = detector_model(converted_img)
    return jsonify({"response":
                result['detection_class_entities']})
```

2. Deploy in Docker Container

```dockerfile
FROM python:3.8-buster
RUN pip install uwsgi==2.0.20
RUN pip install numpy==1.22.0
RUN pip install tensorflow==2.7.0
RUN pip install flask==2.0.2
RUN pip install gunicorn==20.1.0
COPY models/model.pf /model/
COPY ./serve.py /app/main.py
WORKDIR ./app
EXPOSE 4040
CMD ["gunicorn", "-b 0.0.0.0:4040", "main:app"]
```

3. Put container on a cloud and use auto-scaling

There are dedicated solutions to this (e.g., BentoML, Cortex, TFX model serving, Seldon, …)

… but is it really easy?

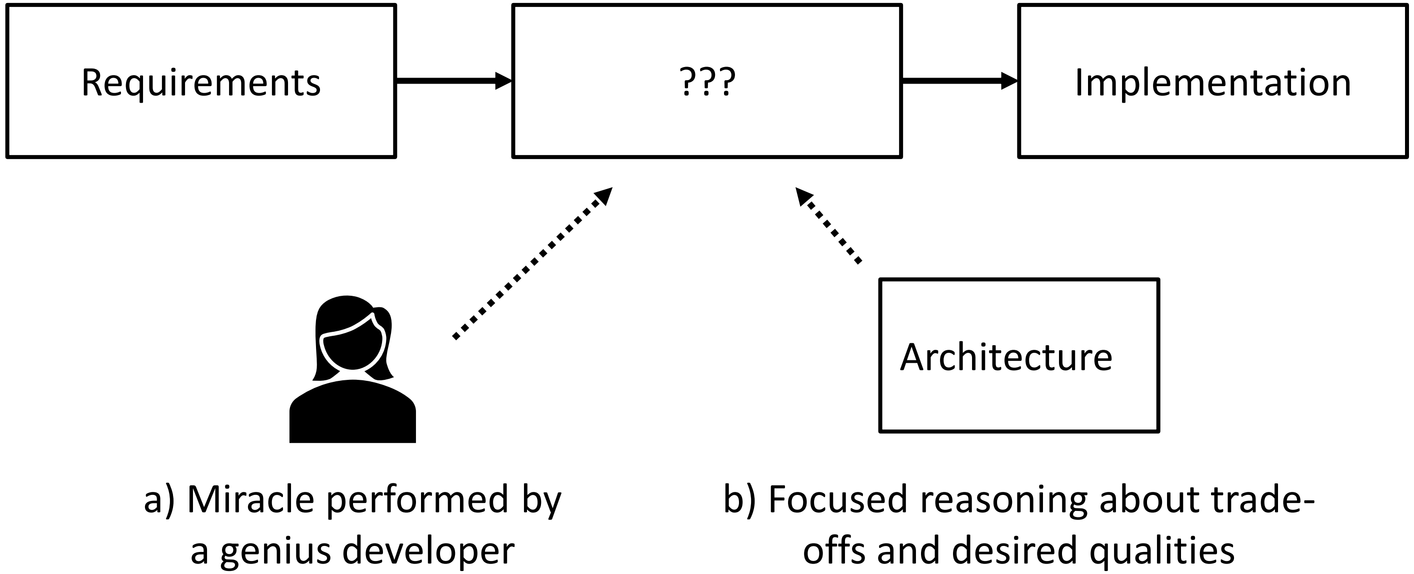
- Offline use?
- Deployment at scale?
- Hardware needs and operating costs?
- Frequent updates?
- Integration of the model into a system?
- Meeting system requirements?

- Every system is different! Not everything is a web service!
  - Personalized music recommendations → Playlists created online only, privacy important
  - Transcription service → Transcription online only, works with large amounts of data
  - Self-driving car → Many different ML components (vision, steering, …) that interact locally
  - Smart keyboard for a mobile device → Very limited compute resources (and storage?)

Software architecture

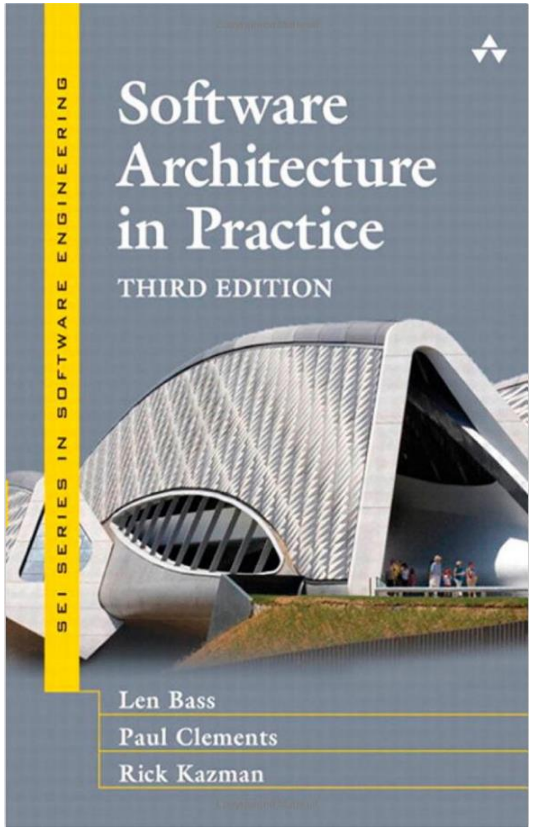
So far: Requirements

- Identify goals, define success metrics
- Understand the requirements, specifications, and assumptions
- Consider risks, plan mitigations
- Understand quality requirements and constraints for models and learning algorithms



Software architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.



