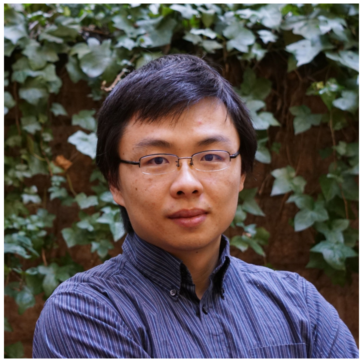
Managing ML Pipelines: Feature Stores and the Coming Wave of Embedding Ecosystems

VLDB 2021

Speakers

Laurel Orr
Stanford

Atindriyo Sanyal
Uber AI



Xiao Ling
Apple

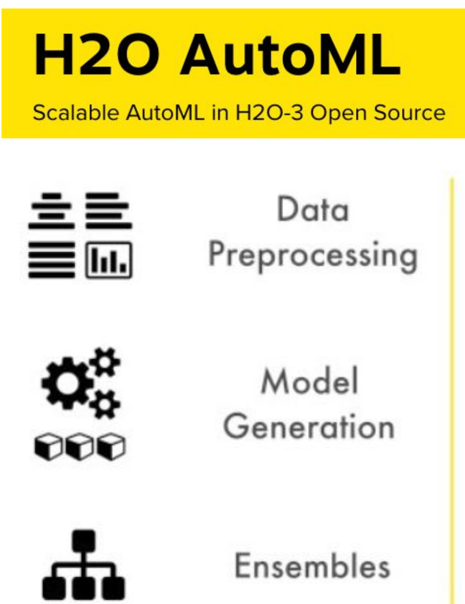
Megan Leszczynski
Stanford

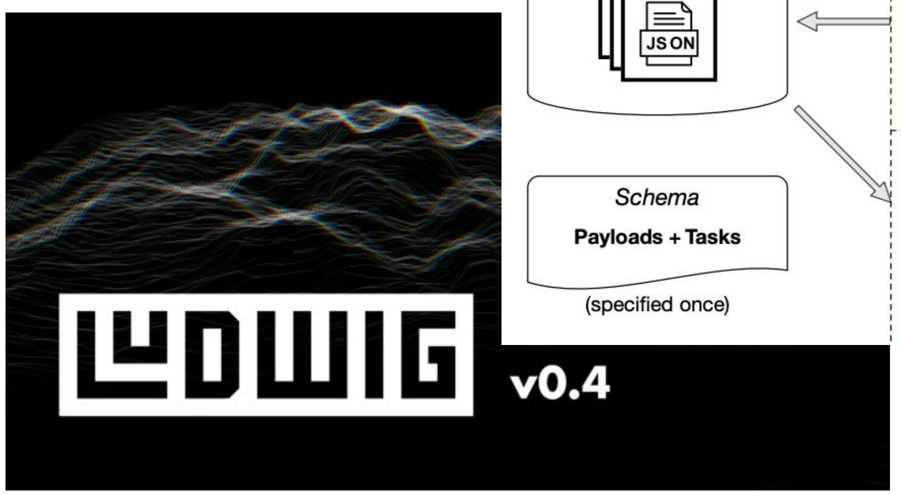
Karan Goel
Stanford

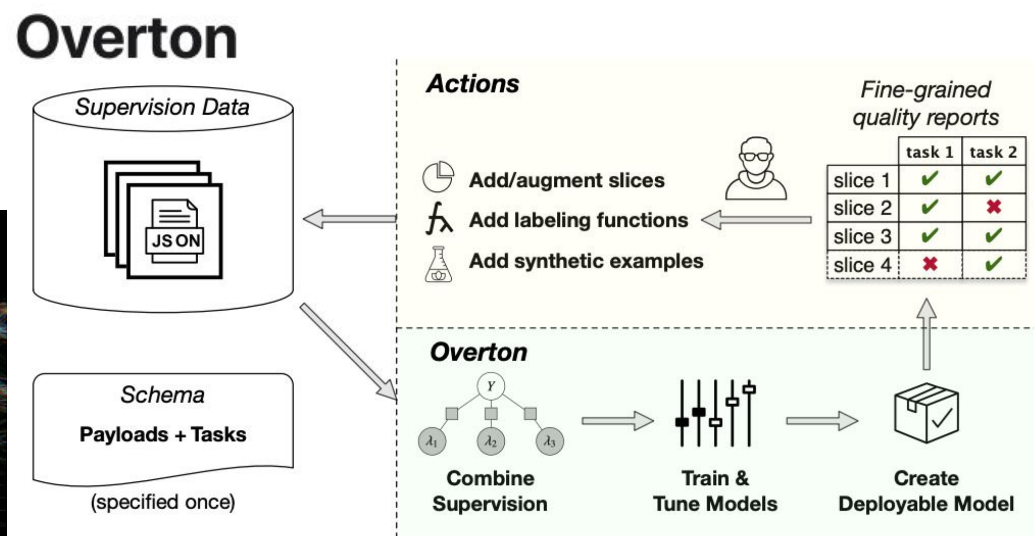
Modern ML Pipelines

ML pipelines help engineers build and deploy models

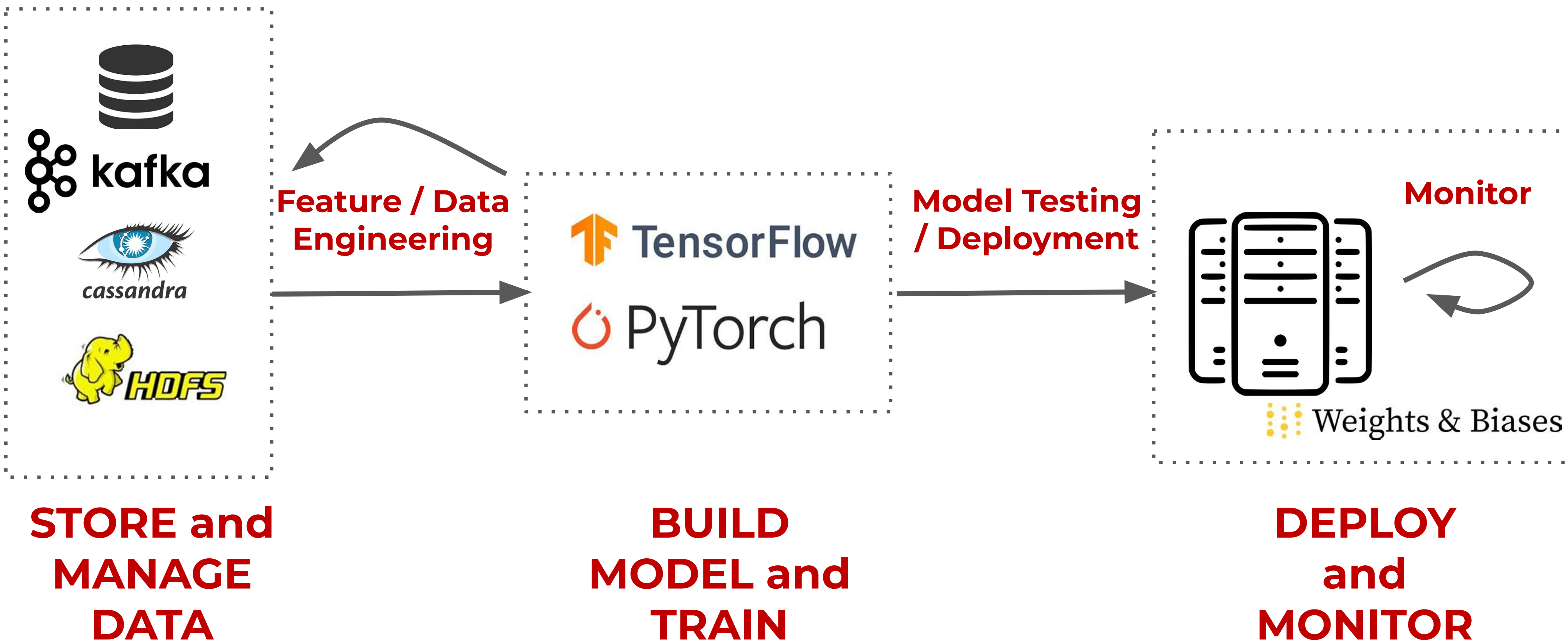
Standardization **Reproducibility** **Easier to Maintain**



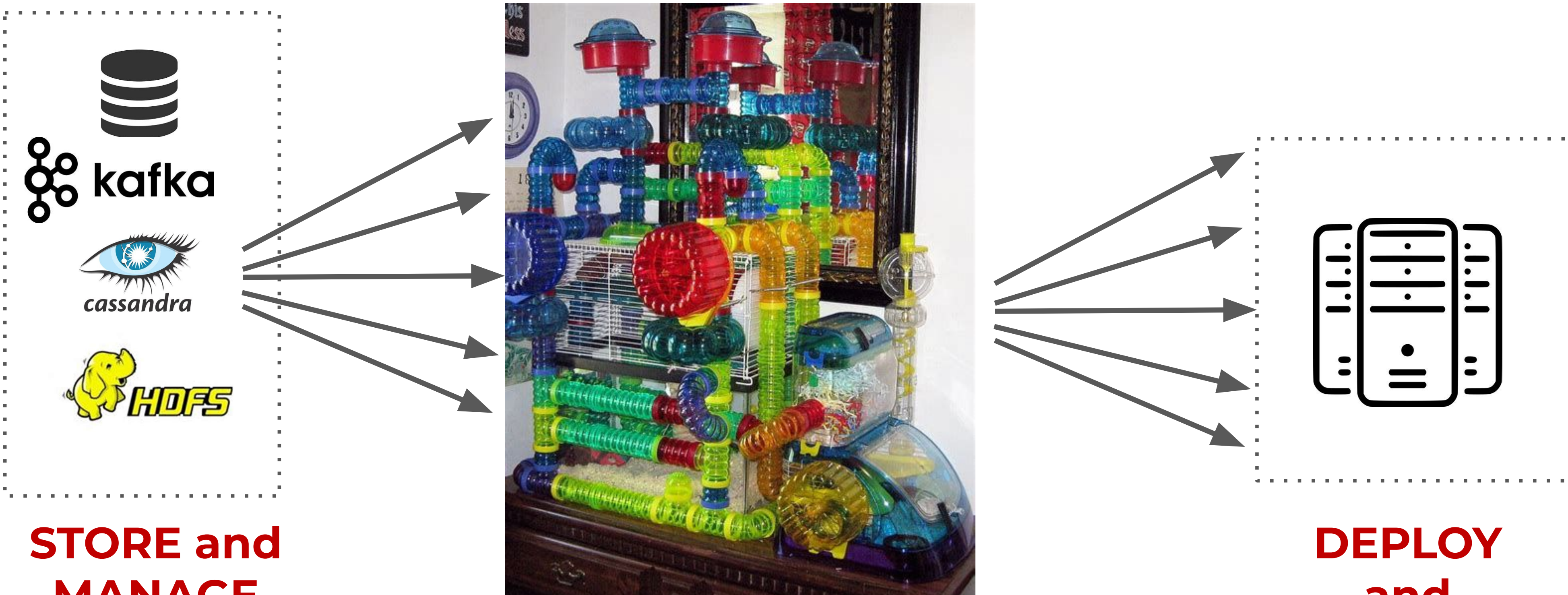




Engineer Workflow Today



Engineer Workflow *of Yesteryear (< 2017-8)*



STORE and MANAGE DATA

"Pipeline Jungle*"

DEPLOY and MONITOR

*Sculley, David, et al. "Hidden technical debt in machine learning systems." *Neurips* (2015)

The “Pipeline Jungle” Experience

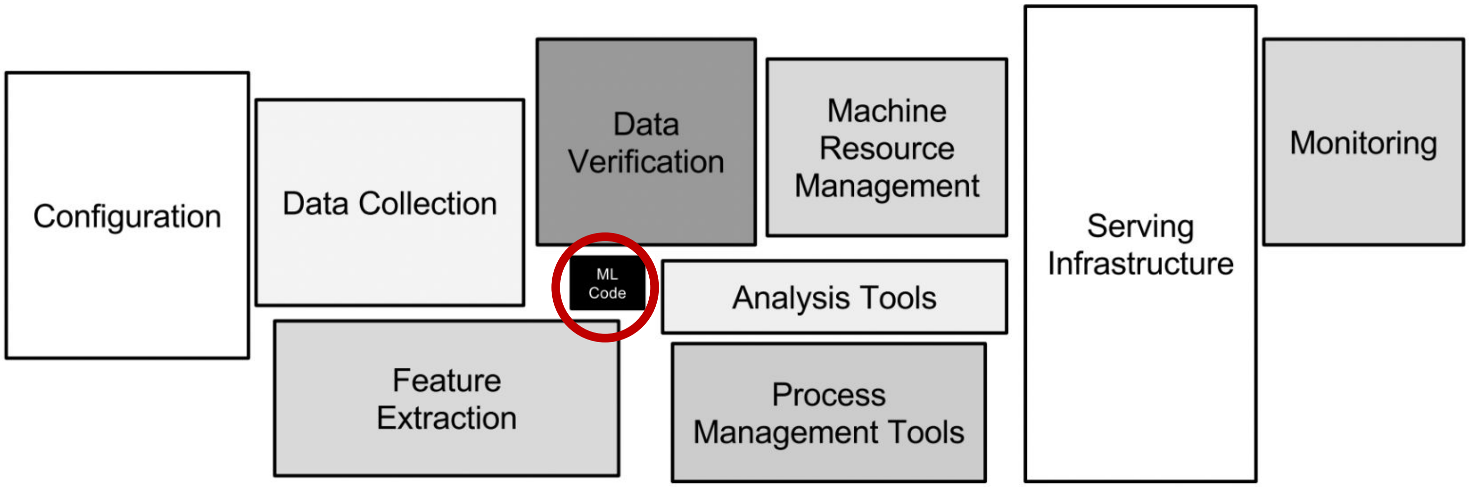


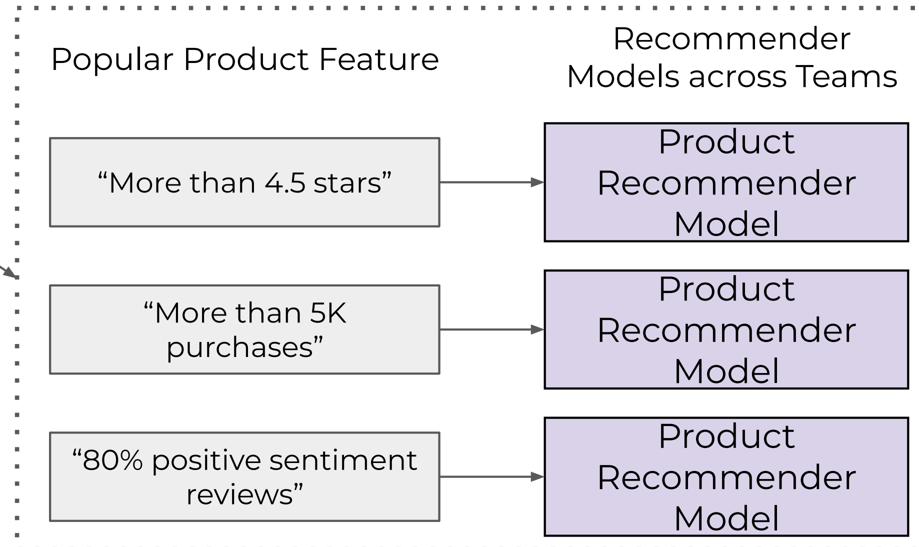
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

*Sculley, David, et al. "Hidden technical debt in machine learning systems." *Neurips* (2015)

The “Pipeline Jungle” Experience

The challenges to deploying a model:

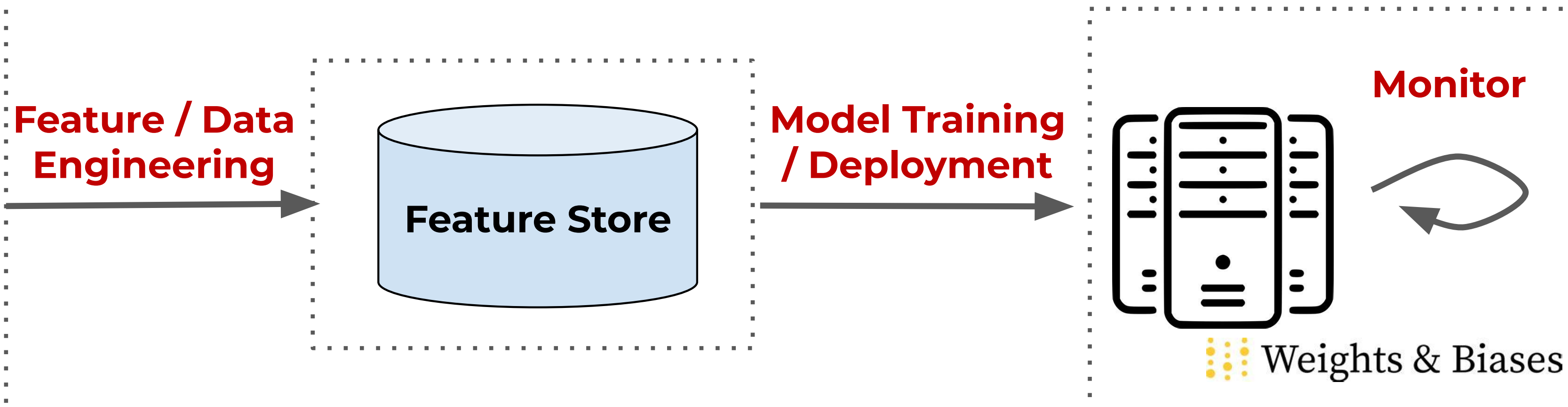
- One-off feature definitions
- Lack of reproducibility
- Inconsistent storage
- No standard evaluations and testing
- Difficult to detect and recover from errors
- ...



