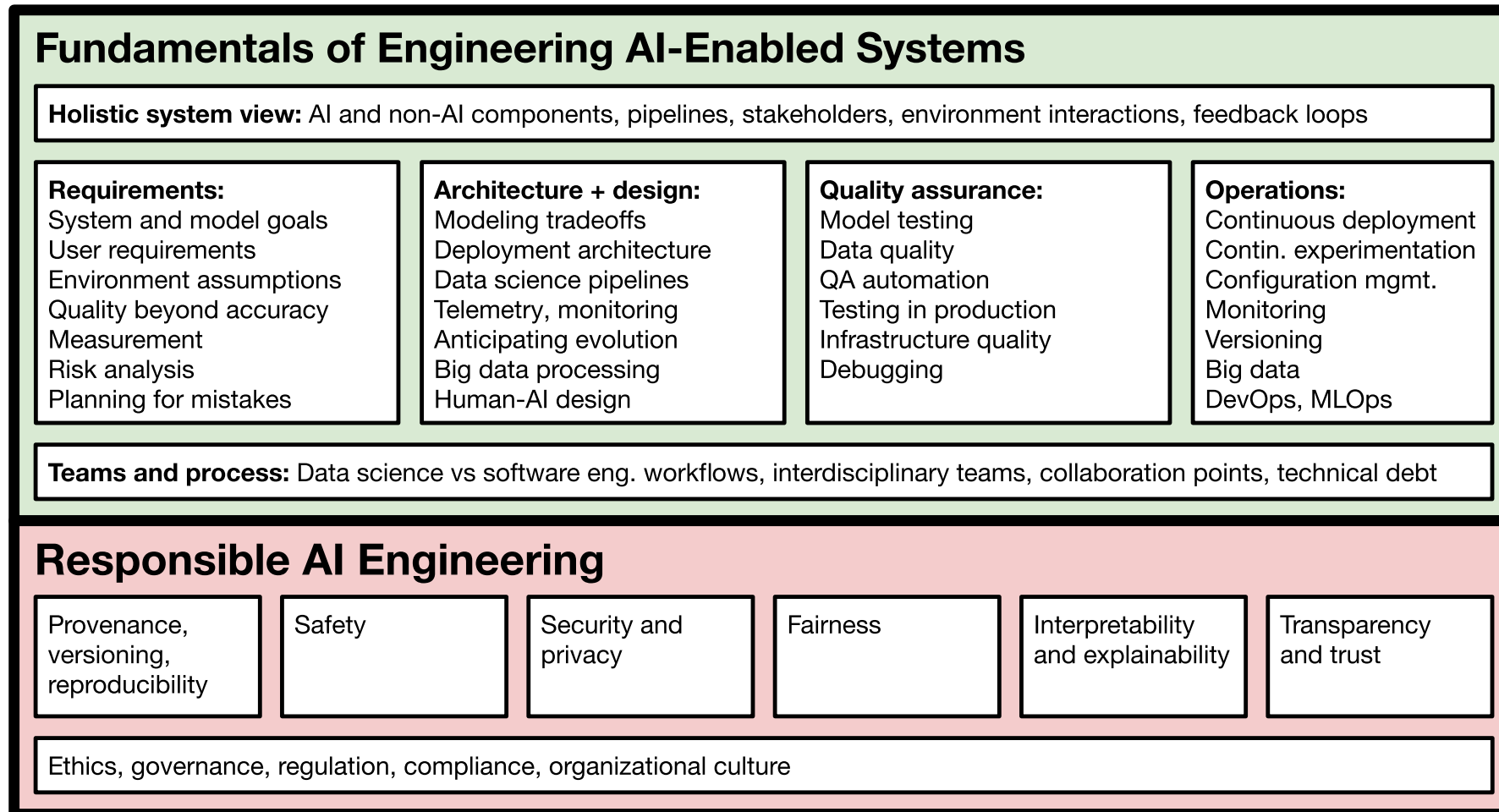


# Machine Learning in Production Versioning, Provenance, and Reproducibility



# More Foundational Technology for Responsible Engineering



# Readings

## Required readings

- Hulten, Geoff. "[Building Intelligent Systems: A Guide to Machine Learning Engineering.](#)" Apress, 2018, Chapter 21 (Organizing Intelligence).
- Halevy, Alon, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. [Goods: Organizing google's datasets.](#) In Proceedings of the 2016 International Conference on Management of Data, pp. 795-806. ACM, 2016.

# Learning Goals

- Judge the importance of data provenance, reproducibility and explainability for a given system
- Create documentation for data dependencies and provenance in a given system
- Propose versioning strategies for data and models
- Design and test systems for reproducibility



# Case Study: Credit Scoring



DHH

@dhh · [Follow](#)



The [@AppleCard](#) is such a fucking sexist program. My wife and I filed joint tax returns, live in a community-property state, and have been married for a long time. Yet Apple's black box algorithm thinks I deserve 20x the credit limit she does. No appeals work.

8:34 PM · Nov 7, 2019



[Read the full conversation on Twitter](#)



25.8K



Reply



Copy link

[Read 1.3K replies](#)







**DHH**  · Nov 8, 2019



@dhh · [Follow](#)

Replying to @dhh

I wasn't even pessimistic to expect this outcome, but here we are: [@AppleCard](#) just gave my wife the VIP bump to match my credit limit, but continued to be an utter fucking failure of a customer service experience. Let me explain...



**DHH** 

@dhh · [Follow](#)

She spoke to two Apple reps. Both very nice, courteous people representing an utterly broken and reprehensible system. The first person was like "I don't know why, but I swear we're not discriminating, IT'S JUST THE ALGORITHM". I shit you not. "IT'S JUST THE ALGORITHM!".