Rick Kazman, Paul Clements, and Len Bass. Software architecture in practice. Addison-Wesley Professional

Importance of architecture

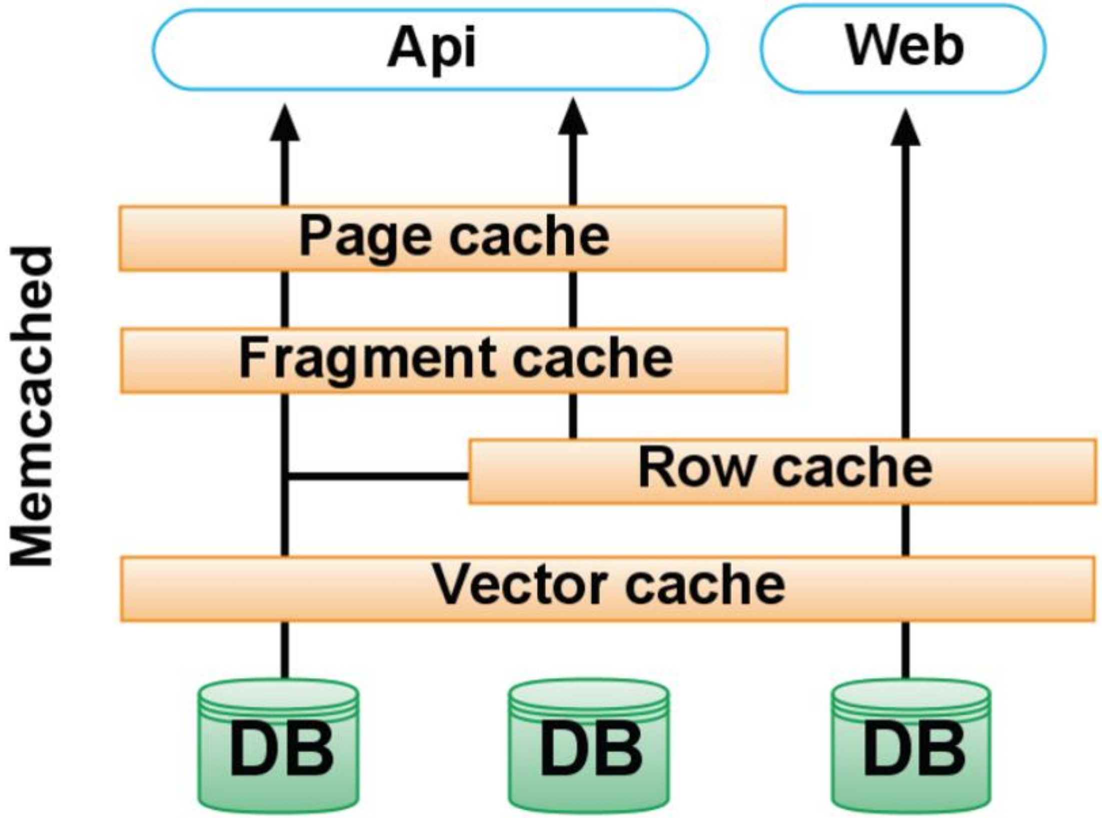
- Represents earliest design decisions
- Aids in communication with stakeholders
  - Shows them *how* at a level they can understand, raising questions whether it meets their needs
- Defines constraints on the implementation
  - Design decision form a *load-bearing wall* of application (e.g., interfaces and how scaling is achieved)
- Dictates organizational structure
  - Teams work on different components
- Inhibits or enables quality attributes
  - Similar to design patterns
- Supports predicting costs, quality, and schedule
  - Typically by predicting information for each component → breaks down complexity
- Aids in software evolution
  - Breaking down complexity aids change analysis
- Aids in prototyping
  - Can implement architectural skeleton early

Example: Twitter redesign

https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how

Old architecture

- One of the world‘s larges Ruby on Rails installations
- 200 engineers working on a monolithic architecture
  - Manages raw database
  - Memcached + multiple dedicated caches
  - Public Twitter API
  - Rendering the Website
- Increasingly difficult to understand the system
  - Organizational challenge to distribute and parallelize tasks
- Reached limit of throughput of the storage system (MySQL)
- Increasing number of machines only limited potential
  - Low throughput per machine (CPU+RAM limited, network not saturated)
- Potential for optimization: trade of readability vs. performance



https://www.slideshare.net/Eweaver/improving-running-components-at-twitter
https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how

Redesign goals

- Performance
  - Improve median latency, reduce outliers
  - Reduce number of machine by a factor of ten

- Reliability
  - Isolate failures

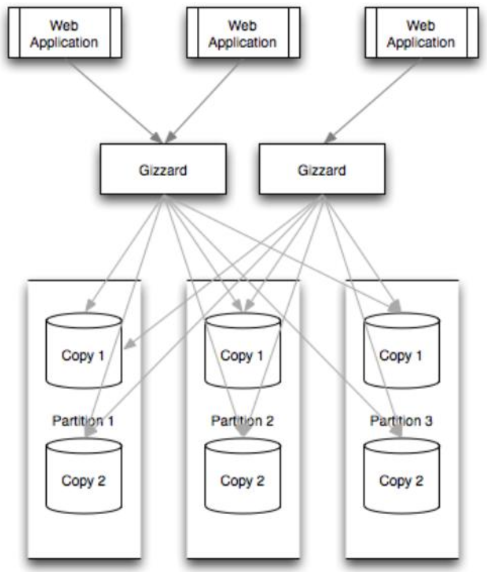
- Maintainability
  - Encapsulation and modularity at the system level (rather than at the class, module, or package level): “We wanted cleaner boundaries with related logic being in one place”

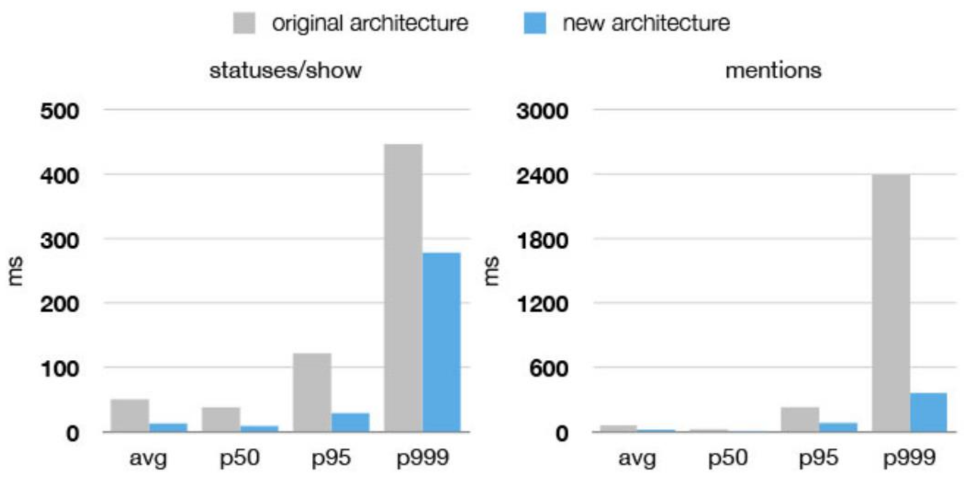
- Modifiability
  - Quicker release of new features: “run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams”

https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how

New architecture

- JVM/Scala instead of Ruby on Rails
- Microservices instead of a monolith: one service for tweets, one service for the timeline, …
- RPC framework with built-in monitoring, connection pooling, failover strategies, load balancing, …
- Gizzard as new storage solution with temporal clustering and roughly sortable Ids





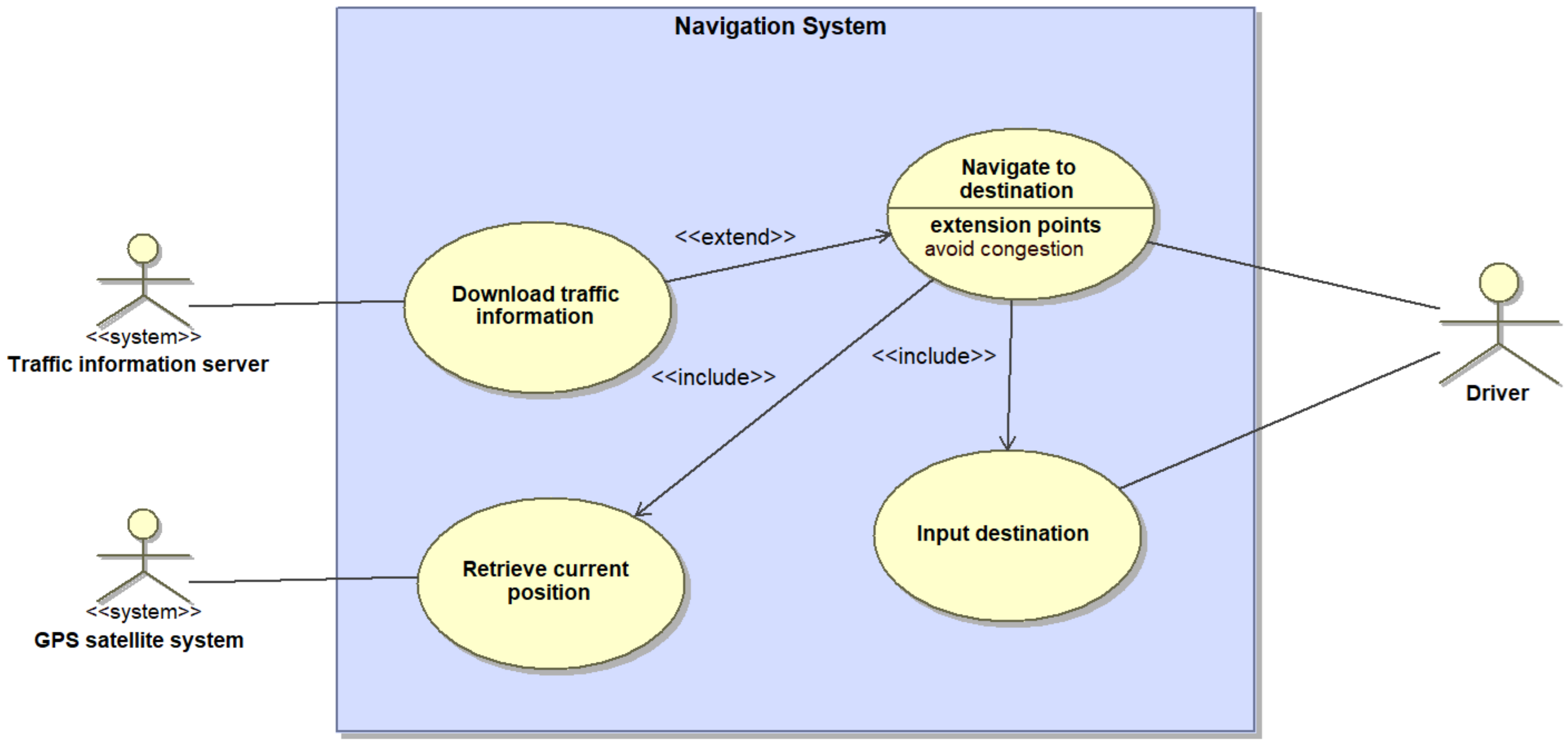
https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how
https://blog.twitter.com/engineering/en_us/a/2010/introducing-gizzard-a-framework-for-creating-distributed-datastores

Insights from Twitter Redesign

- Architectural decision have a huge impact on the entire system
- Good decision require early reasoning about quality attributes
- Architectural decision should be made explicit and documented

Live exercise: Did the original architects make poor decisions?