Feature Store Solution

PART 1

Systems to build, deploy, and monitor ML pipelines with special focus on *feature management*.



STORE and MANAGE DATA

FEATURE MANAGEMENT

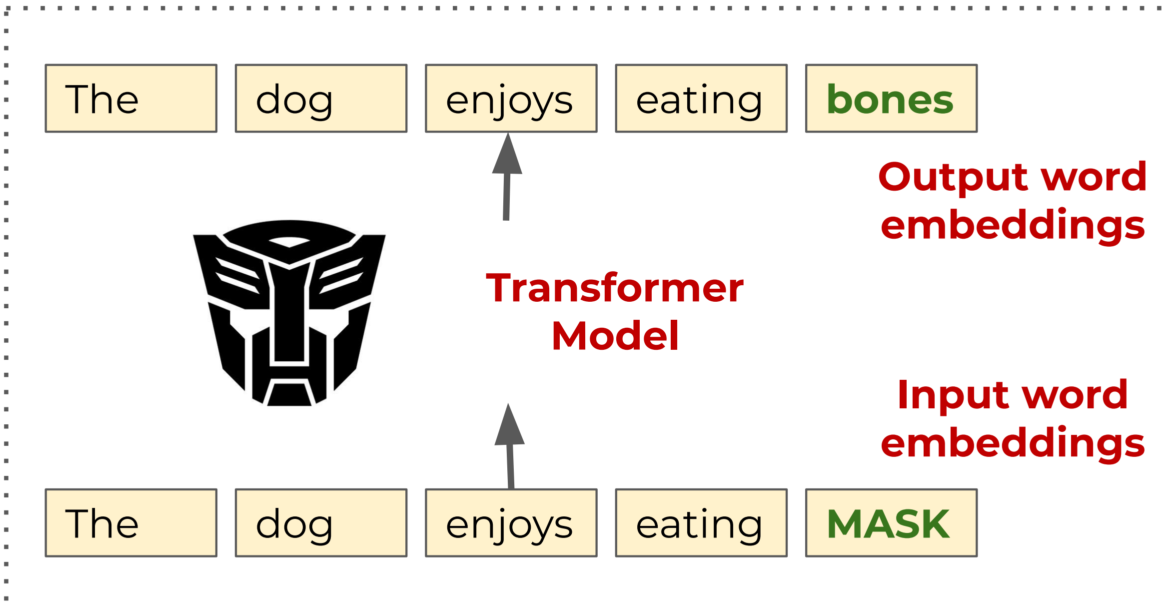
DEPLOY and MONITOR

Enter Self-Supervision

Paradigm where models learn embedding representations of underlying training data *without* needed manually labels.

Self-Supervision Example: Transformers and MLM

Learn word embeddings by train a language model to predict a masked word in a given context.

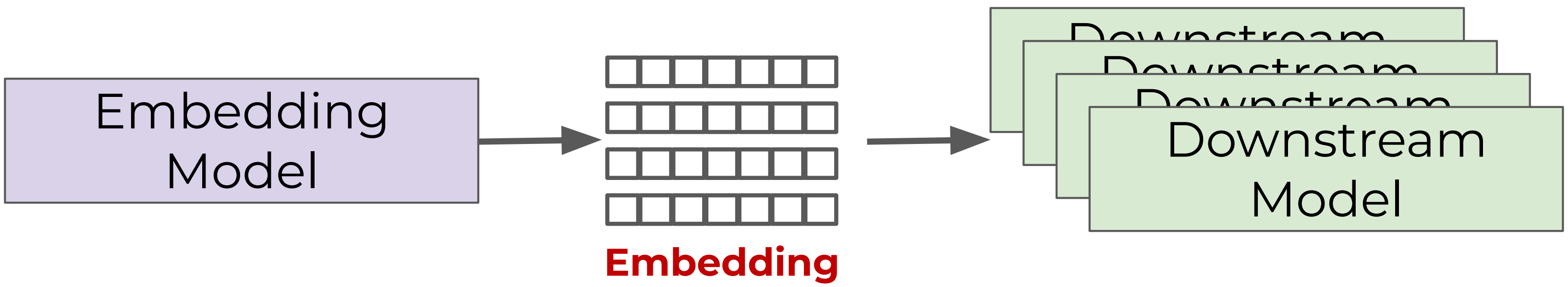


Word embeddings encode contextual information.

Enter Self-Supervision

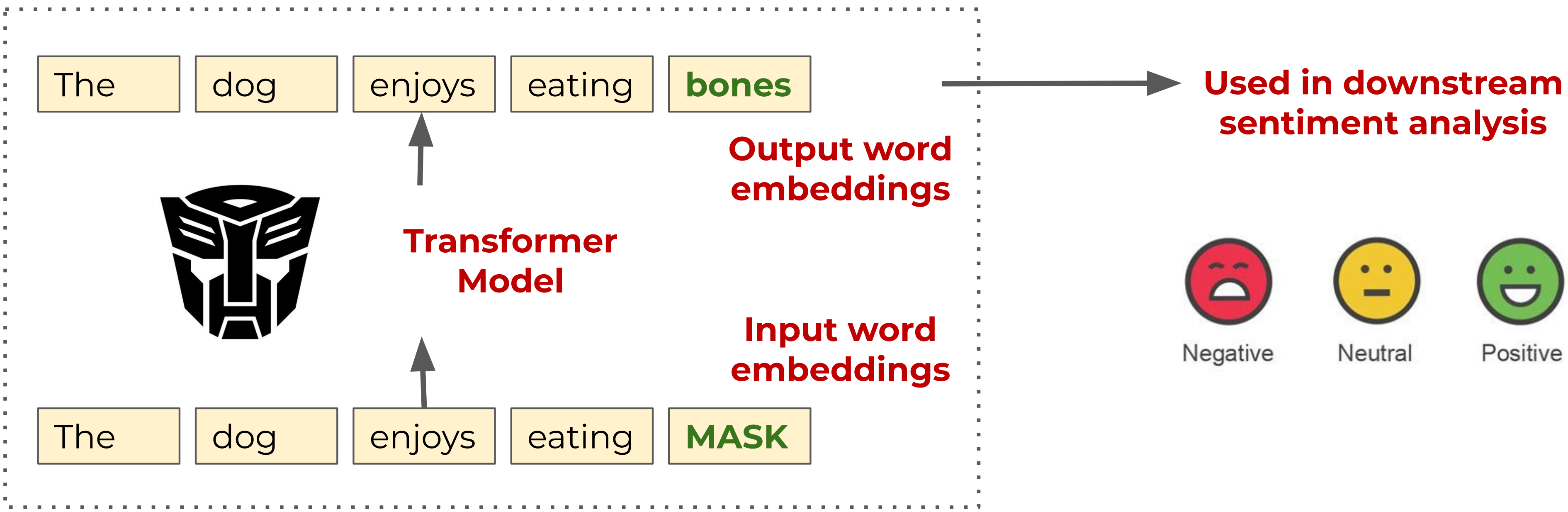
Paradigm where models learn embedding representations of the underlying training data *without* manual labels.

Embeddings are then used in downstream models.



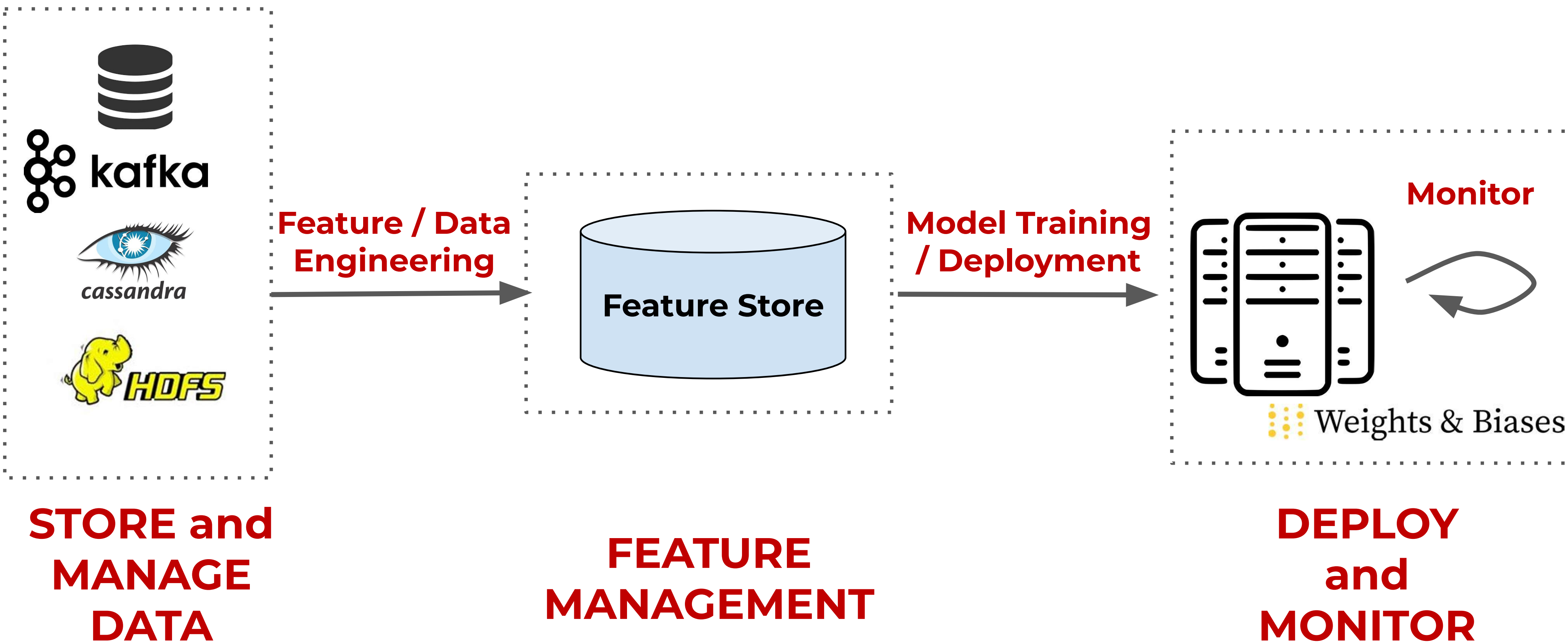
Self-Supervision Example: Transformers

Learn word embeddings by train a language model to predict a masked word in a given context.



Word embeddings encode contextual information.

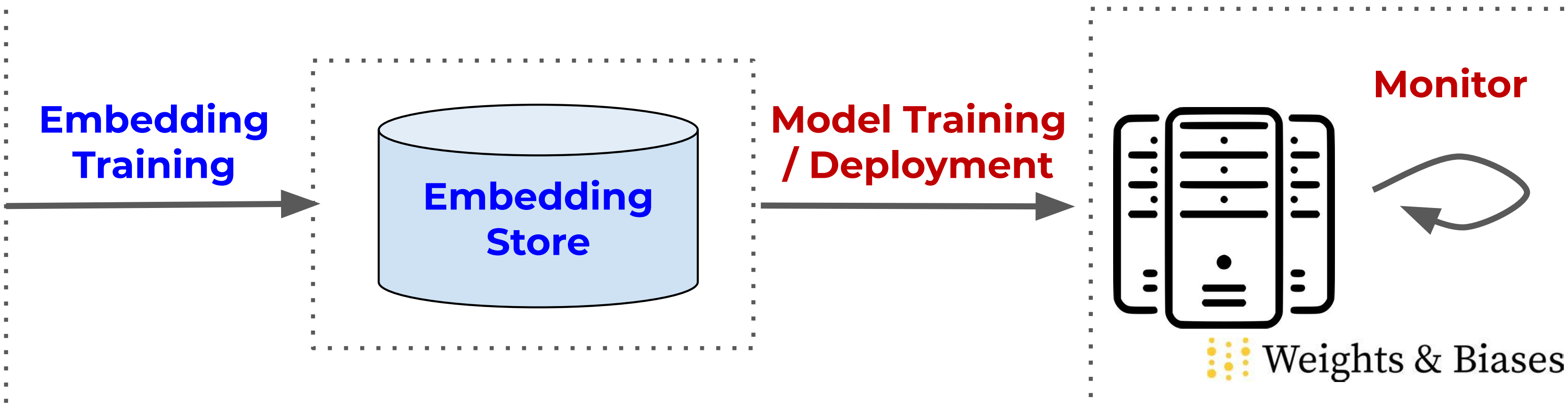
Recall Feature Store Solution



Embedding Ecosystems

PART 2

Self-supervised embeddings, models that train them, and downstream systems that use them.