11:20 PM · Nov 8, 2019



4.3K



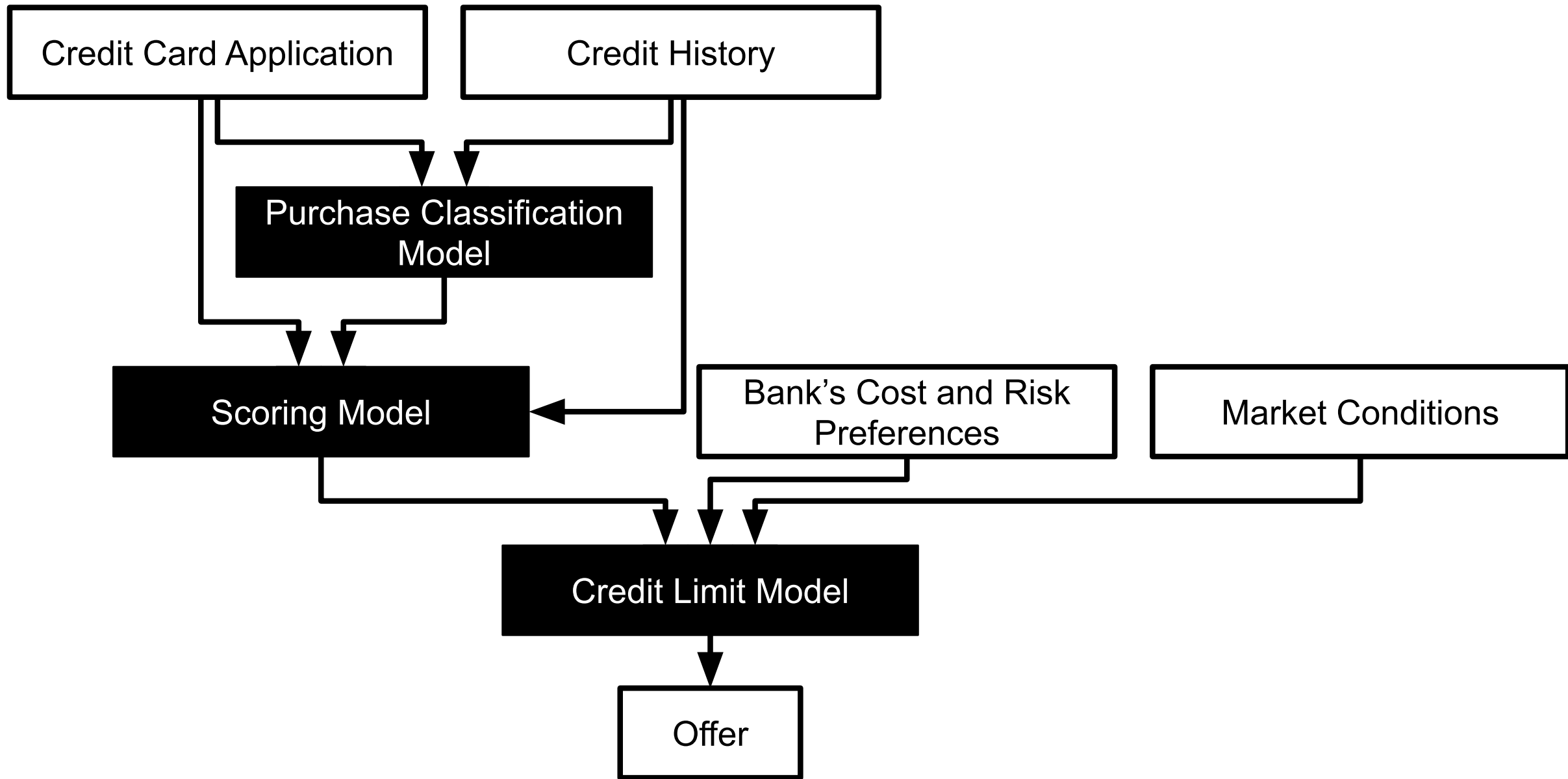
Reply



Copy link

[Read 60 replies](#)







# Debugging?

What went wrong? Where? How to fix?

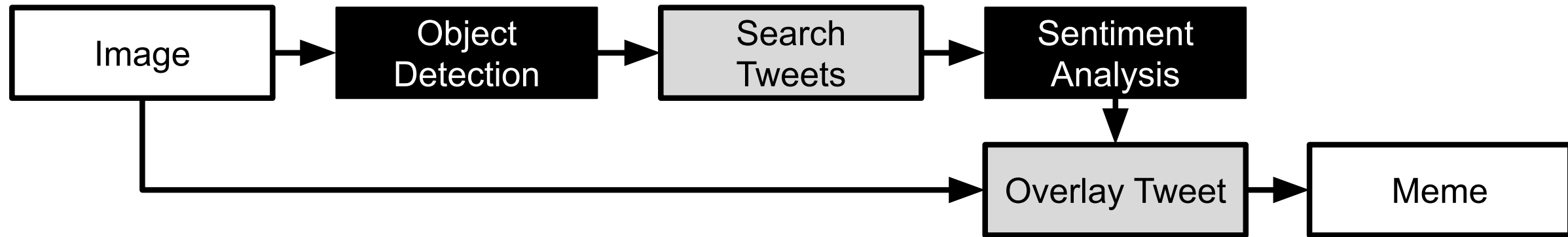


# Debugging Questions beyond Interpretability

- Can we reproduce the problem?
- What were the inputs to the model?
- Which exact model version was used?
- What data was the model trained with?
- What pipeline code was the model trained with?
- Where does the data come from? How was it processed/extracted?
- Were other models involved? Which version? Based on which data?
- What parts of the input are responsible for the (wrong) answer?
- How can we fix the model?



# Model Chaining: Automatic meme generator



*Version all models involved!*

# Complex Model Composition: ML Models for Feature Extraction

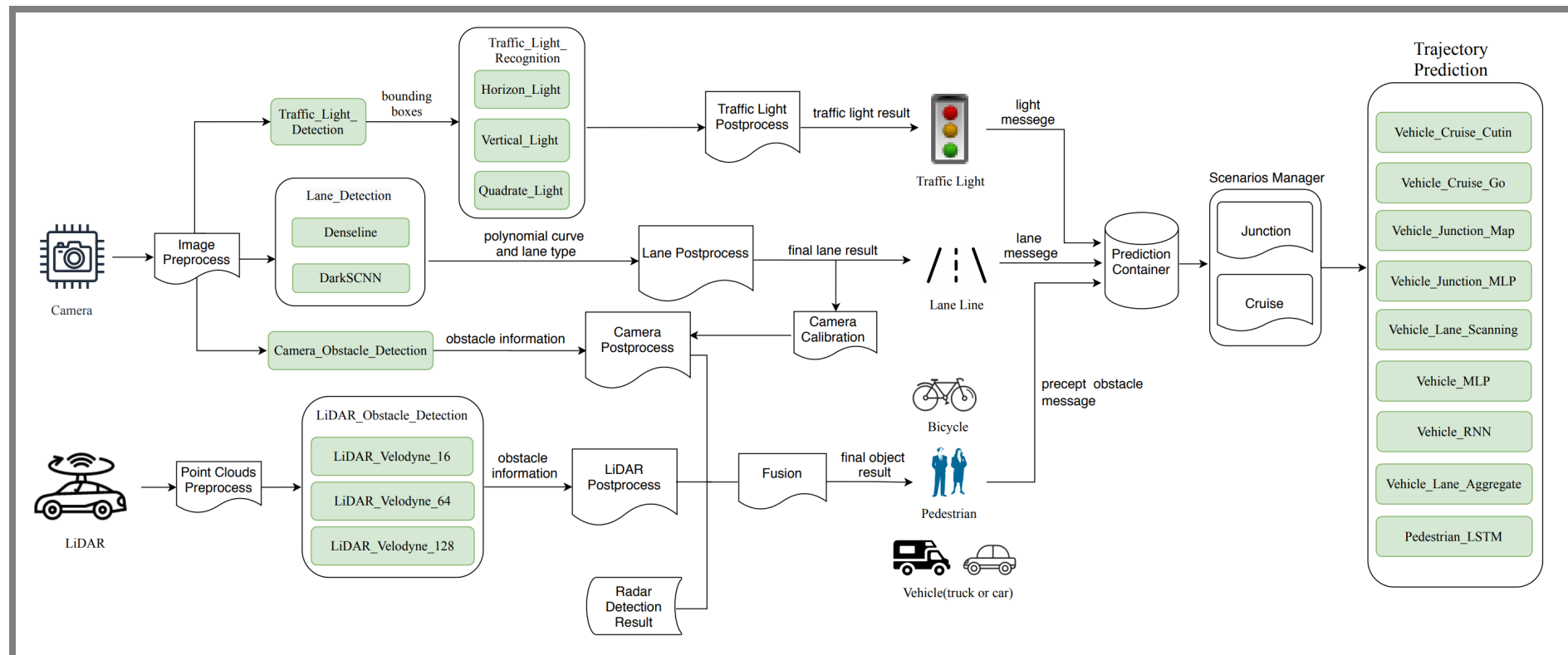


Image: Peng, Zi, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo." In Proc. FSE. 2020.



see also Zong, W., Zhang, C., Wang, Z., Zhu, J., & Chen, Q. (2018). [Architecture design and implementation of an autonomous vehicle](#). IEEE access, 6, 21956-21970.