Reasoning about architecture: Use cases and interfaces



Reasoning about architecture: Data flow and storage components

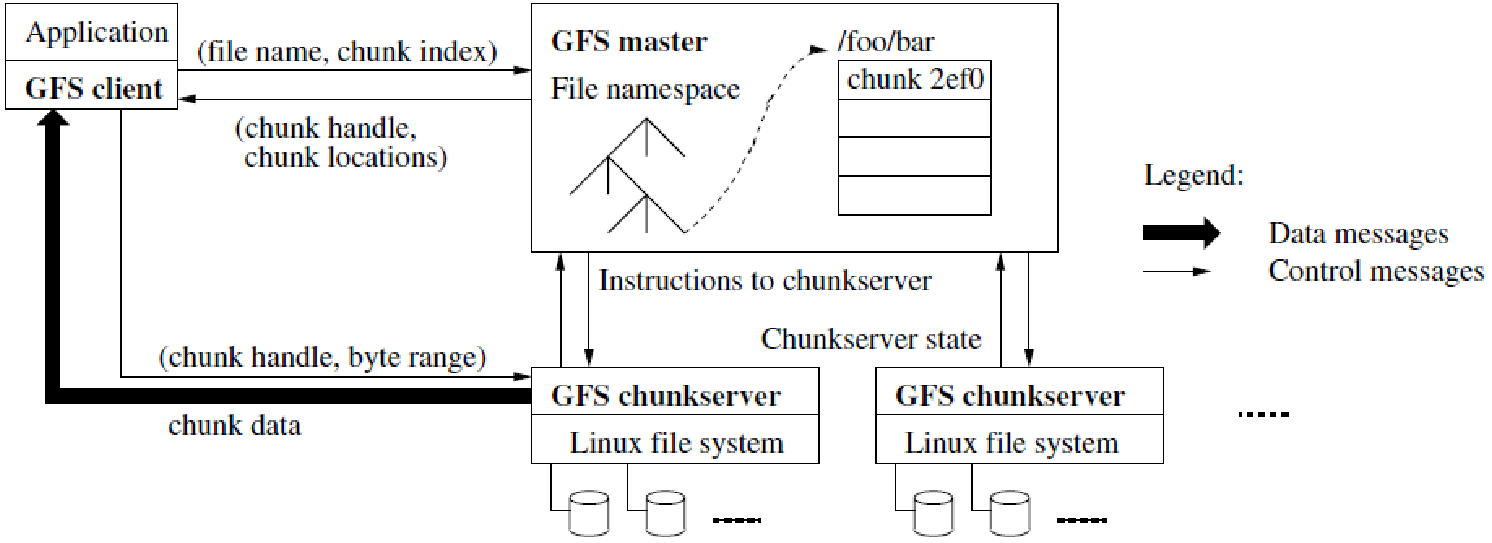
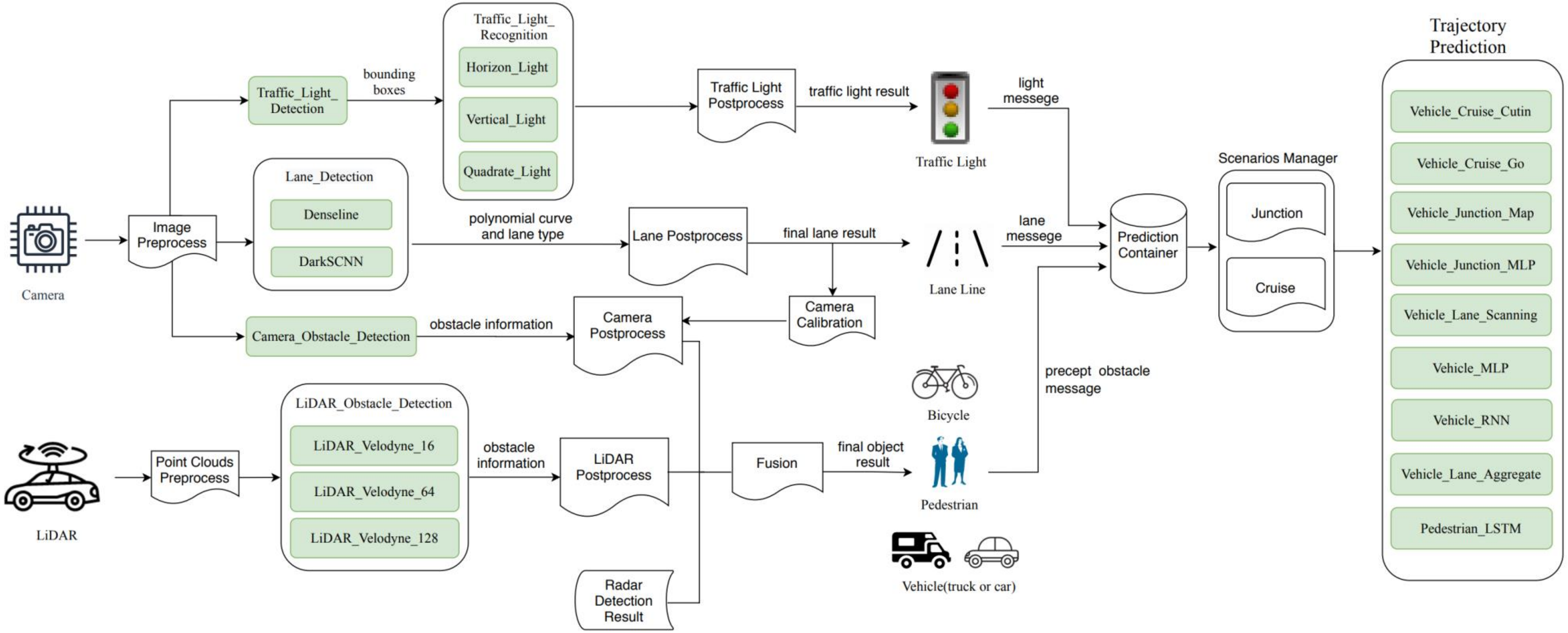


Figure 1: GFS Architecture

Scalability through redundancy and replication, no single points of failures

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

Reasoning about architecture: ML pipeline

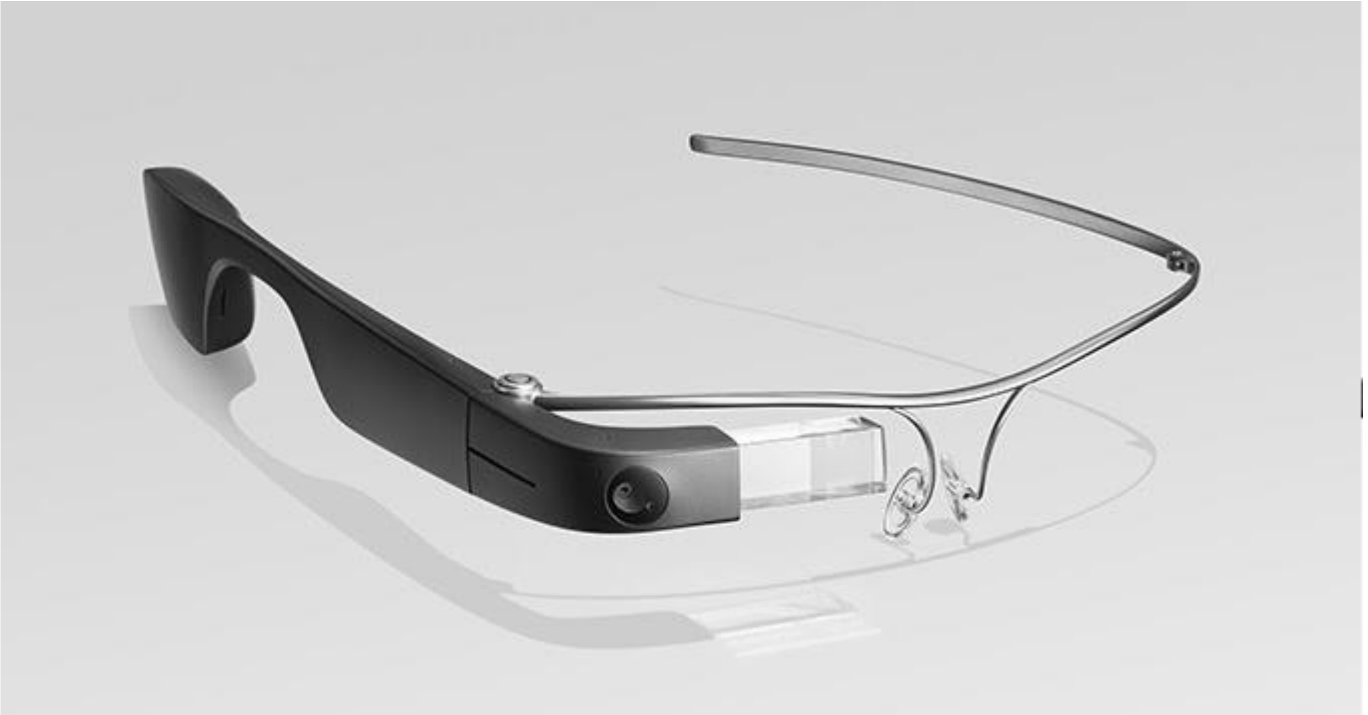


Peng, Zi, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1240-1250. 2020.

Graphical Notations for Architecture