STORE and MANAGE DATA

EMBEDDING MANAGEMENT

DEPLOY and MONITOR

Reduction in engineer effort

Higher quality downstream systems

One embedding for multiple tasks

Feature Stores

Uber

Engineer Workflow *of Yesteryear (< 2017-8)*

kafka

cassandra

HDFS

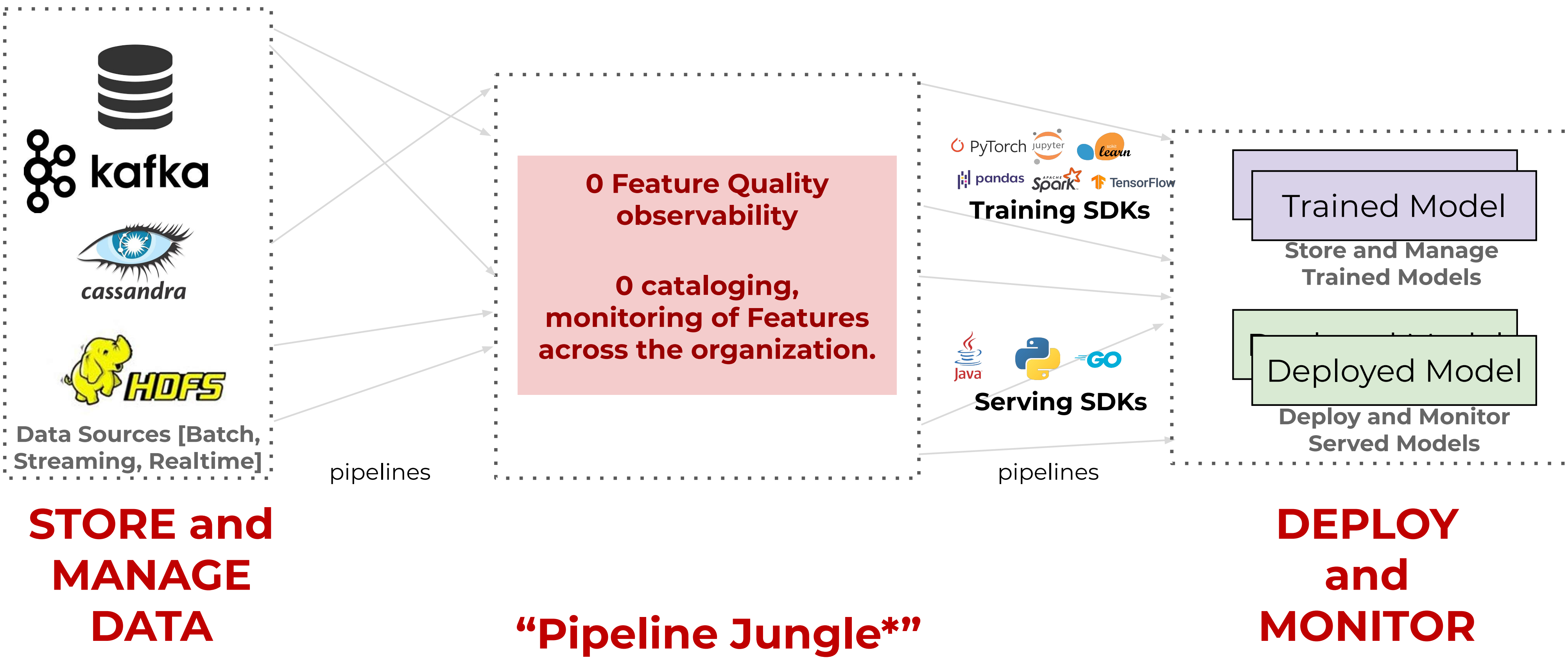
STORE and MANAGE DATA

"Pipeline Jungle*"

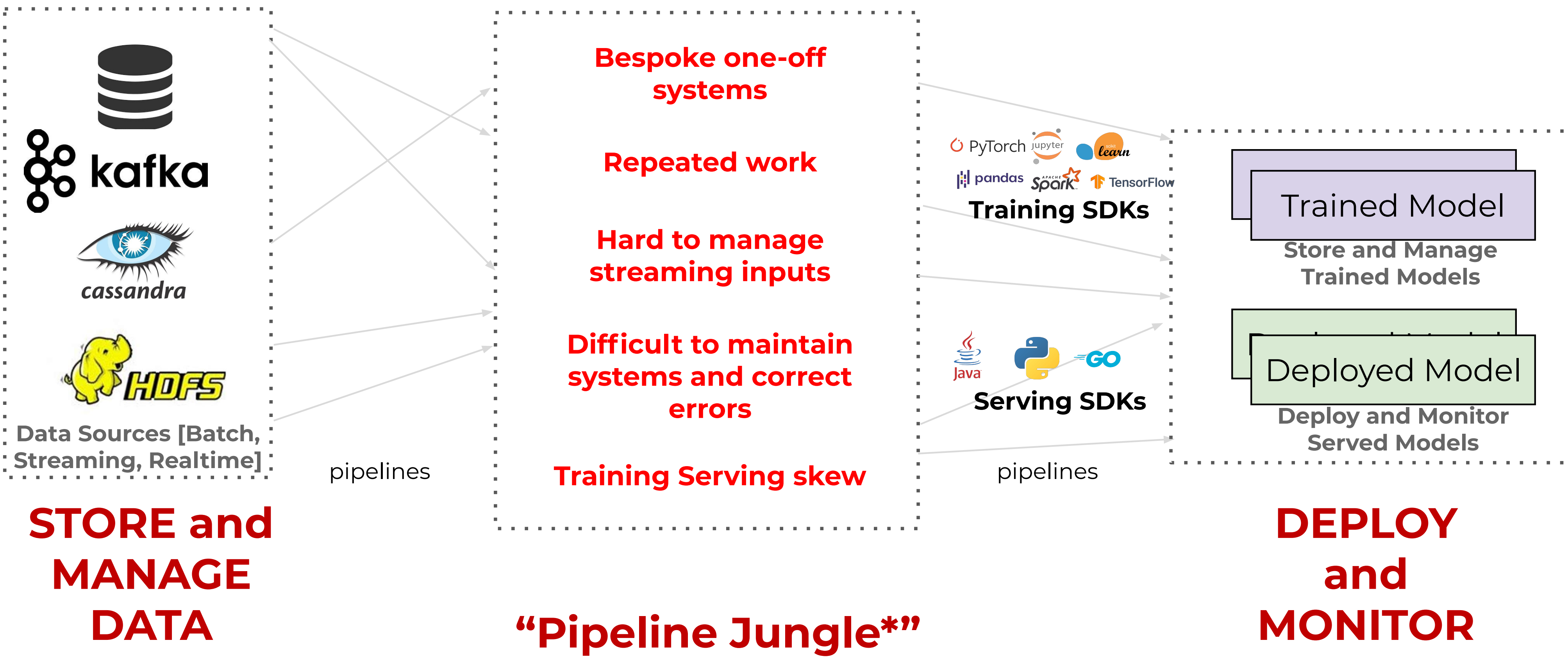
DEPLOY and MONITOR

*Sculley, David, et al. "Hidden technical debt in machine learning systems." *Neurips* (2015)

Lack of Feature Management



Lack of Feature Management



Lack of Feature Management

Days/Weeks to make data ML ready

- Materializing Features from **various data sources**.
- **Duplicating code while materializing** in training & serving
- **No guarantees** of training-serving parity

Near 0 monitoring of Features

- **No Feature health metrics** out of the box (due to the various sources problem)
- **No online-offline parity monitoring,** leading to models performing poorly
- **No feature drift** monitoring
- **No idea about Feature impact** on a model

High latency, unreliable Feature serving in production models at scale

- **Poor Model latencies** leading to bad user experience.
- **No dedicated dynamic resource allocation** for feature engineering
- **Multiple RPC calls at high throughputs to fetch features** dramatically increasing latencies

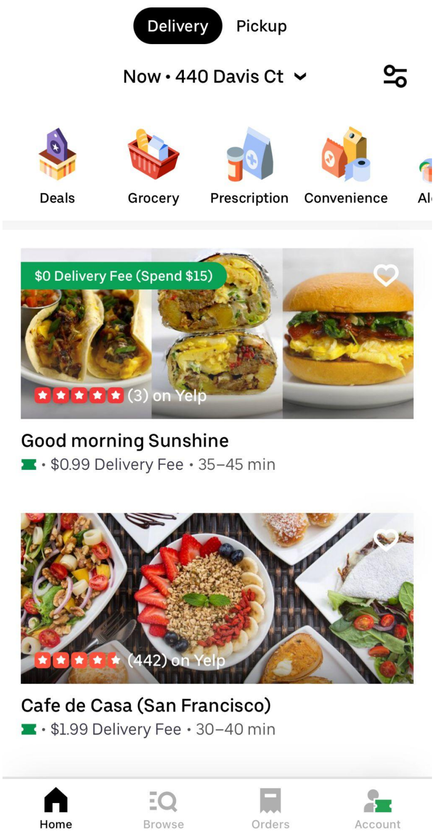
Feature Stores

Uber

Use case - ETA of an Uber EATS Order

🔑 Key ML Features

- How large is the order? (*order_size*)
- How busy is the restaurant? (*n_meal*)
- How quick is the restaurant? (*meal_preptime*)
- How busy is the traffic? (*n_busy*)



Palette Feature Store: Workflow

Lookup Existing Features

- Search for features
    - by feature_name
    - by entity (e.g. eater_features)
    - by type (e.g. categorical_features)
    - by models (e.g. features used in eta_prediction_model)

    or any combination ...

Palette Feature Store: Workflow

Lookup Existing Features → Onboard Features → Author Feature Pipelines

- Onboard Features and Author Pipelines
  - Metadata driven onboarding process
  - Feature Pipelines automatically created
  - Immediately available for consumption during Training & Inferencing

Palette Feature Store: Workflow

Lookup Existing Features → Onboard Features → Author Feature Pipelines → Train Your Model → Deploy Your Model

```
dsl1 = DSLEstimator(lambdas={
  "region_id": regionId("@palette:restaurant:realtime_feature:lat:r_id", "@palette:restaurant:realtime_feature:lat:r_id")
}

dsl2 = DSLEstimator(lambdas={
       "prep_time": nFill(nVal("@palette:restaurant:batch_features:prep_time:r_id"),
       "n_meal": nFill(nVal("@palette:restaurant:realtime_features:n_nean:r_id"),
       "n_order": nFill(nVal("@basis:n_order"),
       "n_busy": nFill(nVal("@palette:restaurant:service_feature:n_busy:region_id"))
})

ml_pipeline = MLPipeline(dsl1, dsl2)
model = ml_pipeline.fit(basis_dataframe)
michelangelo_api.deploy(model)
```

Palette Feature Store: Workflow

Lookup Existing Features → Onboard Features → Author Feature Pipelines → Train Your Model → Serve Your Model → Monitor Feature Metrics

- Monitor Feature Metrics
  - Training-Serving **Skew**
  - Feature **Drift**
  - Feature **Importance<>Drift** correlation
  - Feature Quality (**Freshness, Consistency, Null Rate**)

Feature Store (Palette) Lifecycle

Feature Preparation
Batch &
Streaming ETLs

Feature Storage

Historical & Near Real-Time

Curated & Crowd-sourced

Metadata

Scalable offline access

Scalable online access

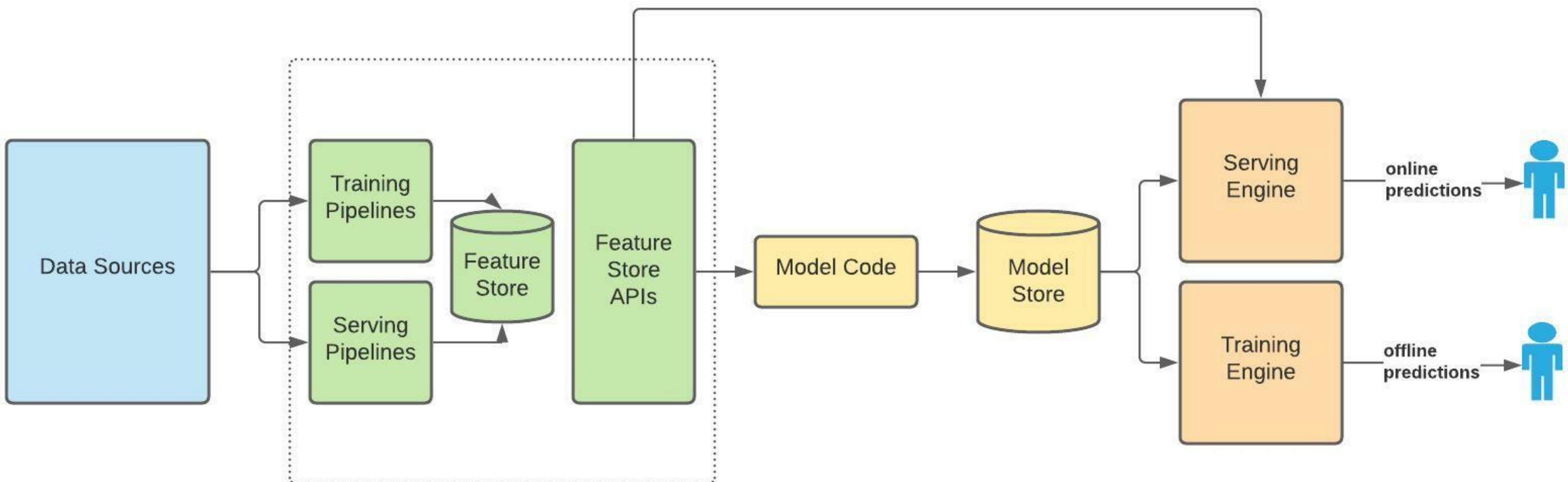
Online/Offline data parity

Feature Discovery
Sharing across Models
Automatic feature selection

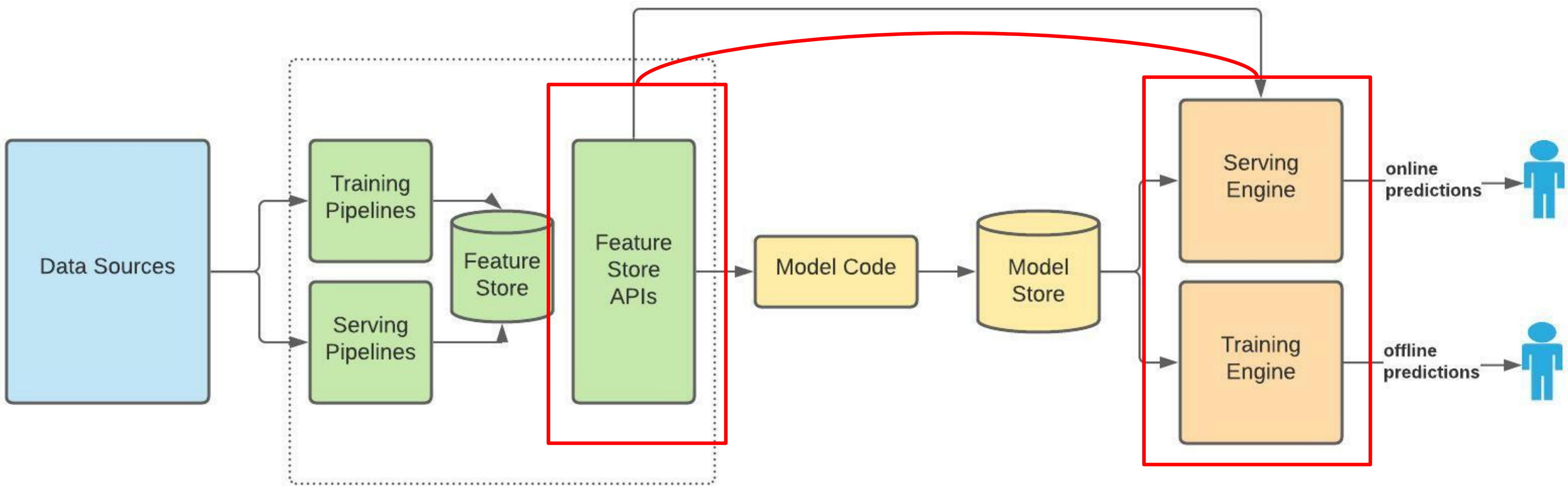
Feature Monitoring
Data Quality reporting

Feature Transforms
Model specific transforms

Feature Stores in an End to End ML Platform



Feature Stores in an End to End ML Platform



Palette Feature Store Organization

Organized as **entity : feature_group : feature : join_key**

e.g. restaurant : order_history : meal_preptime : restaurant_uuid

- Feature Store Abstractions:
  - **Entity**: A Top Level Business Unit (e.g. **eater, courier, restaurant**)
  - **Feature Group**: Group of Features commonly used together (e.g. **order_history**)
  - **Feature**: The Feature (e.g. **meal_preptime, n_meal, sum_orders_1week**)
  - **Entity Key**: The UUIDs of the entities (e.g. **eater_uuid, restaurant_uuid**)
- **Bring** your join keys or UUIDs
- **Join** together cross-entity Feature sets with minimal code
- **Train** on historical Feature values
- **Serve** the latest, most accurate values of Features at Low Latency
- Backed by a dual datastore system (training & serving)
- Get Training Serving parity out of the box

Feature Types in Palette (Michelangelo)

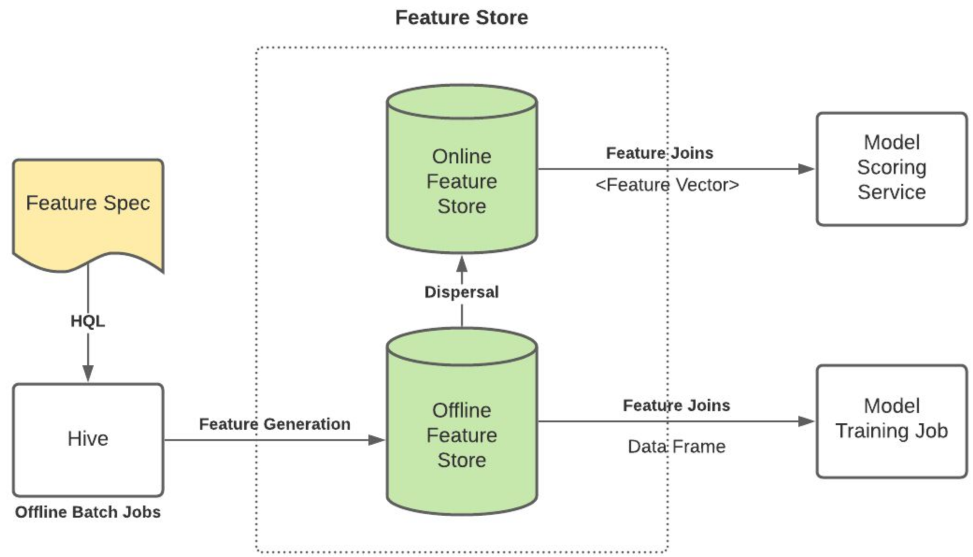
- **Batch Features**:
  Features calibrated on **historical data**
  Generated via offline **batch jobs**
  Auto dispersed for model inferencing
  E.g. meal_preptime (average prep time of historical orders)