# Breakout Discussion: Movie Predictions

*Assume you are receiving complains that a child gets many recommendations about R-rated movies*

In a group, discuss how you could address this in your own system and post to #lecture, tagging team members:

- How could you identify the problematic recommendation(s)?
- How could you identify the model that caused the prediction?
- How could you identify the training code and data that learned the model?
- How could you identify what training data or infrastructure code "caused" the recommendations?

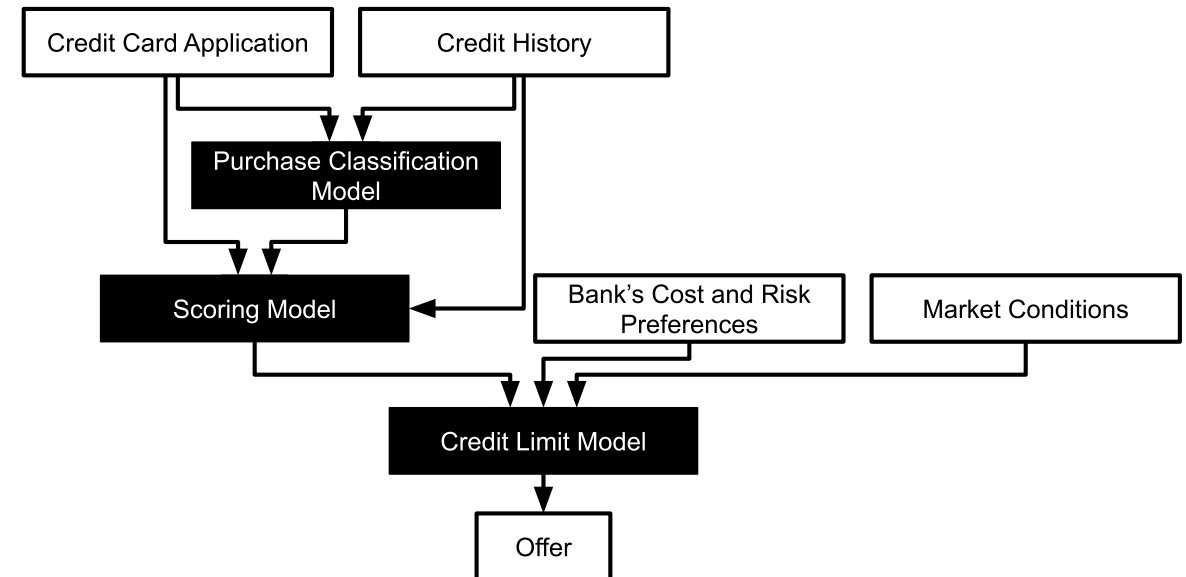
K.G Orphanides. [Children's YouTube is still churning out blood, suicide and cannibalism](#). Wired UK, 2018; Kristie Bertucci. [16 NSFW Movies Streaming on Netflix](#). Gadget Reviews, 2020

# Provenance Tracking

*Historical record of data and its origin*

# Data Provenance

- Track origin of all data
  - Collected where?
  - Modified by whom, when, why?
  - Extracted from what other data or model or algorithm?
- ML models often based on data driven from many sources through many steps, including other models





# Excursion: Provenance Tracking in Databases

Whenever value is changed, record:

- who changed it
- time of change
- history of previous values
- possibly also justification of why

Embedded as feature in some databases or implemented in business logic

Possibly signing with cryptographic methods



# Tracking Data Lineage

Document all data sources

Identify all model dependencies and flows

Ideally model all data and processing code

Avoid "visibility debt"

(Advanced: Use infrastructure to automatically capture/infer dependencies and flows as in [Goods](#))

# Feature Provenance

How are features extracted from raw data?

- during training
- during inference

Has feature extraction changed since the model was trained?

Recommendation: Modularize and version feature extraction code

**Example?**

# Advanced Practice: Feature Store

Stores feature extraction code as functions, versioned

Catalog features to encourage reuse

Compute and cache features centrally

Use same feature used in training and inference code

Advanced: Immutable features -- never change existing features, just add new ones (e.g., creditscore, creditscore2, creditscore3)

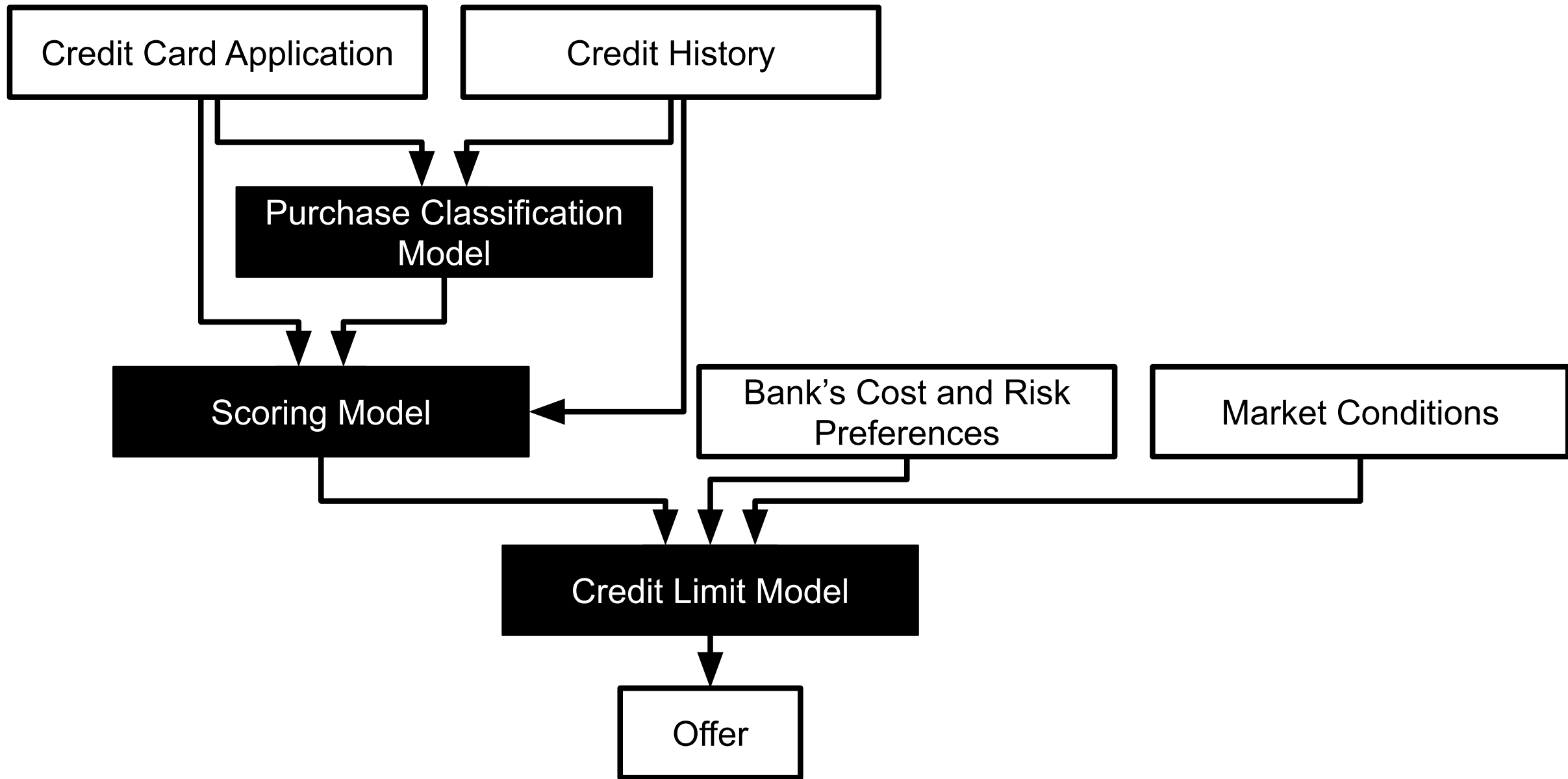
# Model Provenance

How was the model trained?