- Notation should be suitable for the analysis
  - No single best solution!
- Meaning of elements (boxes, edges, colors, …) should be documented
  - Ideally with a legend
- Graphical notation and text are both okay
  - Can be combined, e.g., graphical overview with details as text
- Formal notations available
  - Allow verification of architecture constraints

Live exercise



Consider a translation service running embedded in glasses as augmented reality service.

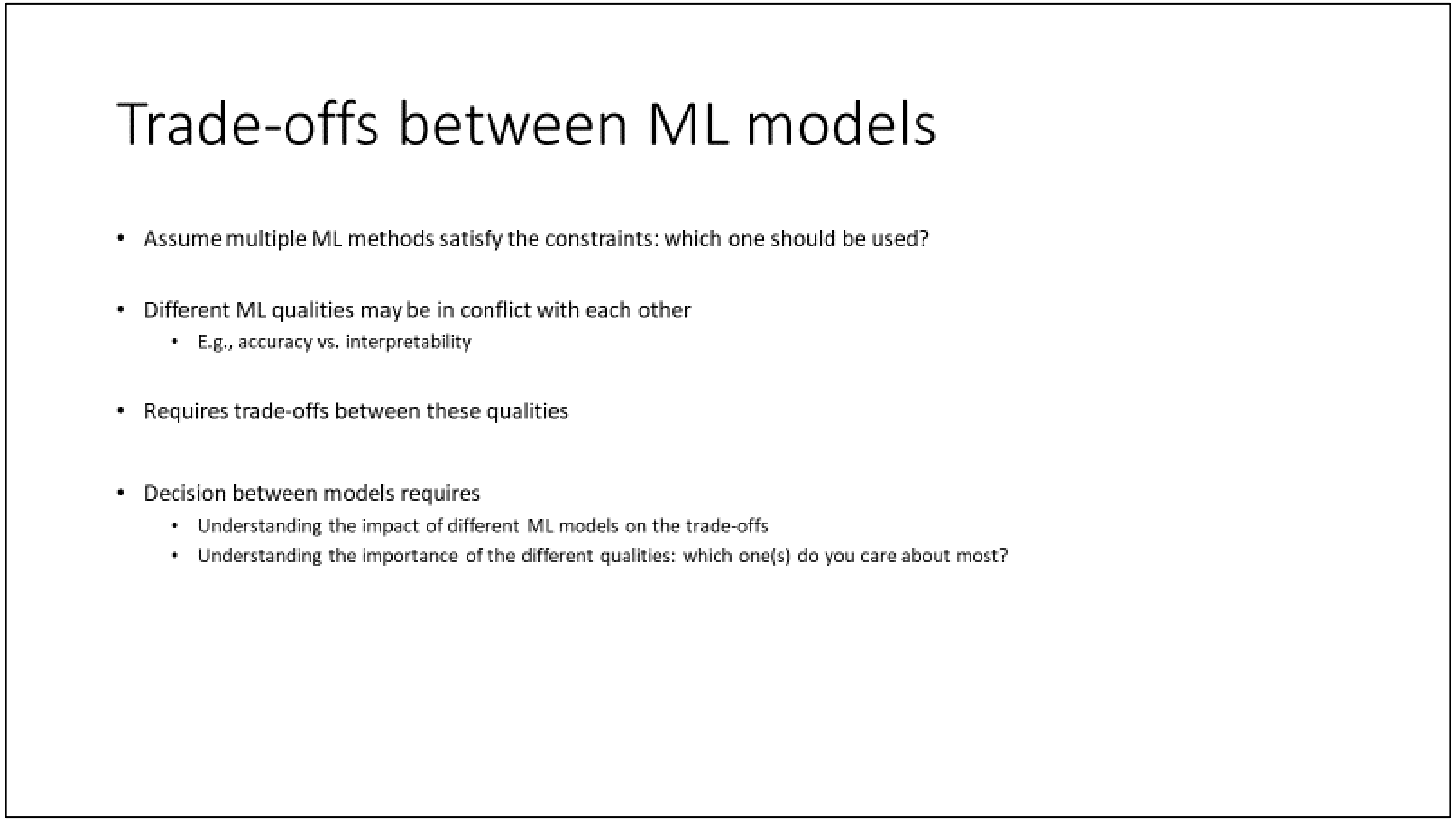
What are architectural considerations and qualities of interest?