- **Near Real Time Features**:
  Features calibrated on **streaming data**
  Generated via **near real-time streaming jobs** (Flink, AthenaX)
  Auto dispersed for model training
  E.g. n_meal (how busy is the restaurant)

- **RPC Features**: Features retrieved via 3P APIs
  Features calibrated on 3P API calls
  Calculated at run time and served to models directly
  Auto dispersed for model training
  E.g. location_geohash (current geohash location of the courier)

Computing Batch Features

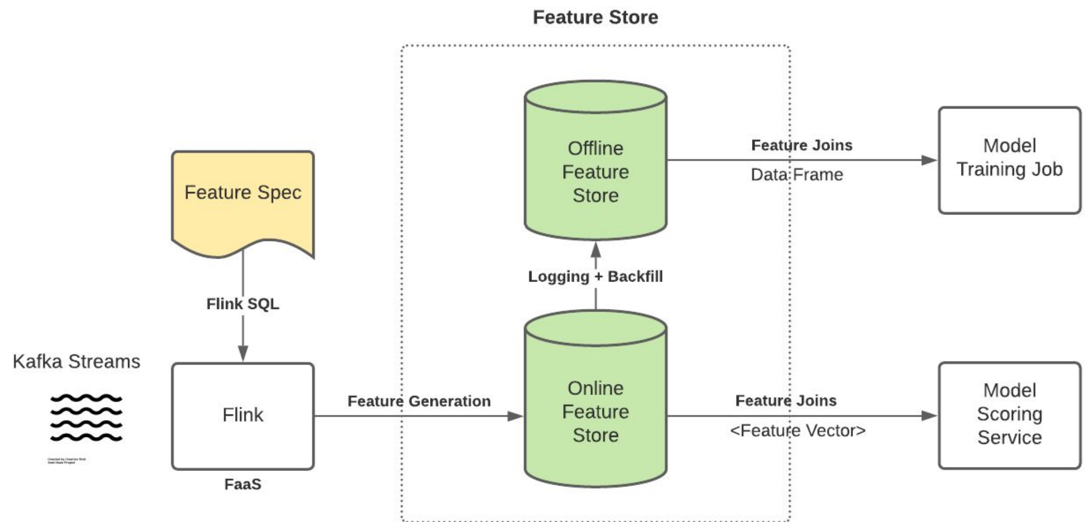
- Computed using **Historical Data**
- Not time sensitive
- Ingested from Hive Queries or Spark Jobs
- Aggregates over days/weeks
- E.g. *meal_preptime*



Computing Near Real Time Features

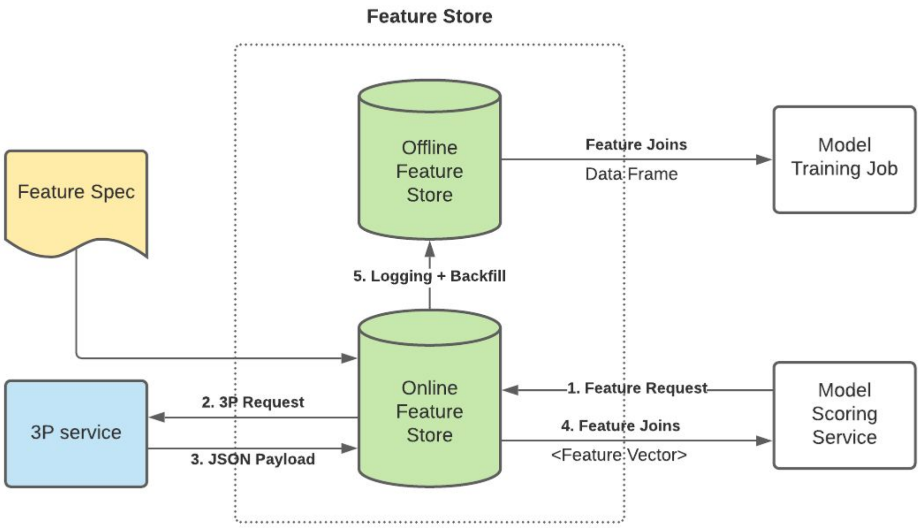
Uber

- Signals generated **seconds ago**
- Write Flink SQL to perform real time aggregations
- Materialize to the online store
- Auto ETL and Backfill to the offline store
- E.g. n_busy (How busy is the restaurant)
    - **Kafka** event streams
    - Perform **Real-Time aggregations**



Computing RPC Features

- Signals generated **in real-time**
- Make RPC calls to Fetch Features behind the scenes
- Auto ETL and Backfill to the offline store
- E.g. *lat/long*:
  - Fetched via HTTP calls



Feature Extraction & Transformation

- Michelangelo Transformer
  - transform() and scoreInstance()
  - ML Readable / Writable
  - Extension of the Spark Transformer Framework
  - Parity across Spark and Spark-less environments
    - UDFs / DSLs
    - In-house unit testing framework for parity

- Feature Store APIs as Transformers
  - Feature Engineering as an integral part of the ML Pipeline

Uber

Michelangelo Feature Store APIs as Spark Transformers

```
palette_tx1 = PaletteTransformer( {
   "nMeal": "@palette:restaurant:realtime_feature:nMeal:r_id",
   "prep_time": "@palette:restaurant:batch_feature:prep_time:r_id",
   "lat": "@palette:restaurant:realtime_feature:lat:r_id",
   "long": "@palette:restaurant:realtime_feature:long:r_id",
})
```

- Instantiate Palette Transformer with Feature expressions
- Create a pipeline with one or more stages of estimators and transformers
- model = **pipeline.fit()**
- Evaluate your model via **transform()**
- Score your model via **scoreInstance()**

DSLs: Feature Manipulation / Imputation

- Write expressions to define Transformations
- Pre-compiled Scala code execution at runtime
- Example Michelangelo code:

```
dsl_est1 = DSLEstimator(lambdas={
  "region_id": regionId("restaurant:fg:lat:r_id", "restaurant:fg:long:r_id")
}

dsl_est2 = DSLEstimator(lambdas={
  "prep_time":  nFill(nVal("restaurant:batch_fg:prep_time:r_id"), -1),
  "n_meal":     nFill(nVal("restaurant:realtime_fg:n_meal:r_id"), -1),
  "order_size": nVal("basis:order_size"),
  "n_busy":     nFill(nVal("restaurant:service_fg:n_busy:region_id"), -1)
})
```

***fg: feature_group**

Uber EATS Transformation Example

Computation Order

- n_meal:restaurant_id -> n_meal
- meal_preptime:restaurant_id -> meal_preptime -> DSL
- busy_scale: restaurant_id -> lat, long -> regionId(lat, long) -> busy_scale

Palette Transformer
Id -> n_meal
Id -> meal_preptime
Id -> lat, long

DSL Transformer
Lat, long -> region_id

Palette Transformer
region_id -> n_busy

DSL Transformer
impute(n_meal)
impute(meal_preptime)

Training: transform()
Serving: score_instance()

Feature Store Results & Takeaways

- **Democratized Usage**: 20K+ Features used across 8K+ production models
- **Model development times** reduced from days to hours
- **Multi Modality Support**: Batch, Realtime and RPC Features with online and offline parity
- **Offline scalability**: Joins across billions of rows
- **Online serving latency**: Parallel IO, fast storage with caching
- **Feature Transformers**: Setup chains of transformations at training/serving time

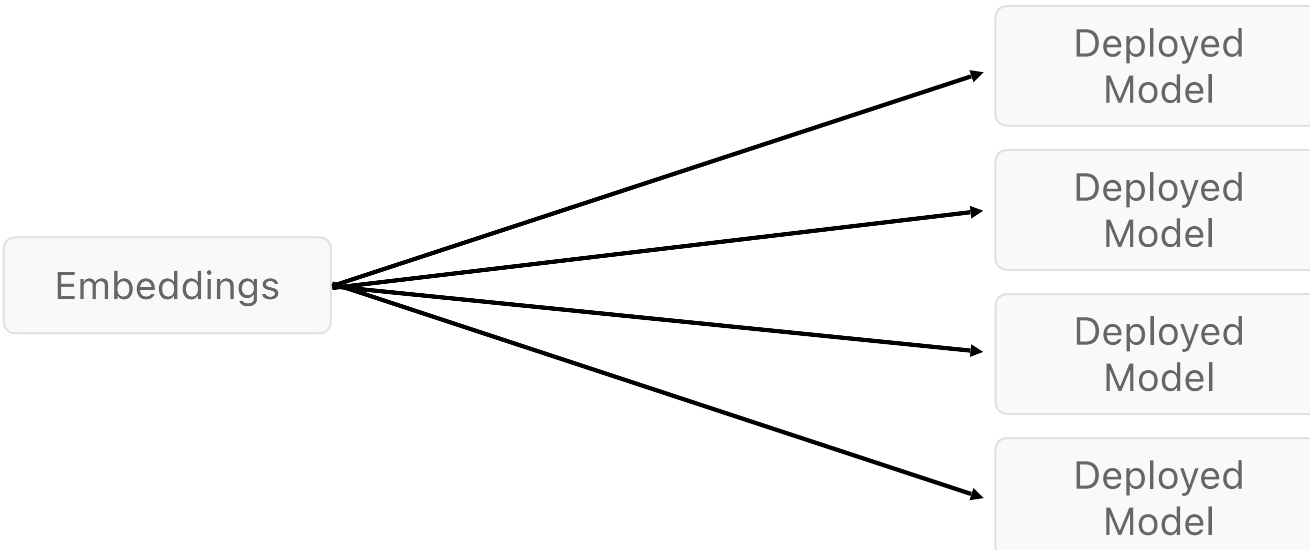
Embedding Ecosystems

Managing ML Pipelines:
Feature Stores and the Coming Wave of Embedding Ecosystems

Xiao Ling | VLDB Tutorial 2021

Recap: Self-Supervision Embeddings

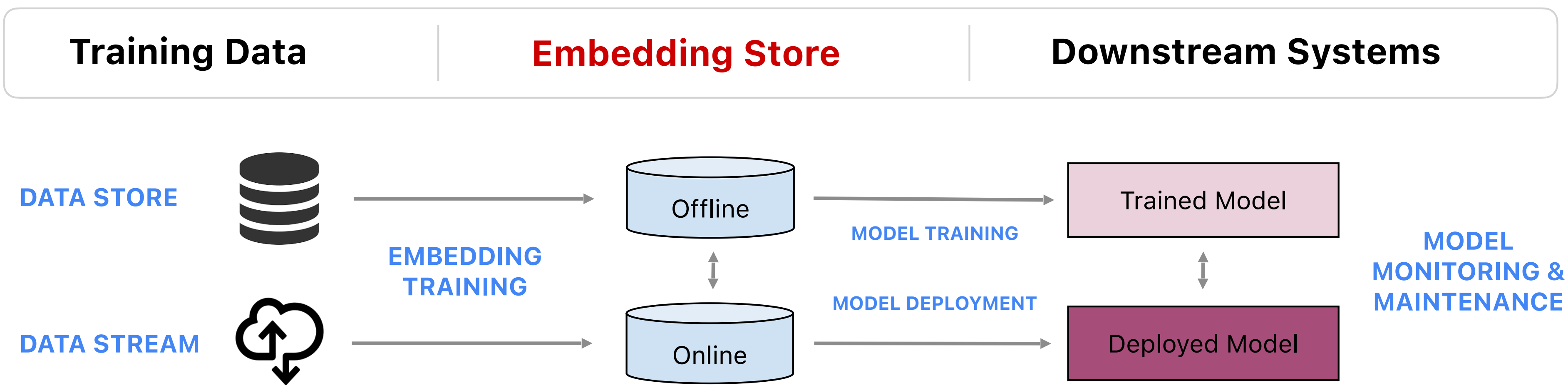
Used in many different downstream systems



Downstream systems require less supervised data and provide a quality lift compared to hand-tuned predecessors.

Recap: Embedding Ecosystems

New age of feature store systems manage pretrained embeddings downstream systems use them as inputs.



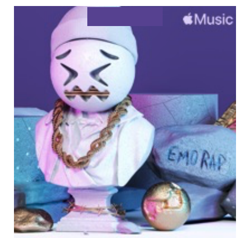
Grounding Use Case: Named Entity Disambiguation

Map "strings to things" in a knowledge base.

Key part of assistant, search, and information extraction

How tall is Lincoln?

The Long Tail of Entities

HEAD	TORSO	TAIL	UNSEEN
			
Washington, DC	Chevrolet, Corvette	Reddish Potato Beetle	Sauce! by XXXTENTACION
Q61	Q56166	Q14934552	Q???
>1k examples in training	(10, 1000]	[1, 10]	0 example

Popular → **Rare**

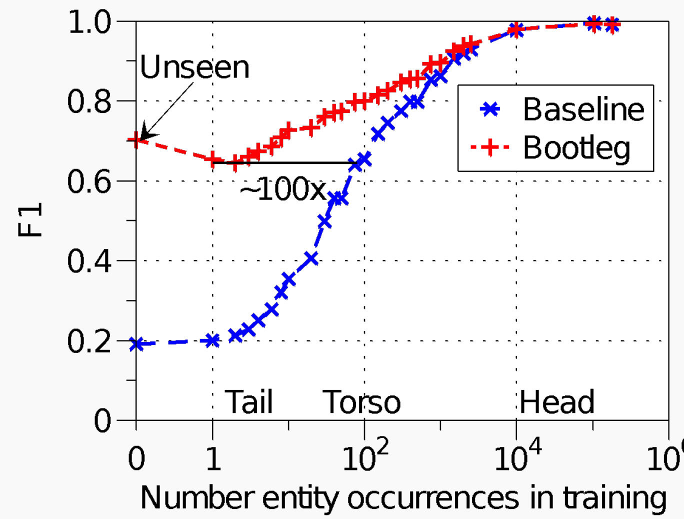
IR and BERT/IR work great for the HEAD.

The majority of entities are rare!
13% entities have Wikipedia page.
< 1% of songs in Wikidata!

#1 Tail Scalability Challenge

Large number of patterns needed to resolve the tail, making it difficult to scale a system that can learn the patterns.

90 million entities in Wikidata -> 90*100 million examples for 60 F1



Subtle reasoning clues are needed for the tail!
(+40 F1 points by encoding these reasoning patterns)

Bootleg: Chasing the Tail with Self-Supervised Named Entity Disambiguation. Orr et al, CIDR 2021

Entity Embeddings in Downstream Applications

Experiment on the entity linking task in an existing Q&A system

- With and without Bootleg-learned entity embeddings

- The entity embeddings significantly improve F1 by a relative 8%

  Also, a relative 8% improvement on tail entities!