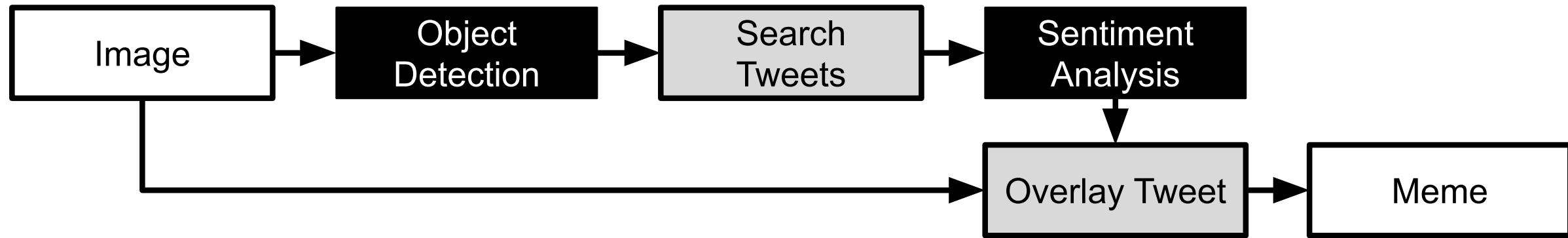
What data? What library? What hyperparameter? What code?

Ensemble of multiple models?





# In Real Systems: Tracking Provenance Across Multiple Models



Example adapted from Jon Peck. [Chaining machine learning models in production with Algorithmia.](#)  
Algorithmia blog, 2019

# Complex Model Composition: ML Models for Feature Extraction

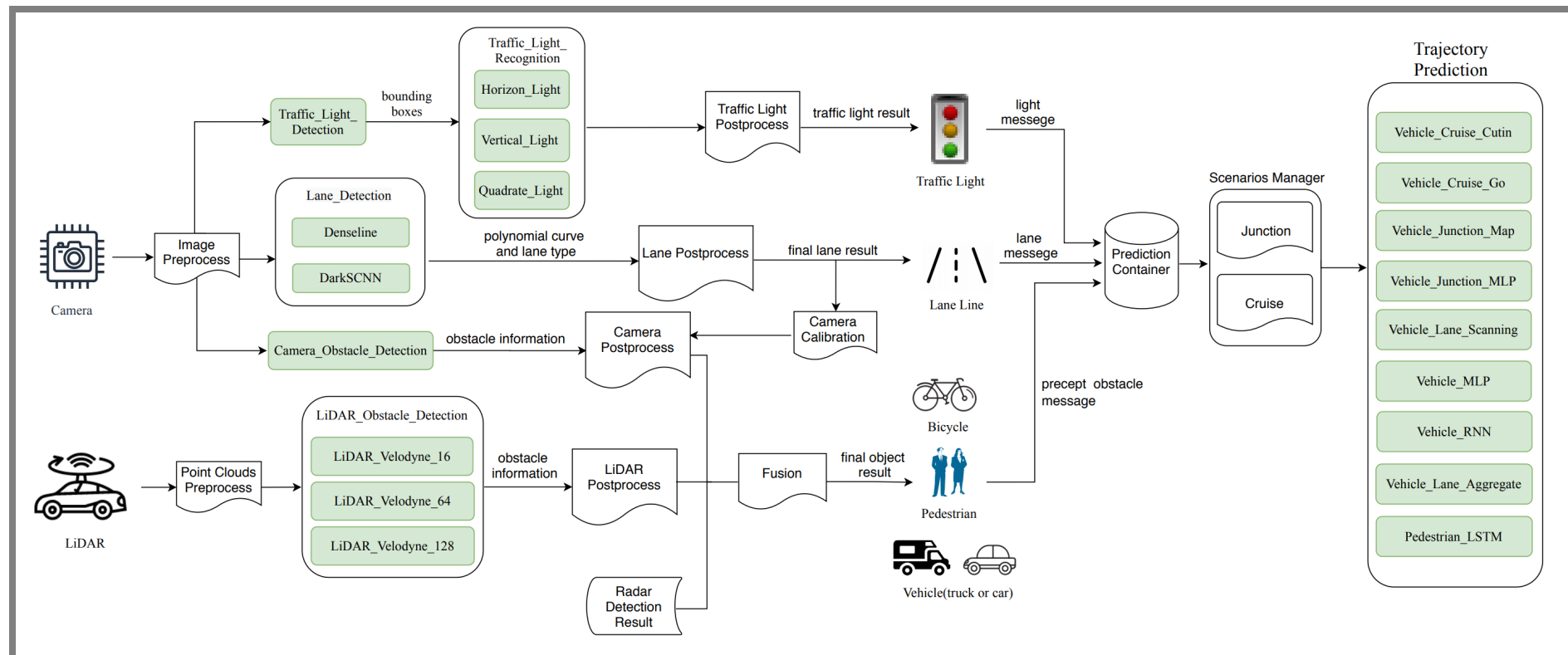


Image: Peng, Zi, Jinqiu Yang, Tse-Hsun Chen, and Lei Ma. "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo." In Proc. FSE. 2020.

# Summary: Provenance

Data provenance

Feature provenance

Model provenance

# Practical Data and Model Versioning



# How to Version Large Datasets?



(movie ratings, movie metadata, user data?)