https://www.tecnoneo.com/2020/02/las-gafas-ar-glass-2-de-google-estan.html

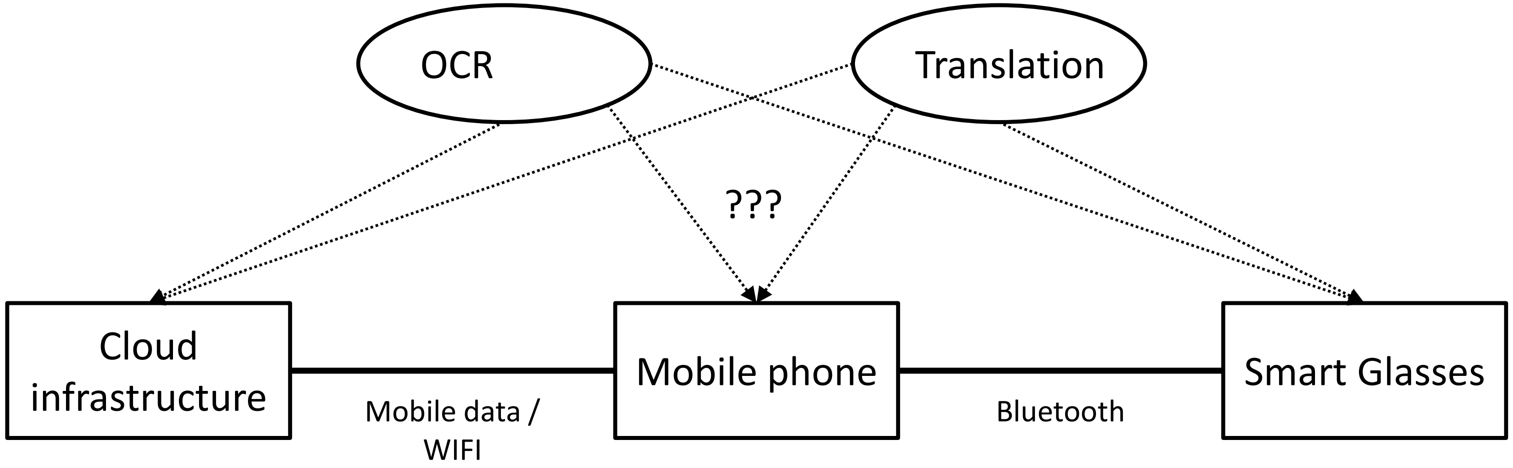
Design decisions

Which ML algorithm should be used?

- See previous chapter



Deployment architecture: Where should the model live?



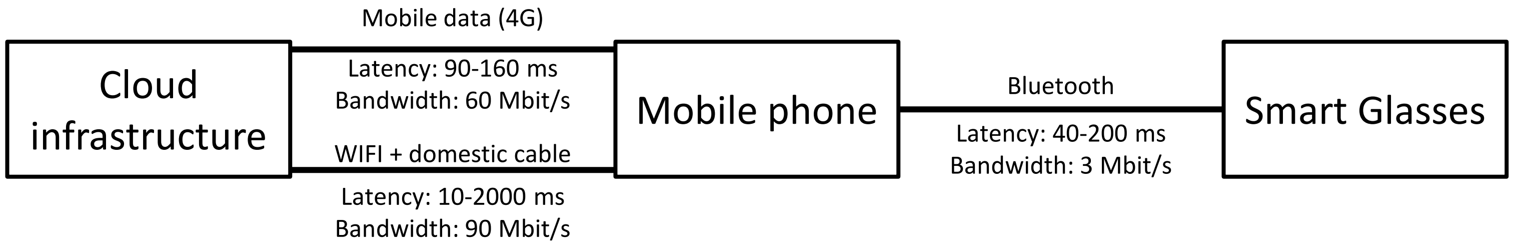
What qualities are relevant for the decision?

Considerations for deployment architecture

- How much data is needed as input for the model?
- How much output data is produced by the model?
- How fast/energy consuming is the model execution/inference?
- What latency is needed for the application?
- How big is the model? How often does it need to be updated?
- What is the cost of operating the model (distribution & execution)?
- What are opportunities for telemetry?
- What happens if users are offline?
- …

For the AR use case

- Some important data
  - 200 ms latency is notable as a speech pause
  - 20 ms is perceivable as video delay
  - 10 ms as haptic delay
  - 5 ms as cybersickness threshold for VR – 20 ms sometimes acceptable
  - 5 megapixel camera, 640x360 pixel screen, up to 2 GB ram, 16 GB storage



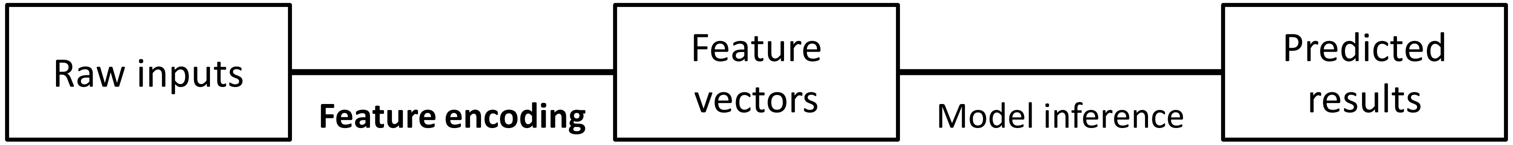
Where to run OCR? Where to run translation?