Bootleg: Chasing the Tail with Self-Supervised Named Entity Disambiguation. Orr et al, CIDR 2021

#2 Memory Usage

Embeddings linearly grow per number of entities

- 128d float32 x 5M ~= 2.4 Gb (English Wikipedia)
  - 128d float32 x 96M ~= 46.08 Gb (Wikidata)

It requires larger and larger servers over time 💸💸💸

- More computation affects service latency 🐌🐌🐌

Hard to fit *on device!* 📱

Memory Usage

Memory can be saved w/o a big quality sacrifice

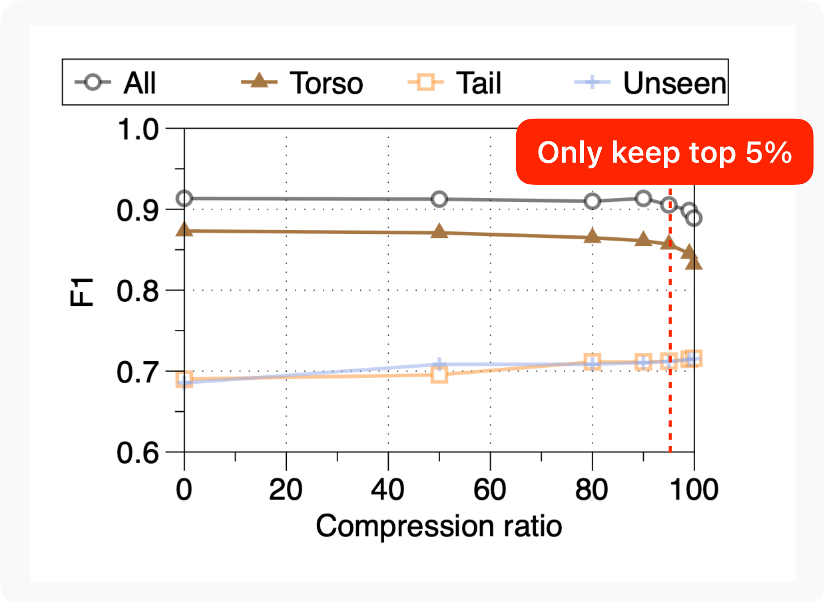
Only keep the top k entity embeddings (i.e., compression ratio 100 - k)

- Uses a random UNK entity embedding

- Less memory-heavy signals remain

F1 only drops by 0.8 overall

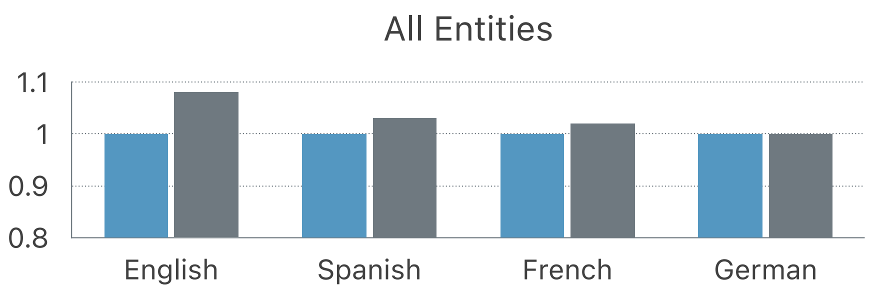
- Memory drops from 5.2GB to 0.3GB!

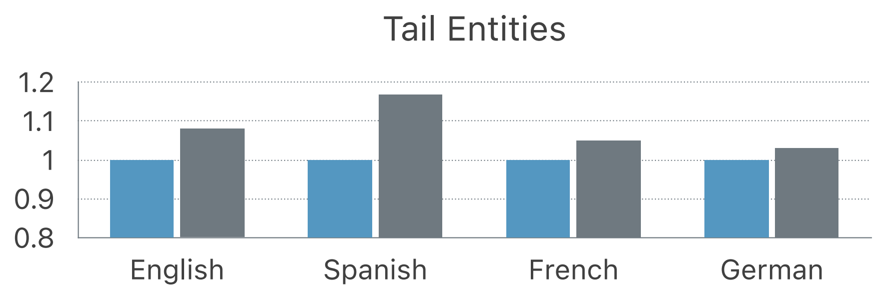


Bootleg: Chasing the Tail with Self-Supervised Named Entity Disambiguation. Orr et al, CIDR 2021

#3 Embeddings in i18n languages

Embeddings work on other languages





Challenges

- Lack of equally abundant resources in English

- Memory usage increases the size of embeddings by the num of languages

Multilingual Entity Embeddings

Botha et al. 2020 proposed to train multilingual entity embeddings

- Memory usage doesn't grow with the number of languages
- Entity embeddings trained from resources across languages
  - Enabled by a multilingual language model

Entity Linking in 100 Languages. Botha et al., EMNLP 2020

#4 Embedding Stability

Entity embeddings are self-supervised from Wikipedia

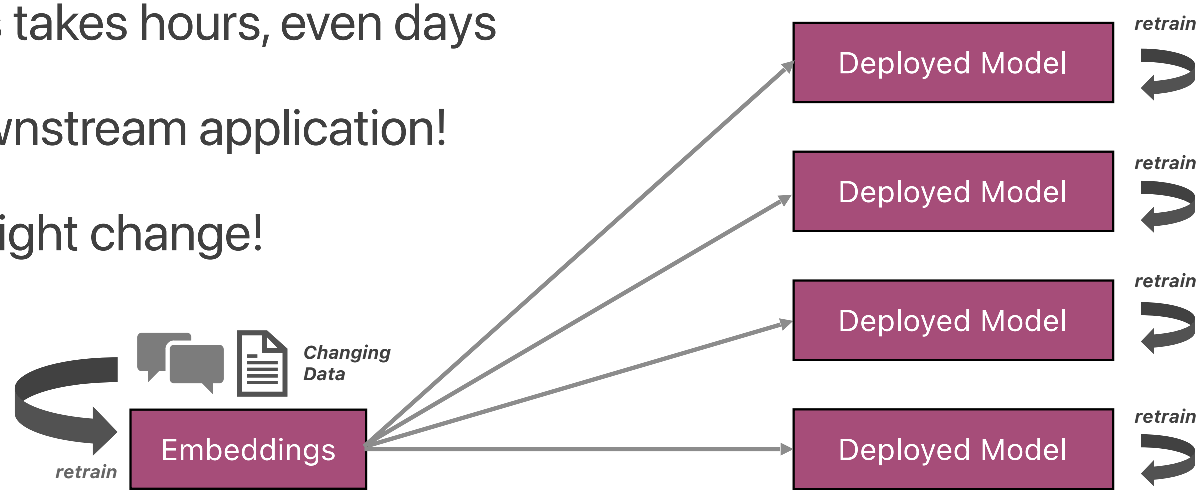
- 20k new articles / month

Updating the model is hard

- Retraining entity embeddings takes hours, even days

  Also, need to retrain *each* downstream application!

- Previous correct prediction might change!



#5 Model Evaluation and Monitoring

Are the embeddings

- sensitive to questions?

- vulnerable to attacks?

- biased to entities popular in one country?, Etc…

Is the downstream application affected by updated embeddings?

- What about 10s or 100s downstream apps?

- How to enable safe regular model updates?

Summary of Challenges

#1 Long-tail of entities

#2 Memory usage

#3 Multi-lingual embeddings

#4 Embedding stability

#5 Model monitoring

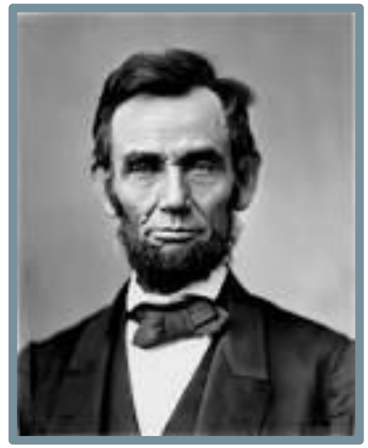
* **Bold** will be discussed in the following sections

Self-Supervised Training Data: The Challenge of the Long Tail

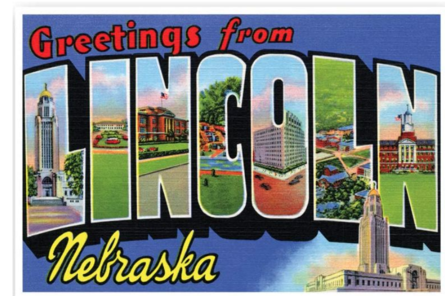
Grounding Use Case: Named Entity Disambiguation

Map “strings to things” in a knowledge base.

How tall is *Lincoln*?



Q91

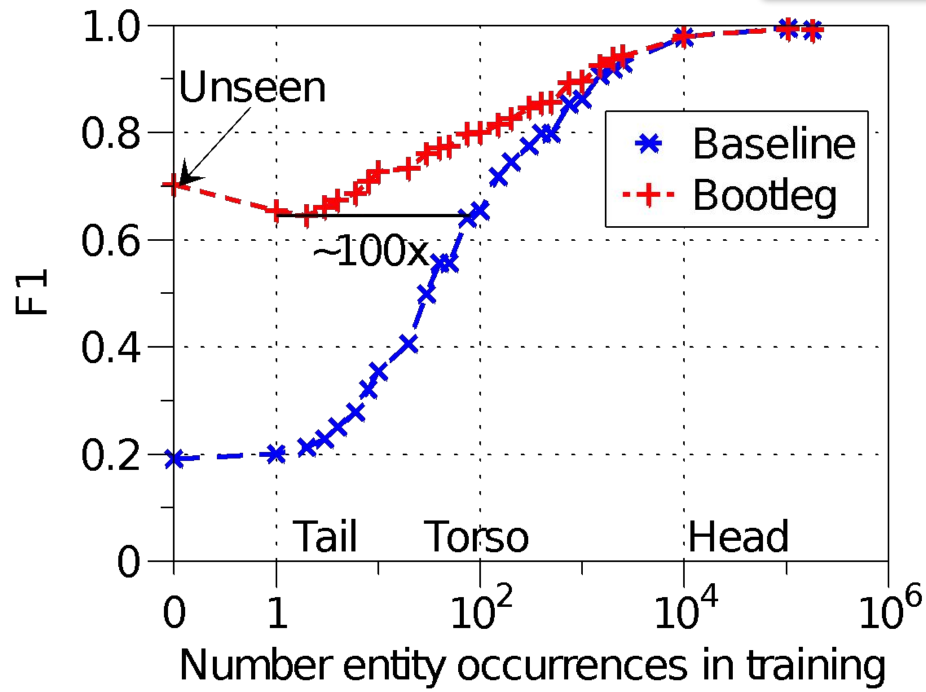


LINCOLN

Key part of assistant, search, and information extraction

Tail Challenge

Impossible to scale the data to memorize all patterns needed for rare entities



Subtle reasoning clues are needed for the tail!
(+40 F1 points by encoding these reasoning patterns)

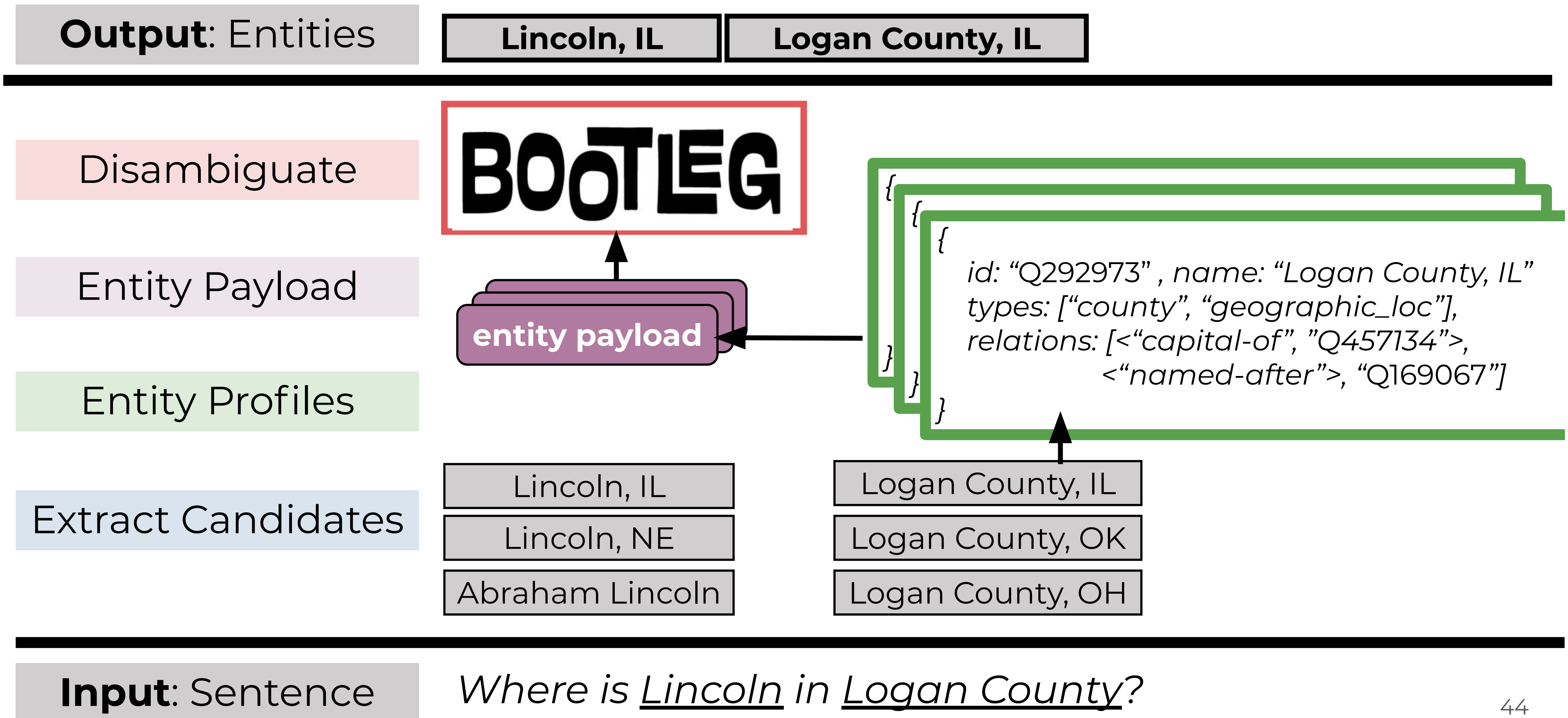
Bootleg: Tackles the Tail with Structural Knowledge

BOOTLEG

Key Idea: reasoning over *type* and *relationship* signals can resolve unseen entities.

Orr, Laurel, et al. "Bootleg: Chasing the tail with self-supervised named entity disambiguation." *arXiv preprint arXiv:2010.10363* (2020).

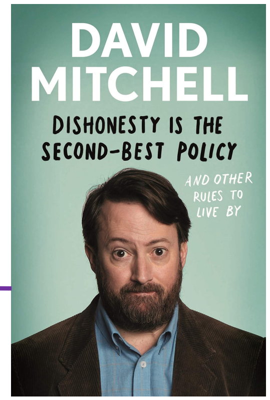
Disambiguation Inputs and Outputs



Reasoning over Relationships

Victoria Mitchell
(poker player, writer)

spouses



David Mitchell

Victoria
Mitchell
(runner)

David and
Victoria Mitchell
added spice to
their marriage

Love Story by Taylor Swift

Love Story by Andy Williams

Play Love Story
by Williams

Reasoning over Types

How tall is Lincoln?

People have heights, not places or brands

What is the cheapest Lincoln?

LINCOLN

Brands have prices, not places or people

How many people are in Lincoln?