# Recall: Event Sourcing

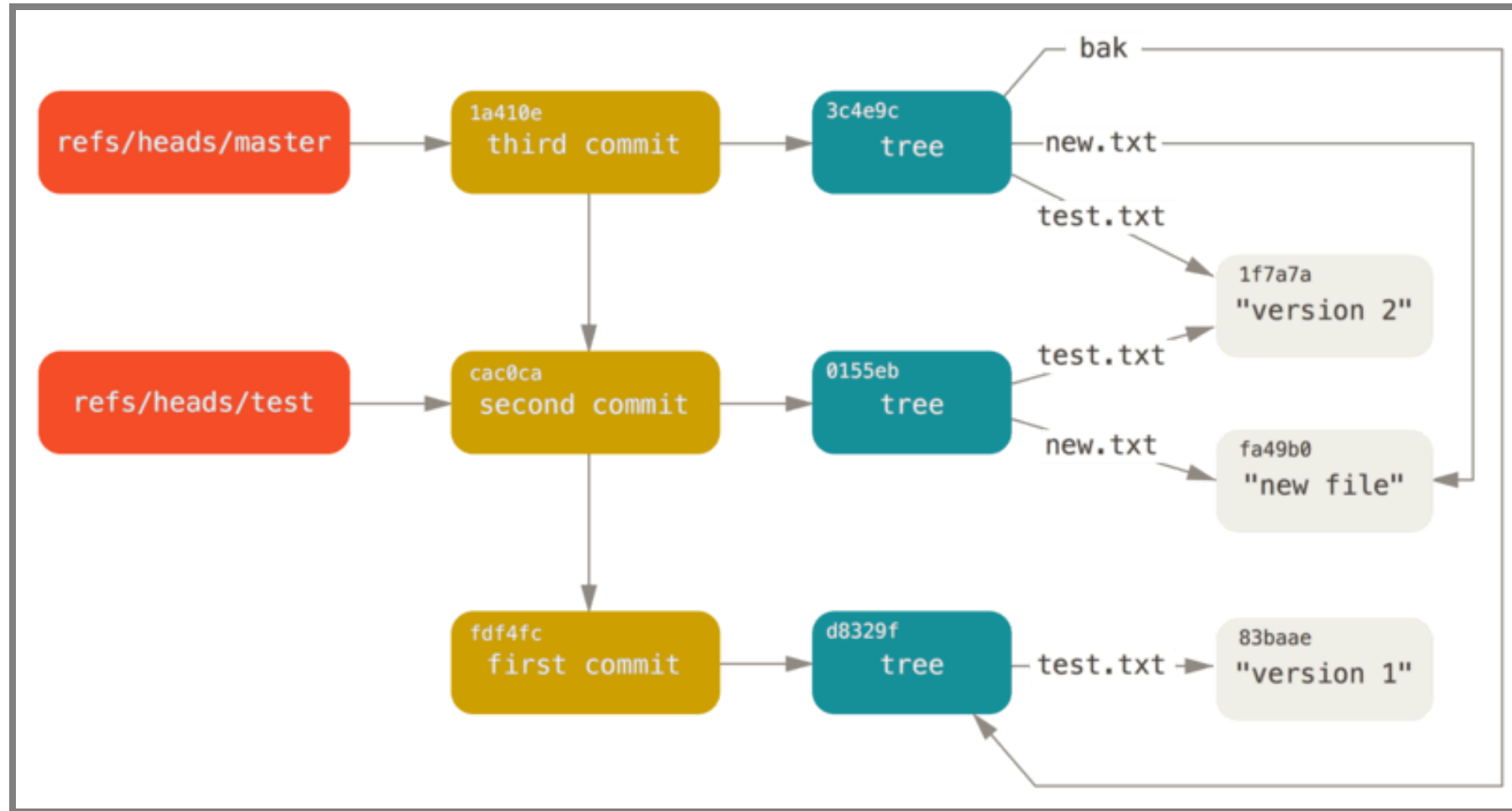
- Append only databases
- Record edit events, never mutate data
- Compute current state from all past events, can reconstruct old state
- For efficiency, take state snapshots
- Similar to traditional database logs

```
createUser(id=5, name="Christian", dpt="SCS")  
updateUser(id=5, dpt="ISR")  
deleteUser(id=5)
```

# Versioning Strategies for Datasets

1. Store copies of entire datasets (like Git), identify by checksum
2. Store deltas between datasets (like Mercurial)
3. Offsets in append-only database (like Kafka), identify by offset
4. History of individual database records (e.g. S3 bucket versions)
  - some databases specifically track provenance (who has changed what entry when and how)
  - specialized data science tools eg [Hangar](#) for tensor data
5. Version pipeline to recreate derived datasets ("views", different formats)
  - e.g. version data before or after cleaning?

# Aside: Git Internals



# Versioning Models



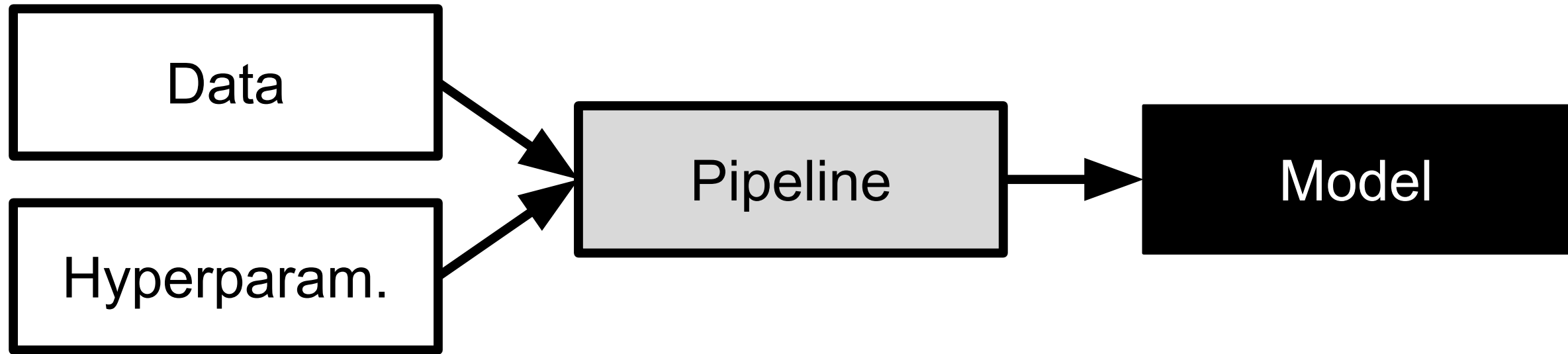


# Versioning Models

Usually no meaningful delta/compression, version as binary objects

Any system to track versions of blobs

# Versioning Pipelines



Associate model version with pipeline code version, data version, and hyperparameters!

# Versioning Dependencies

Pipelines depend on many frameworks and libraries

Ensure reproducible builds

- Declare versioned dependencies from stable repository (e.g. requirements.txt + pip)
- Avoid floating versions
- Optionally: commit all dependencies to repository ("vendoring")

Optionally: Version entire environment (e.g. Docker container)

Test build/pipeline on independent machine (container, CI server, ...)

# ML Versioning Tools (MLOps)

Tracking data, pipeline, and model versions

Modeling pipelines: inputs and outputs and their versions

- automatically tracks how data is used and transformed

Often tracking also metadata about versions

- Accuracy
- Training time
- ...

# Example: DVC

```
dvc add images
dvc run -d images -o model.p cnn.py
dvc remote add myrepo s3://mybucket
dvc push
```

- Tracks models and datasets, built on Git
- Splits learning into steps, incrementalization
- Orchestrates learning in cloud resources

<https://dvc.org/>

# DVC Example

```
stages:  
  features:  
    cmd: jupyter nbconvert --execute featurize.ipynb  
    deps:  
      - data/clean  
    params:  
      - levels.no  
    outs:  
      - features  
    metrics:  
      - performance.json
```

# Experiment Tracking

Log information within pipelines: hyperparameters used, evaluation results, and model files

The screenshot displays the MLflow web interface for an experiment titled "Listing Price Prediction". At the top, the MLflow logo is on the left, and "Github" and "Docs" links are on the right. Below the title, the "Experiment ID" is 0 and the "Artifact Location" is /Users/matei/mlflow/demo/mlruns/0. There are search and filter controls: "Search Runs" with the query "metrics.R2 > 0.24", "Filter Params" with "alpha, lr", and "Filter Metrics" with "rmse, r2". Below these controls, it indicates "4 matching runs" and provides buttons for "Compare Selected" and "Download CSV". A table lists the runs with columns for Time, User, Source, Version, Parameters (alpha, l1\_ratio), and Metrics (MAE, R2, RMSE). Each row includes a checkbox for selection and horizontal bar charts for the MAE, R2, and RMSE values.

	Time	User	Source	Version	Parameters		Metrics		
					alpha	l1_ratio	MAE	R2	RMSE
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.5	0.2	84.27	0.277	158.1
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.2	0.5	84.08	0.264	159.6
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0.5	0.5	84.12	0.272	158.6
<input type="checkbox"/>	17:37	matei	linear.py	3a1995	0	0	84.49	0.249	161.2

Many tools: MLflow, ModelDB, Neptune, TensorBoard, Weights & Biases, Comet.ml, ...