Possible locations for intelligence

- Static intelligence in product
  - Difficult to update
  - Offline operation, low execution latency
  - Cheap operation
  - No telemetry
- Client-side intelligence
  - Updates costly/slow, out-of-sync problems
  - Complexity in clients
  - Offline operation, low execution latency
- Server-centric intelligence
  - Latency in model execution (remote calls)
  - Easy to update and experiment
  - Operation costs and no offline operation
- Back-end cached intelligence
  - Precompute common results
  - Fast execution, partially offline
  - Saves bandwidth, complicated updates

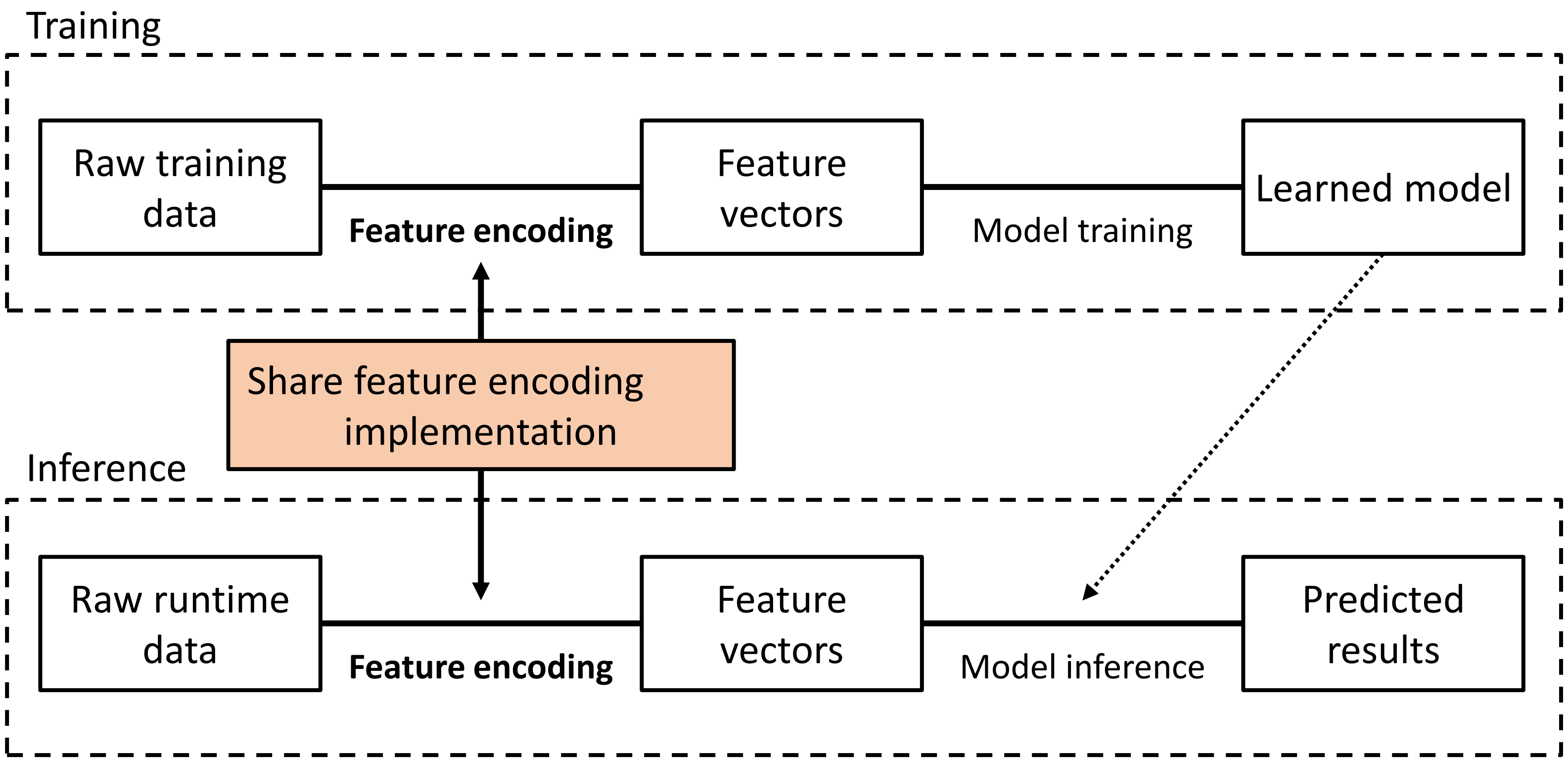
… hybrid models also possible

Where should feature encoding happen?



Server side? Client side? What are the trade-offs?

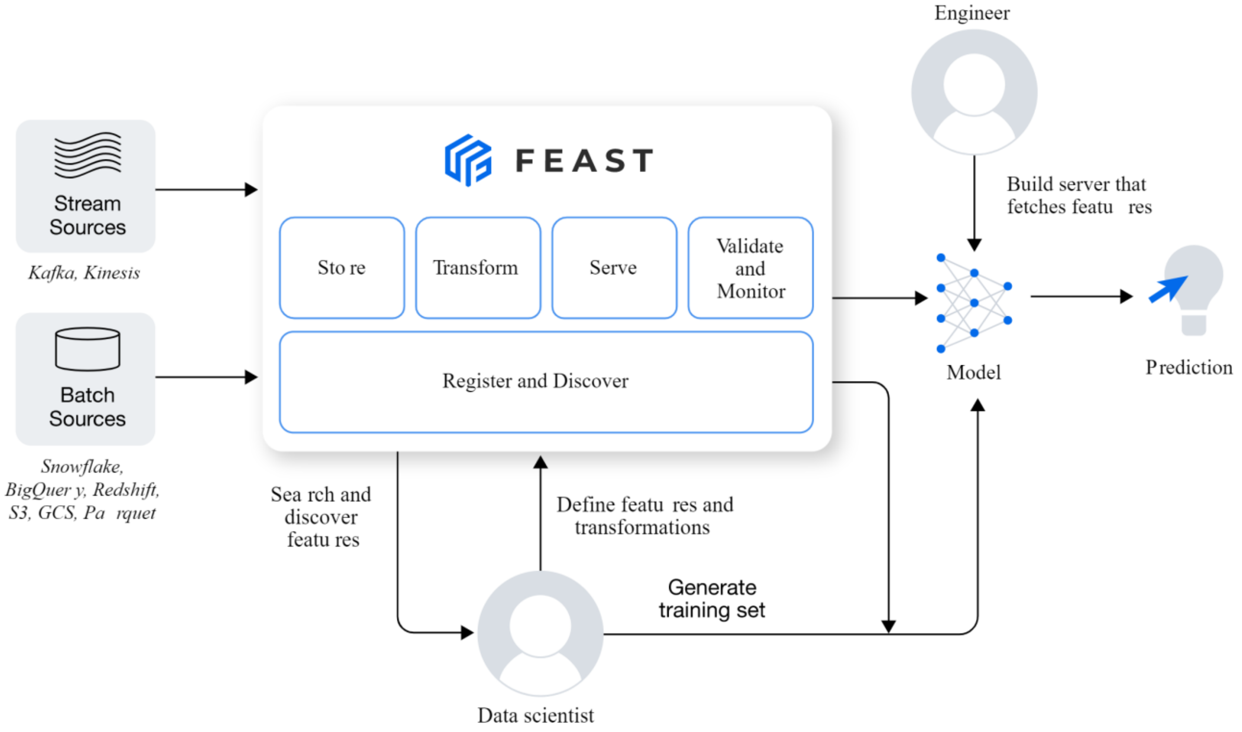
Reusing feature engineering code



Avoids *training-serving skew* of features

Feature store pattern

- Central place to store, version, and describe feature engineering code
- Can be re-used across projects
- Possible caching of expensive features
- Examples:
    - Feast, Tecton, AWS SageMaker Feature Store, …

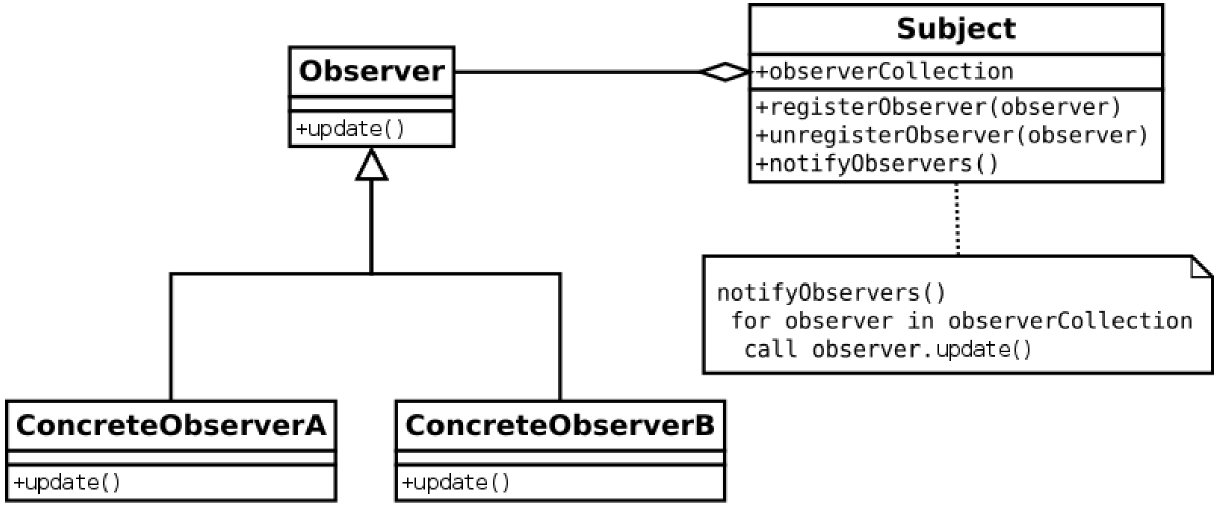


More design considerations

- Coupling of ML pipeline parts
- Coupling with other parts of the system
- Ability for developers and analysts to collaborate
- Support online experiments
- Ability to monitor
- Redundancy for availability
- Load balancing for scalability
- Isolation of mistakes for error handling
- Logging and log analysis
- …

Design pattern for systems with ML components

Design patterns are codified design knowledge

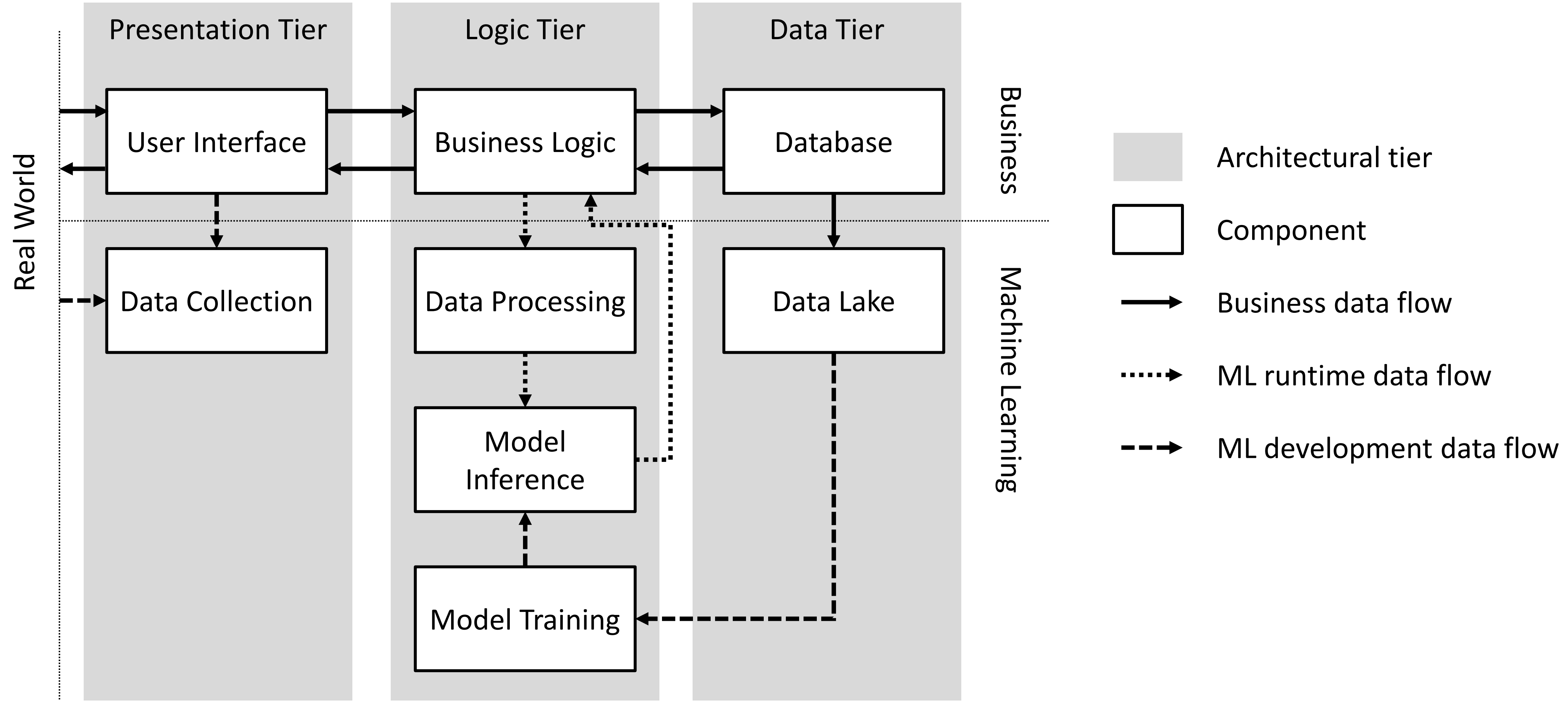


Example: Observer pattern to decouple observers from subjects

https://commons.wikimedia.org/wiki/File:Observer_w_update.svg

Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). Elements of reusable object-oriented software. *Design Patterns*.

Multi-tier architecture: Separating models and business logic

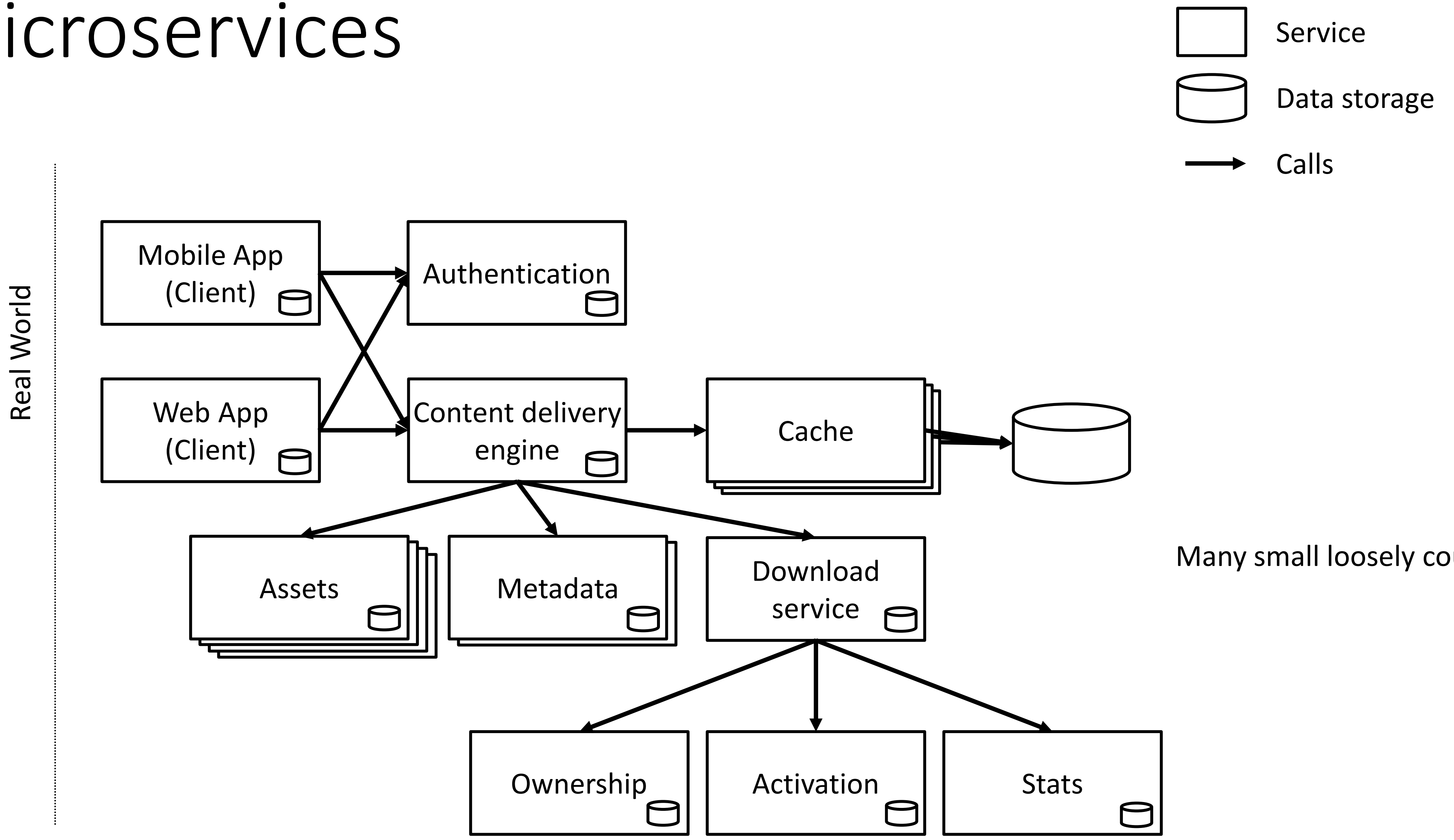


Based on: Yokoyama, Haruki. "Machine learning system architectural pattern for improving operational stability." In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 267-274. IEEE, 2019.