Places have populations, not people or brands

Bootleg: Tackles the Tail with Structural Knowledge

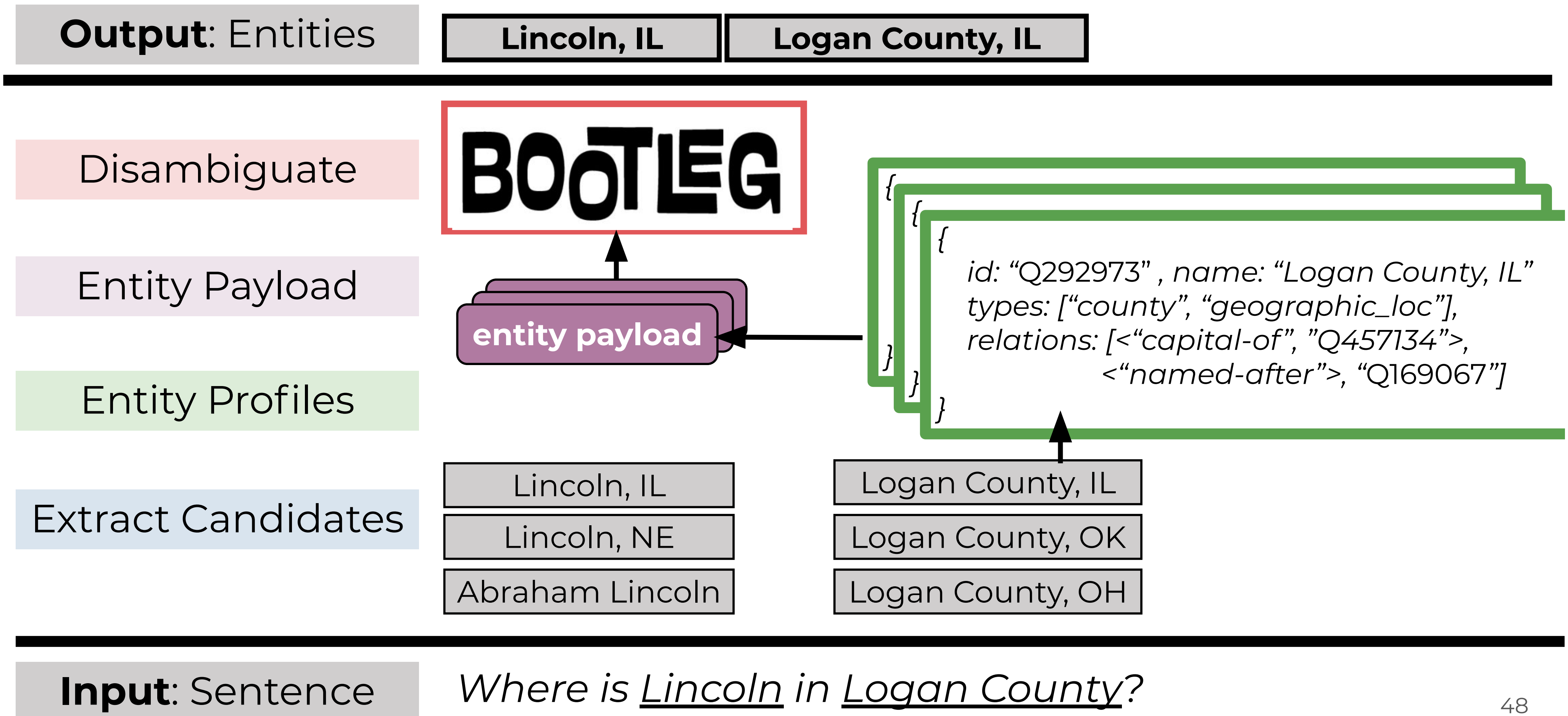
BOOTLEG

Key Idea: reasoning over *type* and *relationship* signals can resolve unseen entities.

Implementation: use *embeddings* to teach a model to reason over types and relationships.

Orr, Laurel, et al. "Bootleg: Chasing the tail with self-supervised named entity disambiguation." *arXiv preprint arXiv:2010.10363* (2020).

Disambiguation Inputs and Outputs



Using Embeddings to Encode Signals

Entity Embedding

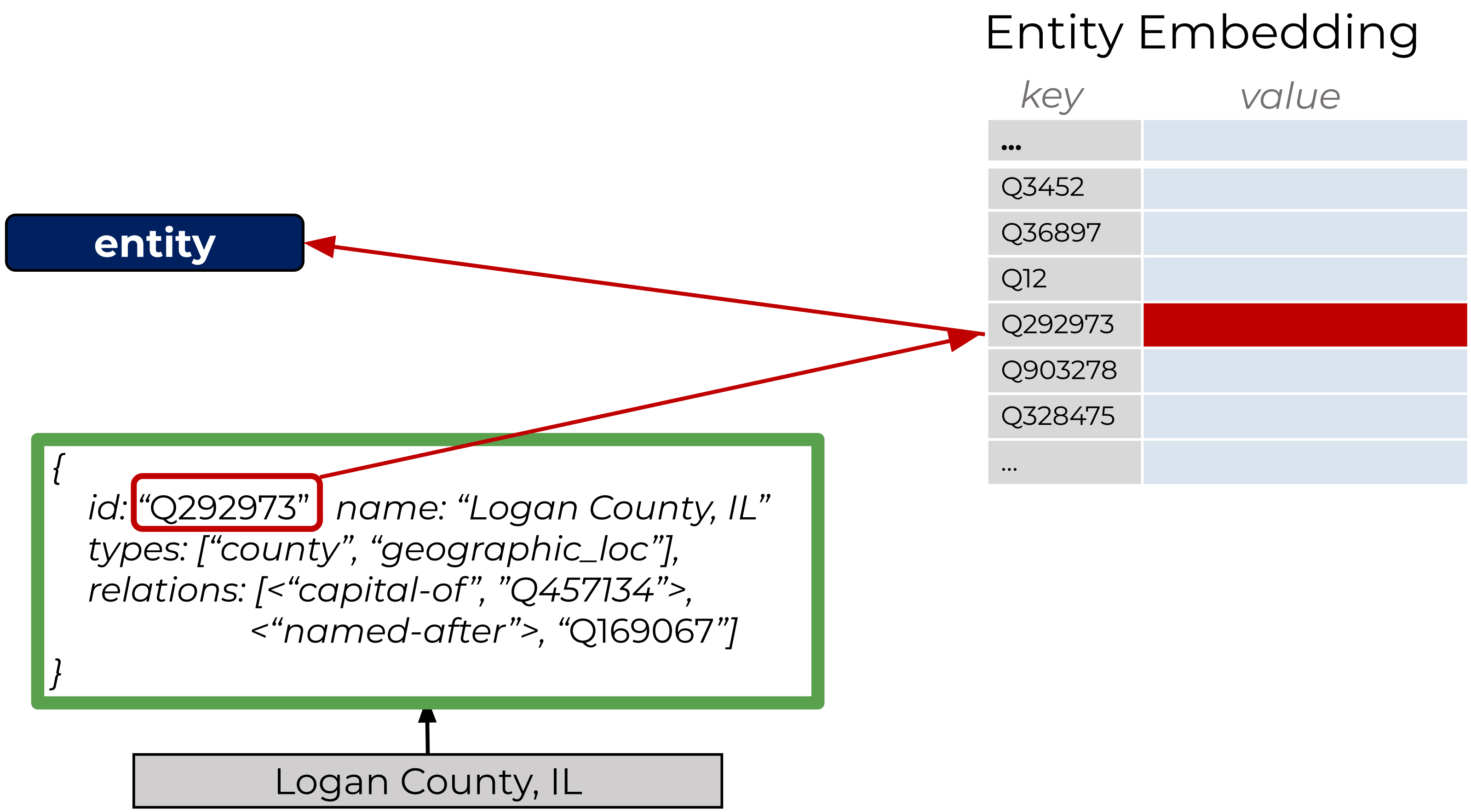
key	*value*
...	
Q3452	
Q36897	
Q12	
Q292973	
Q903278	
Q328475	
...	

{
id: “Q292973” , name: “Logan County, IL”
types: [“county”, “geographic_loc”],
relations: [<“capital-of”, ”Q457134”>,
<“named-after”>, “Q169067”]
}

Logan County, IL

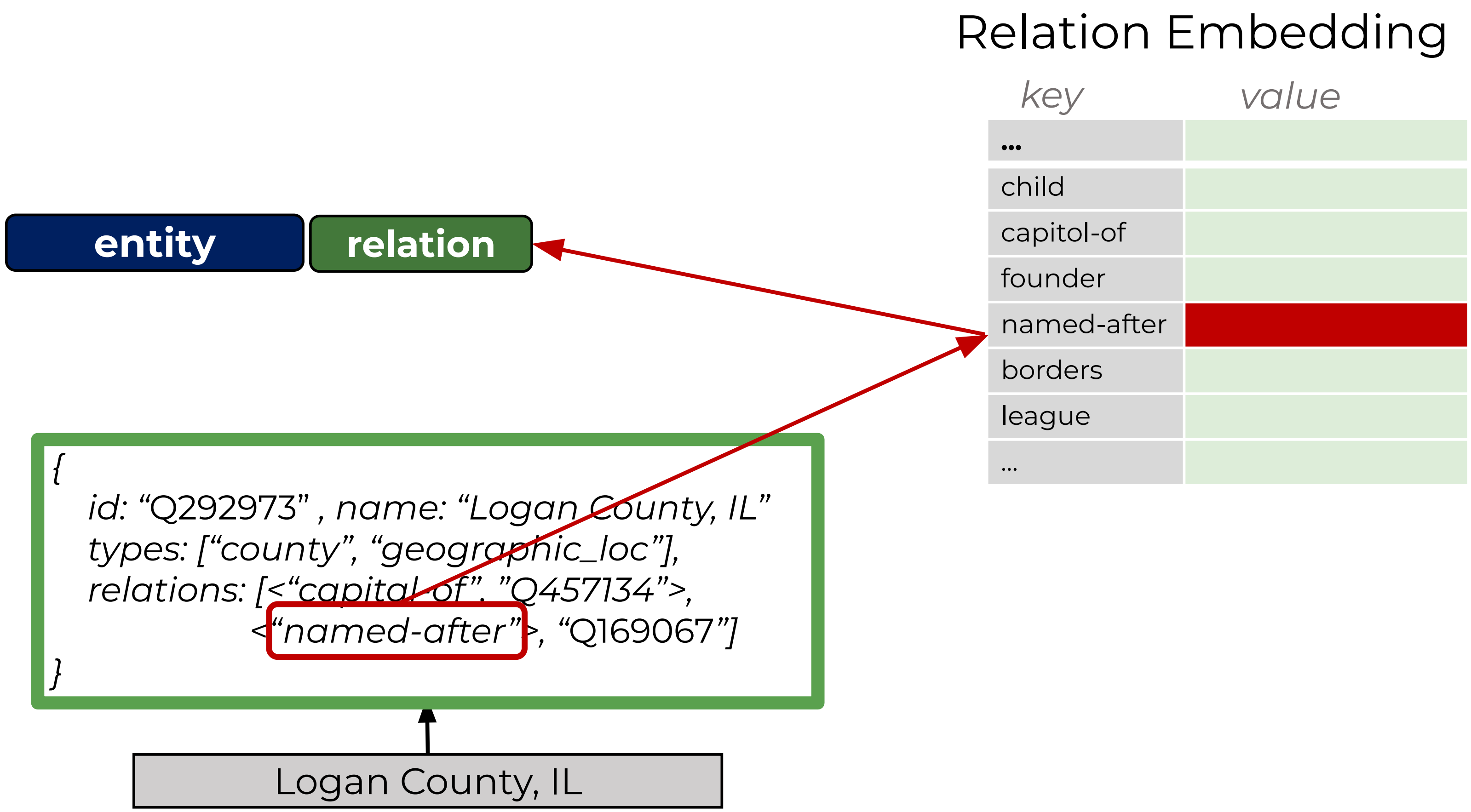
For each candidate, we use the entity profile to extract (learned) embeddings.

Using Embeddings to Encode Signals

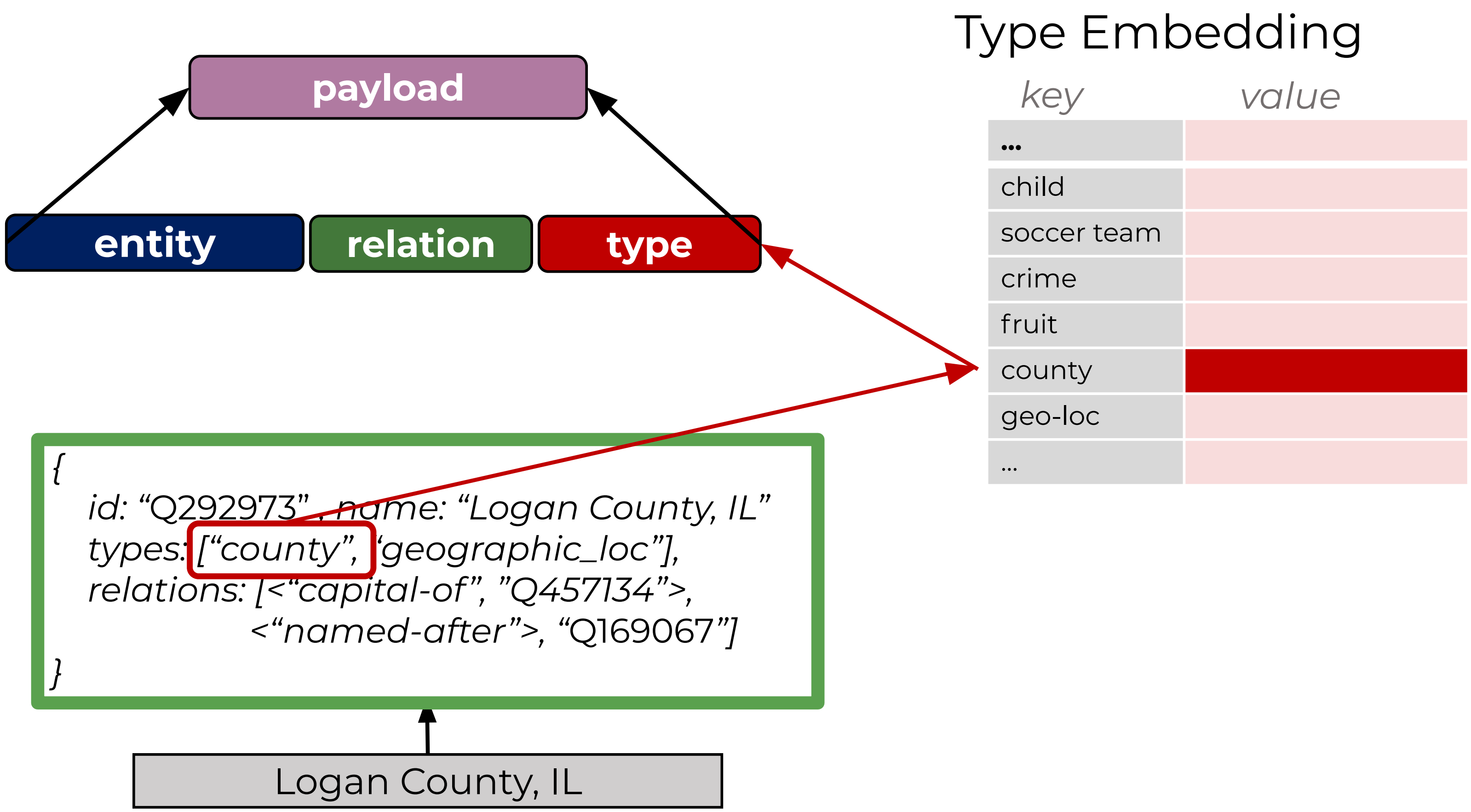


For each candidate, we use the entity profile to extract (learned) embeddings.

Using Embeddings to Encode Signals

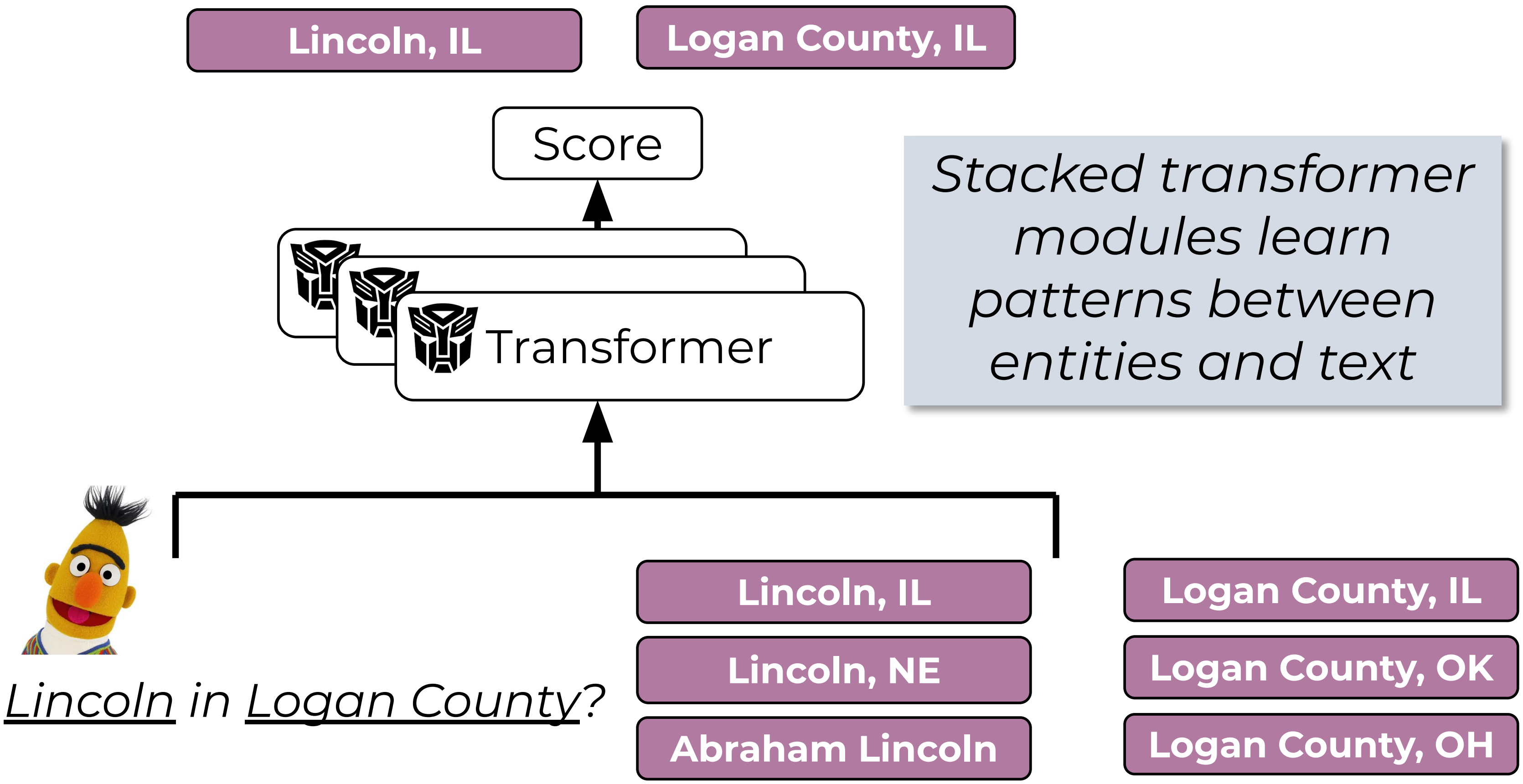


Using Embeddings to Encode Signals



The entity payload has embeddings mapping for each structural resource.