Image from Matei Zaharia. [Introducing MLflow: an Open Source Machine Learning Platform](#), 2018



# ModelDB Example

```
from verta import Client
client = Client("http://localhost:3000")

proj = client.set_project("My first ModelDB project")
expt = client.set_experiment("Default Experiment")

# log the first run
run = client.set_experiment_run("First Run")
run.log_hyperparameters({"regularization" : 0.5})
run.log_dataset_version("training_and_testing_data", dataset_v
model1 = # ... model training code goes here
```

# Google's Goods

Automatically derive data dependencies from system log files

Track metadata for each table

No manual tracking/dependency declarations needed

Requires homogeneous infrastructure

Similar systems for tracking inside databases, MapReduce, Sparks, etc.

# From Model Versioning to Deployment

Decide which model version to run where

- automated deployment and rollback (cf. canary releases)
- Kubernetes, Cortex, BentoML, ...

Track which prediction has been performed with which model version (logging)

# Logging and Audit Traces

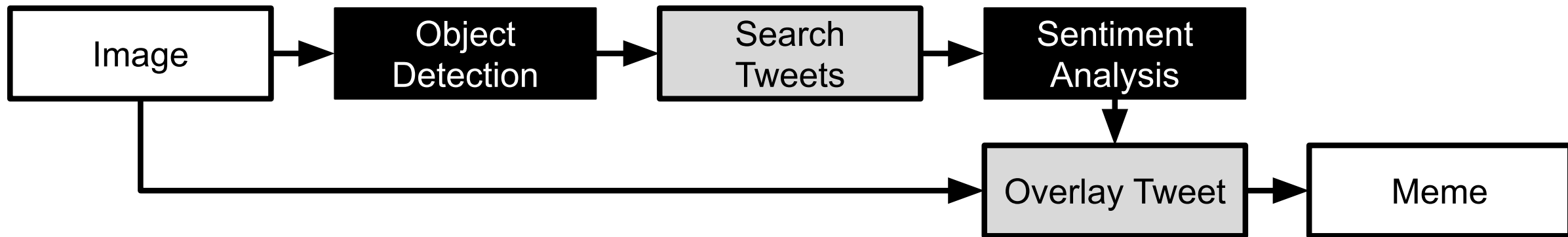
**Key goal:** If a customer complains about an interaction, can we reproduce the prediction with the right model? Can we debug the model's pipeline and data? Can we reproduce the model?

- Version everything
- Record every model evaluation with model version
- Append only, backed up

```
<date>, <model>, <model version>, <feature inputs>, <output>  
<date>, <model>, <model version>, <feature inputs>, <output>  
<date>, <model>, <model version>, <feature inputs>, <output>
```

# Logging for Composed Models

*Ensure all predictions are logged*



# Breakout Discussion: Movie Predictions (Revisited)

*Assume you are receiving complains that a child gets mostly recommendations about R-rated movies*

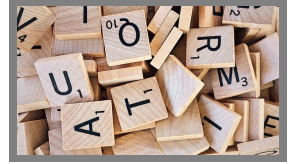
Discuss again, updating the previous post in #lecture:

- How would you identify the model that caused the prediction?
- How would you identify the code and dependencies that trained the model?
- How would you identify the training data used for that model?

# Reproducibility



# On Terminology



**Replicability:** ability to reproduce results exactly

- Ensures everything is clear and documented
- All data, infrastructure shared; requires determinism

**Reproducibility:** the ability of an experiment to be repeated with minor differences, achieving a consistent expected result

- In science, reproducing important to gain confidence
- many different forms distinguished: conceptual, close, direct, exact, independent, literal, nonexperimental, partial, retest, ...

Juristo, Natalia, and Omar S. Gómez. "[Replication of software engineering experiments.](#)" In Empirical software engineering and verification, pp. 60-88. Springer, Berlin, Heidelberg, 2010.

# "Reproducibility" of Notebooks

2019 Study of 1.4M notebooks on GitHub:

- 21% had unexecuted cells
- 36% executed cells out of order
- 14% declare dependencies
- success rate for installing dependencies <40% (version issues, missing files)
- notebook execution failed with exception in >40% (often ImportError, NameError, FileNotFoundError)
- only 24% finished execution without problem, of those 75% produced different results

2020 Study of 936 executable notebooks:

- 40% produce different results due to nondeterminism (randomness without seed)
- 12% due to time and date
- 51% due to plots (different library version, API misuse)
- 2% external inputs (e.g. Weather API)
- 27% execution environment (e.g., Python package versions)

≡ Pimentel, João Felipe, et al. "A large-scale study about quality and reproducibility of jupyter notebooks." In Proc. MSR, 2019. and Wang, Jiawei, K. U. O. Tzu-Yang, Li Li, and Andreas Zeller.

# Practical Reproducibility

Ability to generate the same research results or predictions

Recreate model from data

Requires versioning of data and pipeline (incl. hyperparameters and dependencies)

# Nondeterminism

- Model inference almost always deterministic for a given model
- Many machine learning algorithms are nondeterministic
  - Nondeterminism in neural networks initialized from random initial weights
  - Nondeterminism from distributed computing, random forests
  - Determinism in linear regression and decision trees
- Many notebooks and pipelines contain nondeterminism
  - Depend on time or snapshot of online data (e.g., stream)
  - Initialize random seed
  - Different memory addresses for figures
- Different library versions installed on the machine

# Recommendations for Reproducibility

- Version pipeline and data (see above)
- Document each step
  - document intention and assumptions of the process (not just results)
  - e.g., document why data is cleaned a certain way
  - e.g., document why certain parameters chosen
- Ensure determinism of pipeline steps (-> test)
- Modularize and test the pipeline
- Containerize infrastructure -- see MLOps

# Summary

Provenance is important for debugging and accountability

Data provenance, feature provenance, model provenance

Reproducibility vs replicability

*Version everything!*

- Strategies for data versioning at scale
- Version the entire pipeline and dependencies
- Adopt a pipeline view, modularize, automate
- Containers and MLOps, many tools

# Further Readings

- Sugimura, Peter, and Florian Hartl. “Building a Reproducible Machine Learning Pipeline.” *arXiv preprint arXiv:1810.04570* (2018).
- Chattopadhyay, Souti, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. “[What’s Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities.](#)” In Proceedings of the CHI Conference on Human Factors in Computing Systems, 2020.
- Sculley, D, et al. “[Hidden technical debt in machine learning systems.](#)” In Advances in neural information processing systems, pp. 2503–2511. 2015.

# Bonus: Debugging and Fixing Models

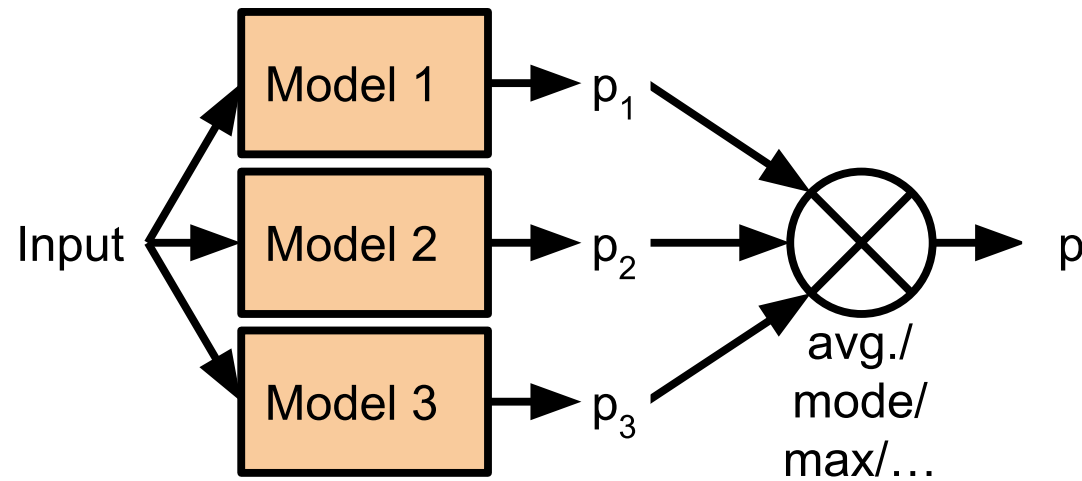
See also Hulten. Building Intelligent Systems. Chapter 21

See also Nushi, Besmira, Ece Kamar, Eric Horvitz, and Donald Kossmann. "[On human intellect and machine failures: troubleshooting integrative machine learning systems.](#)" In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 1017-1025. 2017.