Advantages of separating logic

- Clearly divide responsibilities
- Allows (mostly) independent and parallel work
  - Assumes stable interfaces!
- Allows planning for location of non-ML safeguards
- Shows where ML processing logic is required

Microservices



Many small loosely coupled services

More patterns

- Stateless/serverless serving function pattern
- Feature store pattern
- Batched/precomputed serving pattern
- Two-phase prediction pattern
- Batch serving pattern
- Decoupling-training-from-serving pattern
- …

Based on: Yokoyama, Haruki. "Machine learning system architectural pattern for improving operational stability." In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 267-274. IEEE, 2019.

Anti-patterns (things to avoid)

- Big ass script architecture
- Dead experimental code paths
- Glue code
- Multiple language smell
- Pipeline jungles
- Plain-old datatype smell
- Undeclared consumers

Sculley, David, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. "Hidden technical debt in machine learning systems." In Advances in neural information processing systems, pp. 2503-2511. 2015.

Yokoyama, Haruki. "Machine learning system architectural pattern for improving operational stability." In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 267-274. IEEE, 2019.

Documenting model inference interfaces

Reasons for documentation

- Model interference between teams
  - Data scientists developing the model
  - Other data scientists using the model, evolving the model
  - Software engineers integrating the model as a component
  - Operators managing model deployment

Communicates required knowledge between teams

Documenting input/output types

```
{
  "mid": string,
  "languageCode": string,
  "name": string,
  "score": number,
  "boundingPoly": {
    object (BoundingPoly)
  }
}
```

Documentation of the output types of Google‘s public object detection API

Beyond input/output types

- Intended use cases, model capabilities and limitations
- Supported target distributions
- Accuracy
  - Ideally various measures, including data on slices and for fairness
- Latency, throughput, availability
  - This information is required for Service Level Agreements (SLAs) of served models
- Model qualities such as explainability, robustness, calibration
- Ethical considerations
  - Fairness, safety, security, privacy, …

Live exercise: What would you describe for an OCR model?

Model cards

- Proposal and template for documentation from Google
- 1-2 page summary
- Focused on fairness
- Includes
  - Intended use and scope (incl. out of scope)
  - Training and evaluation data
  - Considered demographic factors
  - Accuracy evaluations
  - Ethical considerations
- Production example
  - https://modelcards.withgoogle.com/object-detection
- Similar approach used by Hugging Face

Model Card - Toxicity in Text

Model Details

- The TOXICITY classifier provided by Perspective API [32], trained to predict the likelihood that a comment will be perceived as toxic.
- Convolutional Neural Network.
- Developed by Jigsaw in 2017.

Intended Use

- Intended to be used for a wide range of use cases such as supporting human moderation and providing feedback to comment authors.
- Not intended for fully automated moderation.
- Not intended to make judgments about specific individuals.

Factors

- Identity terms referencing frequently attacked groups, focusing on sexual orientation, gender identity, and race.

Metrics

- Pinned AUC, as presented in [11], which measures threshold-agnostic separability of toxic and non-toxic comments for each group, within the context of a background distribution of other groups.

Ethical Considerations

- Following [31], the Perspective API uses a set of values to guide their work. These values are Community, Transparency, Inclusivity, Privacy, and Topic-neutrality. Because of privacy considerations, the model does not take into account user history when making judgments about toxicity.

Training Data

- Proprietary from Perspective API. Following details in [11] and [32], this includes comments from a online forums such as Wikipedia and New York Times, with crowdsourced labels of whether the comment is "toxic".
- "Toxic" is defined as "a rude, disrespectful, or unreasonable comment that is likely to make you leave a discussion."

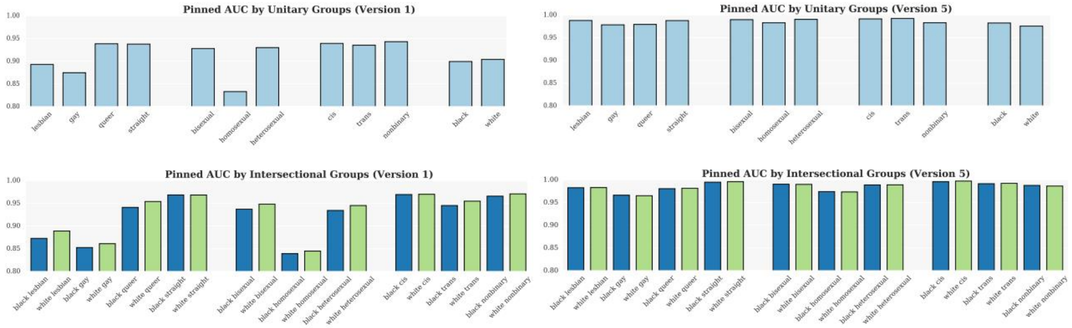
Evaluation Data

- A synthetic test set generated using a template-based approach, as suggested in [11], where identity terms are swapped into a variety of template sentences.
- Synthetic data is valuable here because [11] shows that real data often has disproportionate amounts of toxicity directed at specific groups. Synthetic data ensures that we evaluate on data that represents both toxic and non-toxic statements referencing a variety of groups.

Caveats and Recommendations

- Synthetic test data covers only a small set of very specific comments. While these are designed to be representative of common use cases and concerns, it is not comprehensive.

Quantitative Analyses



Mitchell, Margaret, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. "Model cards for model reporting." In *Proceedings of the conference on fairness, accountability, and transparency*, pp. 220-229. 2019.

FactSheet

- Proposal and template for documentation by IBM
- Intended to communicate intended qualities and assurances
- Longer list of criteria, including
    - Service intention
    - Technical description
    - Intended use
    - Target distribution
    - Own and third-party evaluation results
    - Safety and fairness considerations
    - Explainability
    - Preparation for drift and evolution
    - Security
    - Lineage and versioning

Arnold, Matthew, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, Karthikeyan Natesan Ramamurthy, Darrell Reimer, Alexandra Olteanu, David Piorkowski, Jason Tsay, and Kush R. Varshney. "FactSheets: Increasing trust in AI services through supplier's declarations of conformity." IBM Journal of Research and Development 63, no. 4/5 (2019): 6-1.

Questions?