Bootleg Architecture



Simplest architecture that supports reasoning over types and relations.

Bootleg: Tail Performance

On the head, BERT-based baseline performs ~ 5 F1 points of Bootleg.

On the tail, Bootleg outperforms baseline by > 40 F1 points!

Evaluation Set	BERT NED Baseline	Bootleg	# Examples
All	85.9	**91.3**	4,066K
Torso Entities	79.3	**87.3**	1,912K
Tail Entities	27.8	**69.0**	163K
Unseen Entities	18.5	**68.5**	10K

Performance results on Wikipedia dataset.

Bootleg: Industrial Performance

Included Bootleg embeddings into an Overton production task answering millions of users' factoid queries. We report relative lift.

Evaluation Set	English	Spanish	French	German
All Entities	1.08	1.03	1.02	1.00
Tail Entities	1.08	1.17	1.05	1.03

Using Bootleg Downstream: SoTA on the TACRED Benchmark

Goal: extract the relationship between a subject and object pair.

Gold relation: per:employee_of

(subject) (object)

Mays worked with several other companies aside from **Media Enterprises** in his career.

Micro-Avg. F1 on TACRED Revised test dataset:

Model	Test F1 Score
SpanBERT	78.0
KnowBERT	79.3
Bootleg+SpanBERT	80.2 (SoTA)

Zhang et al., 2017 and Hennig et al., 2020.

Bootleg resolves errors made in by the prior SoTA model.

Gold relation: per:cause_of_death

(subj) (obj)

Vincent Astor , like **Marshall** , died unexpectedly of a **heart attack** in 1959 ...

Q299001 Q12152

Wikidata relation: ['cause of death']

SpanBERT no_relation

BOOTLEG per:cause_of_death

Leveraging type and relation information downstream

Gold relation: org:alternate_names

(obj) (subj)

The **International Water Management Institute** or **IWMI** study said both ...

Q868028 Q868028

Wikidata same entity

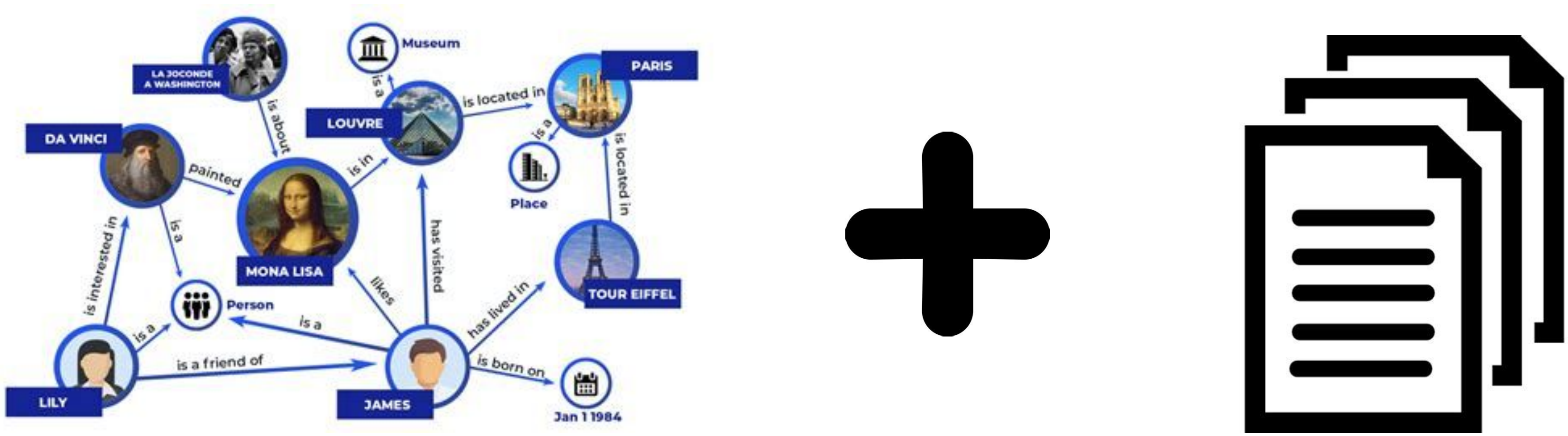
SpanBERT no_relation

BOOTLEG org:alternate_names

Understand that sub-strings relate to the same entity

Self-Supervised Data Take Away

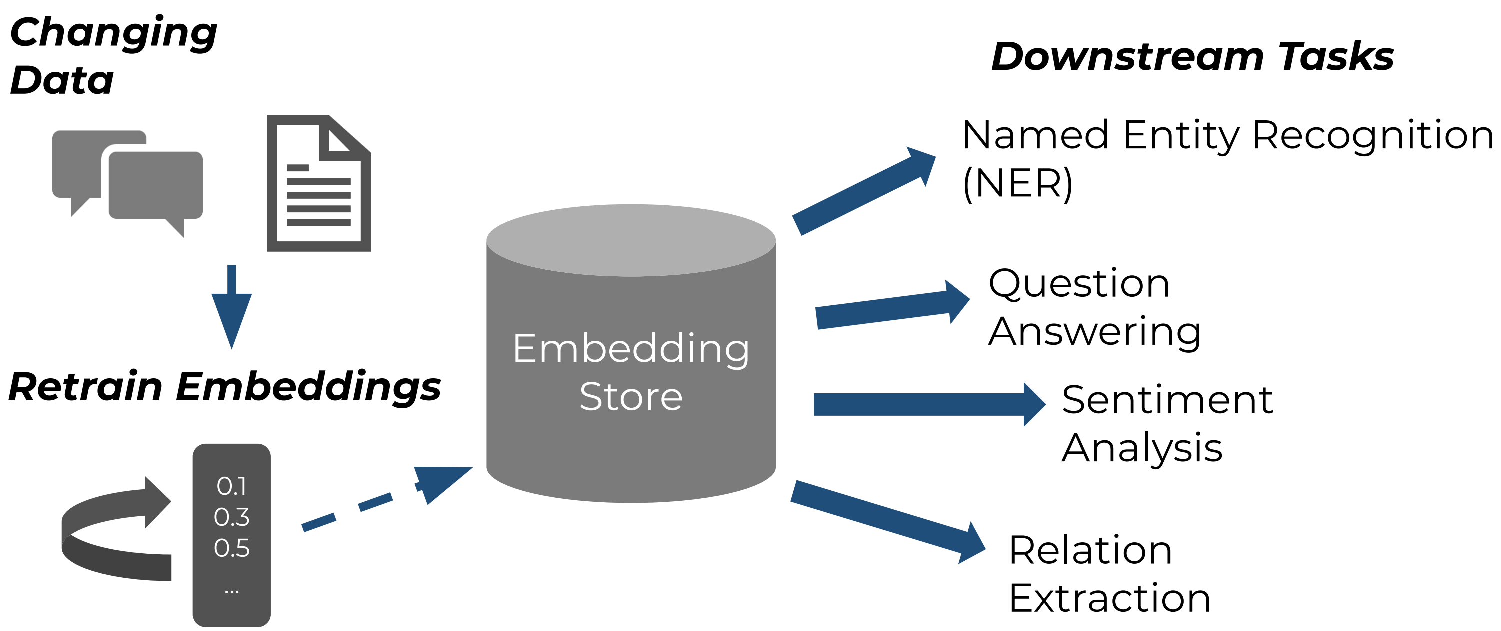
Self-supervised data does not well represent tail distributions -> embeddings may not be high quality for rare entities



Solution: merged unstructured data with structured knowledge that can generalize to the tail.

Embedding Management: Stability

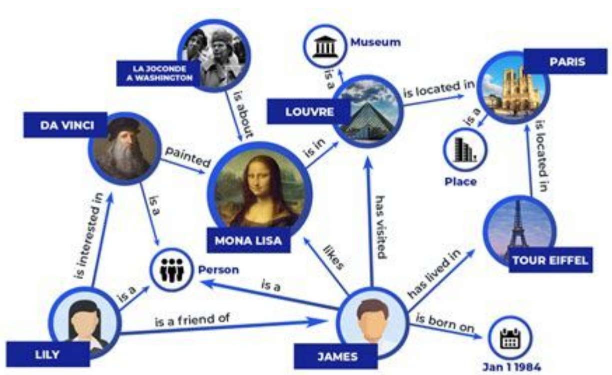
Problem Setting: Embedding Store



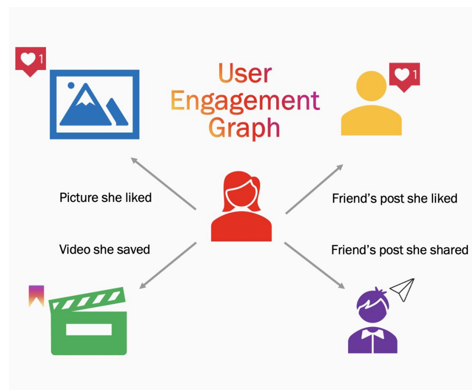
New embeddings require downstream tasks to be retrained!

Why do embeddings need to be retrained?

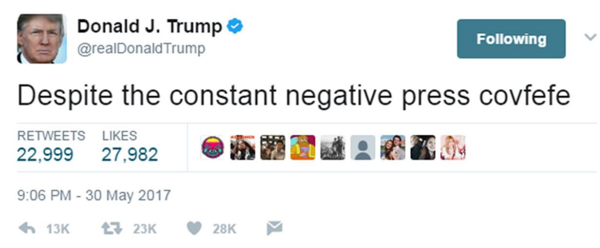
Learn new entities



Leverage new activity[1]



Understand new words



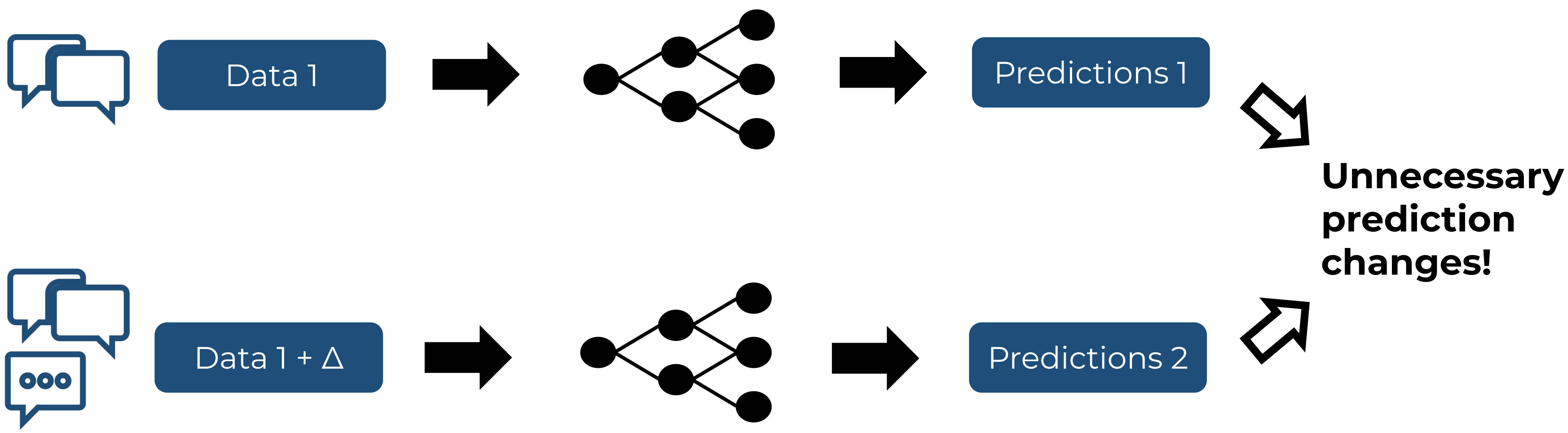
Model freshness is necessary for user satisfaction in many products.

[1] https://about.instagram.com/blog/engineering/designing-a-constrained-exploration-system

Google retrains their app store Google Play models *every day*, and Facebook retrains search models *every hour*.

[1] Baylor et al. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. KDD, 2017.
[2] Hazelwood et al. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. HPCA, 2018.

But model training can be unstable…



Prediction churn

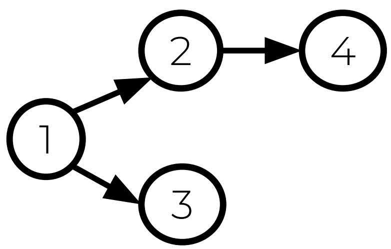
[1] Cormier et al. Launch and Iterate: Reducing Prediction Churn. NeurIPS, 2016.