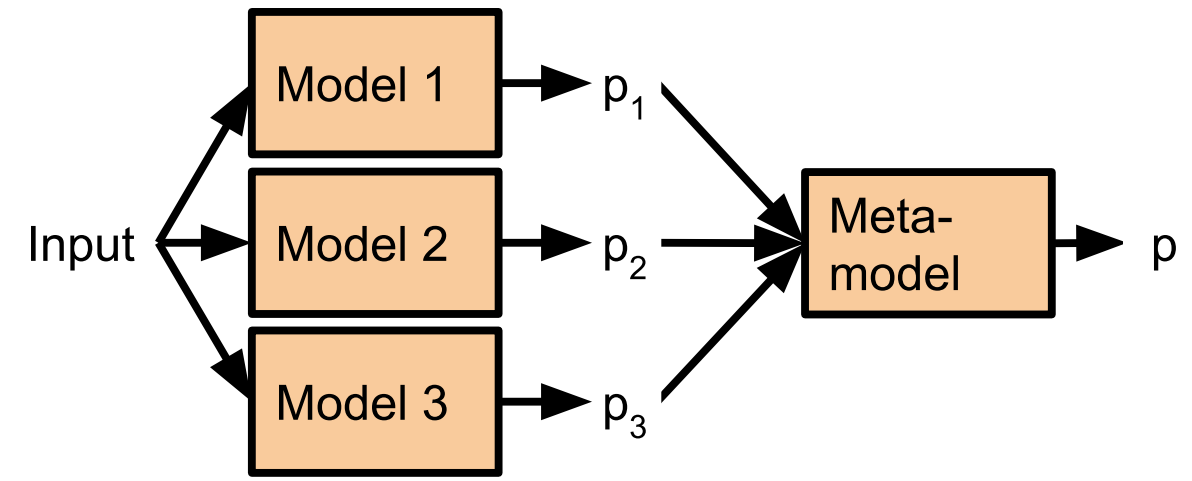


# Recall: Composing Models: Ensemble and metamodels

Ensemble

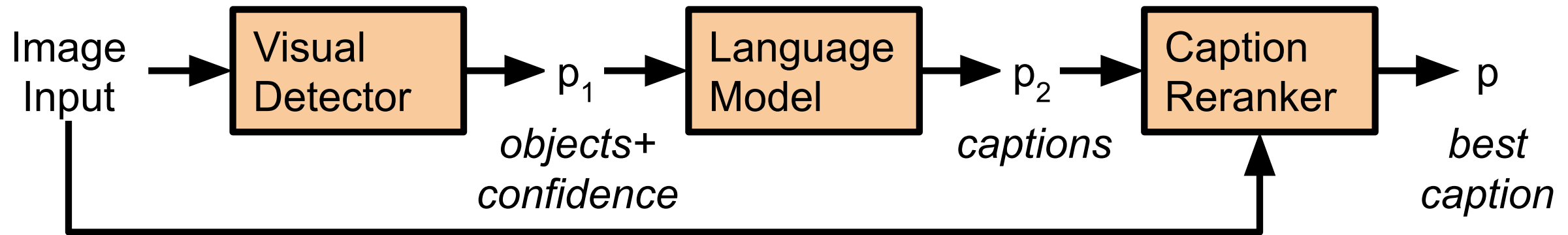


Metamodel / model stacking

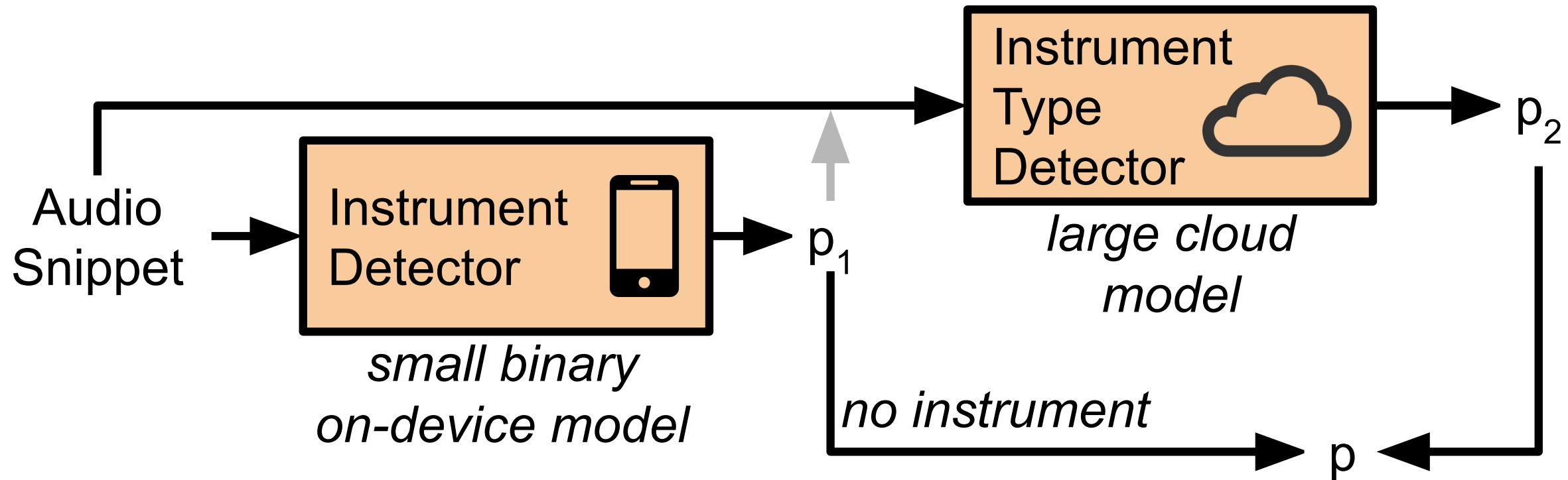


Legend:  machine-learned model,  non-ML aggregation function,  $p$  prediction

# Recall: Composing Models: Decomposing the problem, sequential



# Recall: Composing Models: Cascade/two-phase prediction

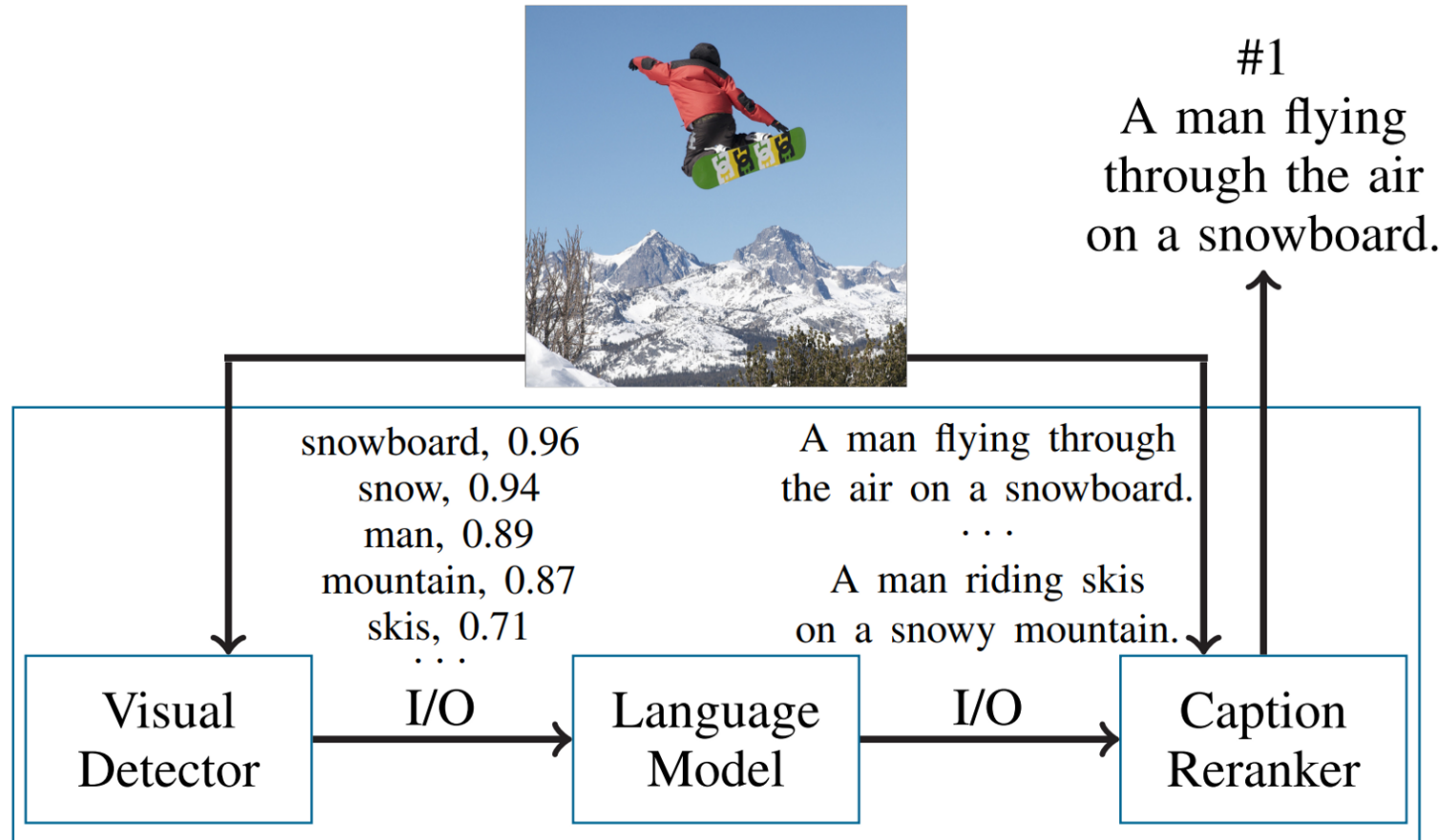


# Decomposing the Image Captioning Problem?



Using insights of how humans reason: Captions contain important objects in the image and their relations. Captions follow typical language/grammatical structure


# State of the Art Decomposition (in 2015)



Example and image from: Nushi, Besmira, Ece Kamar, Eric Horvitz, and Donald Kossmann. "On human intellect and machine failures: troubleshooting integrative machine learning systems." In Proc. AAAI. 2017.


# Blame assignment?

Visual Detector	Language Model	Caption Reranker
1. teddy 0.92	1. A teddy bear.	1. A <b>blender</b> sitting on top of a cake.
2. on 0.92	2. A stuffed bear.	2. A teddy bear <b>in front of</b> a birthday cake.
3. cake 0.90	...	3. A cake sitting on top of a <b>blender</b> .
4. bear 0.87	108. A <b>blender</b> sitting on top of a cake.	
5. stuffed 0.85		
...		
15. <b>blender</b> 0.57		



Example and image from: Nushi, Besmira, Ece Kamar, Eric Horvitz, and Donald Kossmann. "On human intellect and machine failures: troubleshooting integrative machine learning systems." In Proc. AAAI. 2017.

# Nonmonotonic errors



Visual Detector		Fixed Visual Detector	
teddy	0.92	teddy	1.0
computer	0.91	bear	1.0
bear	0.90	wearing	1.0
wearing	0.87	keyboard	1.0
keyboard	0.84	glasses	1.0
glasses	0.63		

1. A teddy bear sitting on top of a computer.

1. a person wearing glasses and holding a teddy bear sitting on top of a keyboard.

Example and image from: Nushi, Besmira, Ece Kamar, Eric Horvitz, and Donald Kossmann. "On human intellect and machine failures: troubleshooting integrative machine learning systems." In Proc. AAAI. 2017.

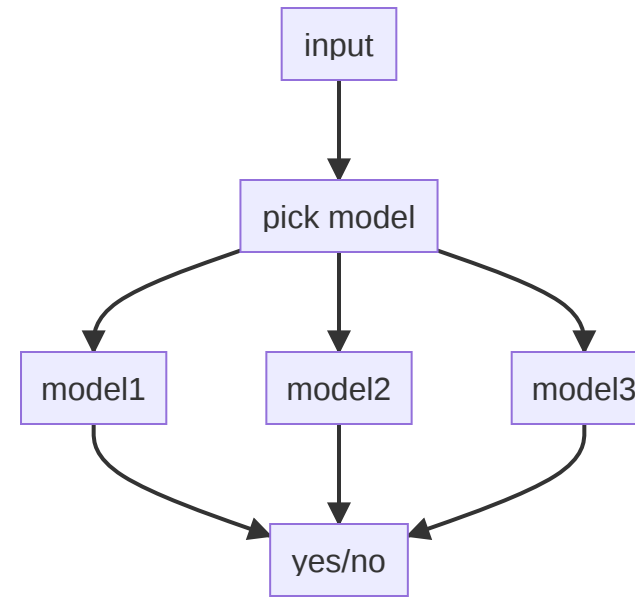


# Chasing Bugs

- Update, clean, add, remove data
- Change modeling parameters
- Add regression tests
- Fixing one problem may lead to others, recognizable only later

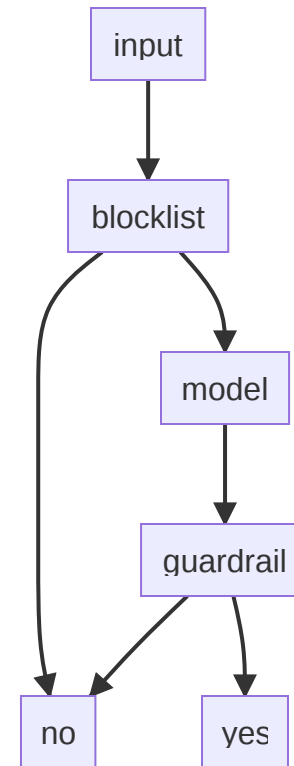
# Partitioning Contexts

- Separate models for different subpopulations
- Potentially used to address fairness issues
- ML approaches typically partition internally already



# Overrides

- Hardcoded heuristics (usually created and maintained by humans) for special cases
- Blocklists, guardrails
- Potential neverending attempt to fix special cases



# Ideas?