Challenges of Instability

Debugging

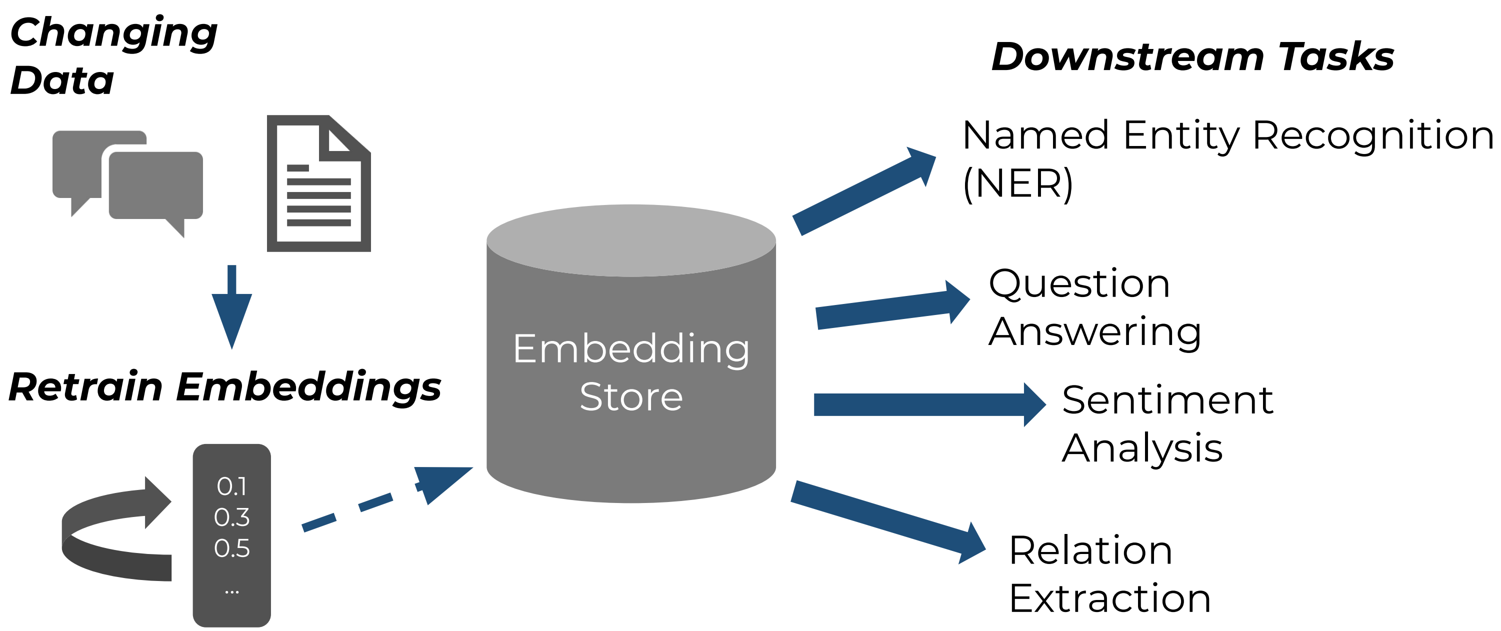
Consistent user-experience

Model dependencies



Research reliability

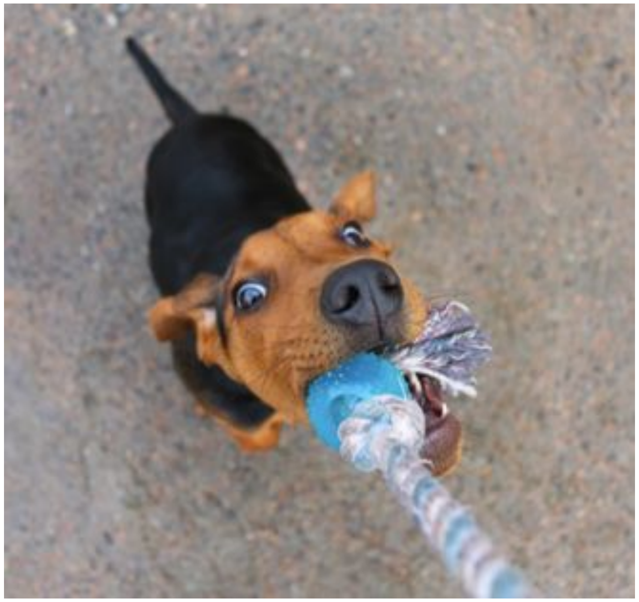
Problem Setting: Embedding Store



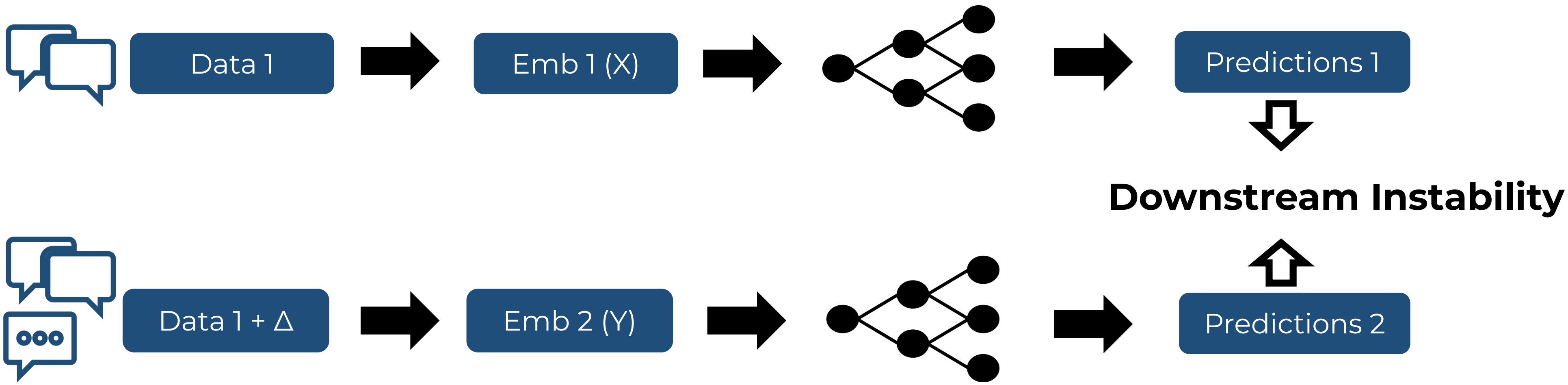
How does the embedding instability propagate to downstream tasks?

Outline

- Downstream instability definition
- Stability-space tradeoff
- Measuring embedding quality with distance measures



Definition: Downstream Instability



Downstream instability = **% prediction disagreement** between models trained on a pair of embeddings

Embedding Hyperparameters that Impact Storage

Dimension
features / word

Precision
bits / feature

=

Embedding Size

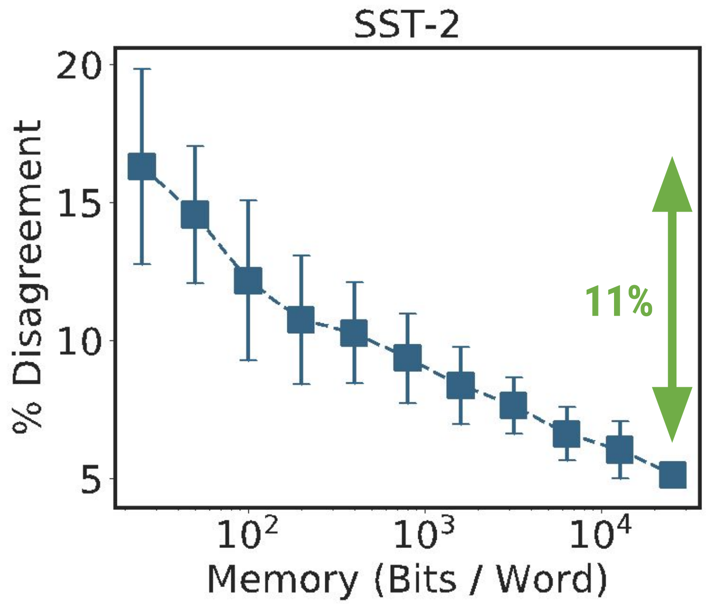
Uniform Quantization	32-bit		1-bit
	0.04		**0.1**
Interval: [-0.1, 0.1]	**-0.03**	→	**-0.1**
	-0.08		**-0.1**

Downstream Instability

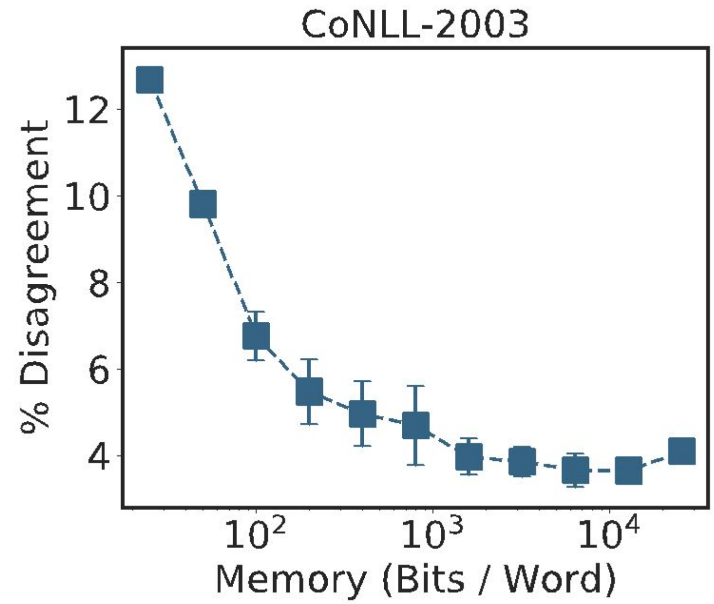
[1] May et al. On the downstream performance of compressed word embeddings. NeurIPS, 2019.

Stability-Space Tradeoff for Word Embeddings

Sentiment Analysis

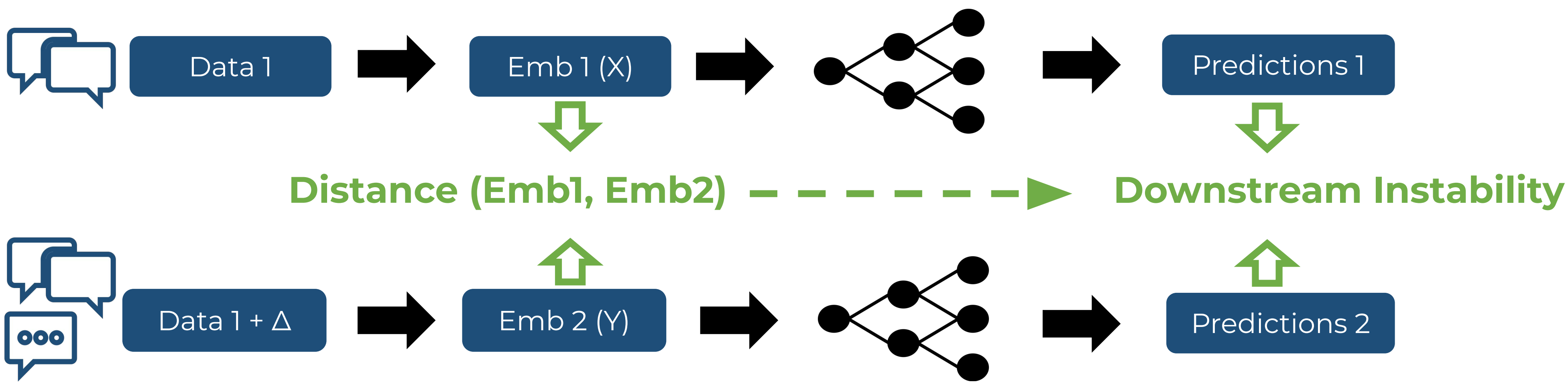


NER



Embedding Size **Downstream Instability**

Goal: Embedding Distance Measure for Instability



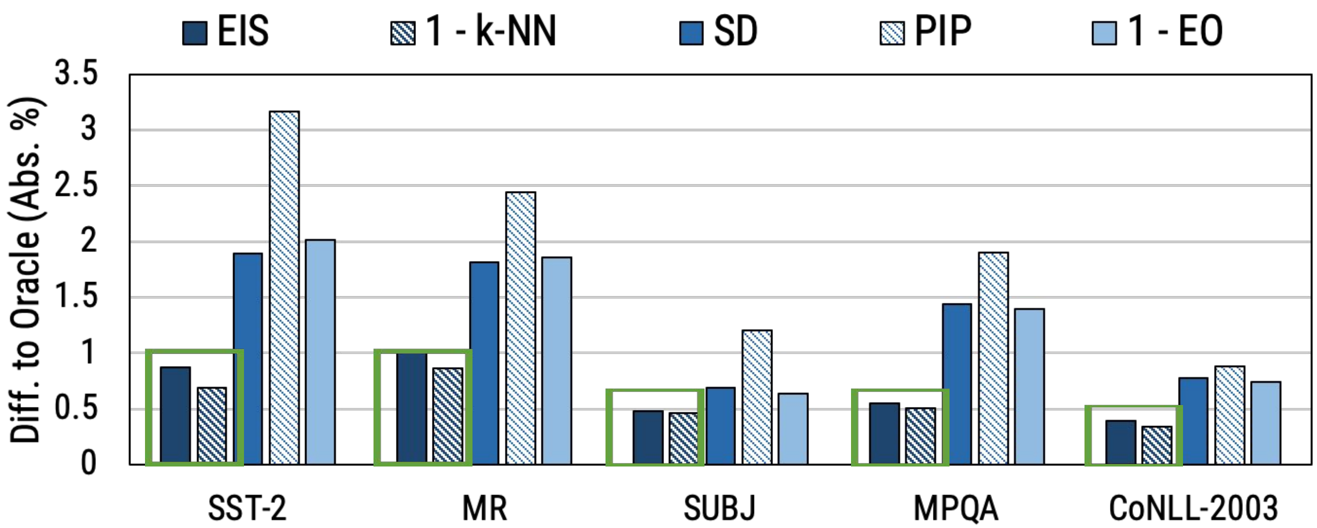
The measure should relate the **distance between the embeddings** to the **downstream instability**.

Embedding Distance Measures

- k-NN measure [1,2,3]
- Semantic displacement (SD) [4]
- PIP loss [5]
- Eigenspace overlap (EO) [6]
- Eigenspace instability measure (EIS) [7]

[1] Hellrich & Hahn, COLING, 2016; [2] Antoniak & Mimno, TACL, 2018; [3] Wendlandt et al., NAACL-HLT, 2018; [4] Hamilton et al., ACL, 2016; [5] Yin & Shen, NeurIPS, 2018; [6] May et al., NeurIPS, 2019; [7] Leszczynski et al., MLSys 2020

Using Embedding Distance Measures to Minimize Downstream Instability



k-NN measure and **theoretically grounded EIS measure** outperform other measures for selecting embeddings to **minimize downstream instability**.

Stability Takeaways

- Defined downstream instability with respect to embeddings
- Stability-space tradeoff (precision, dimension)

Embedding Size **Downstream Instability**

- Measuring embedding quality with embedding distance measures
  - EIS and k-NN measures select embeddings with lower downstream instability

Closing the Loop of Model Development: Monitoring and Patching

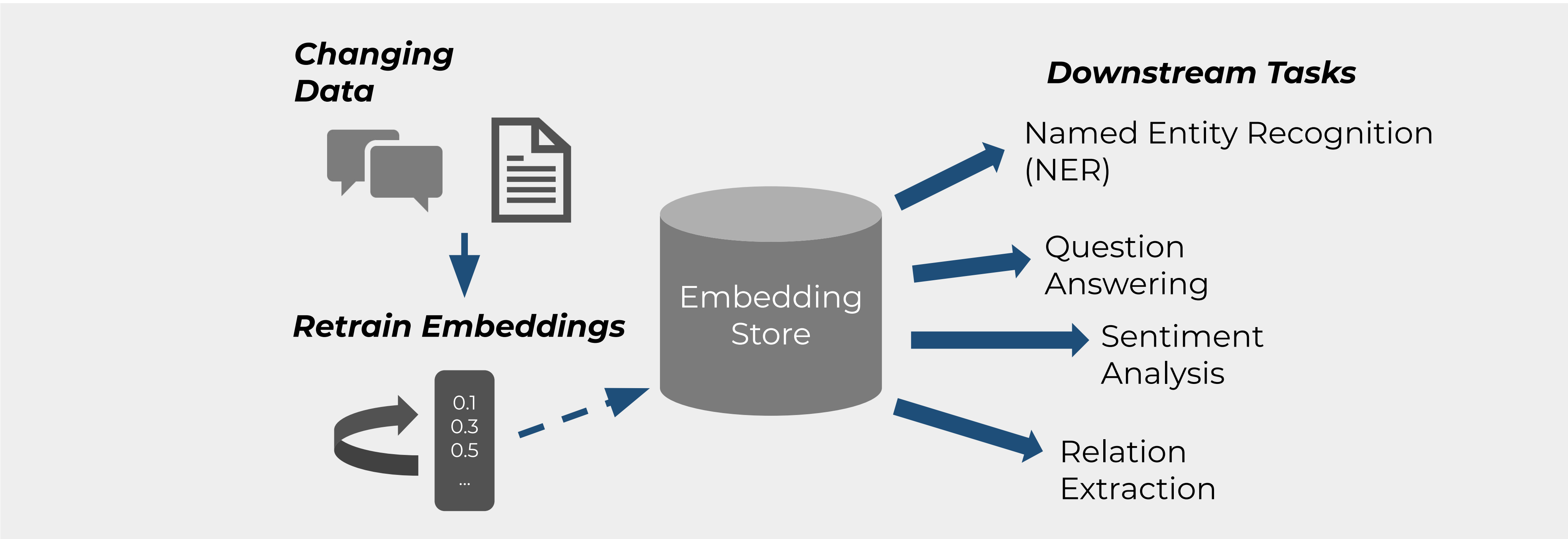
Monitoring and Patching

Embeddings need to be updated: distribution shift, changing needs

Monitor (when to update)
Evaluate and track distribution shift

Patch (how to update)
Fix bugs and improve performance

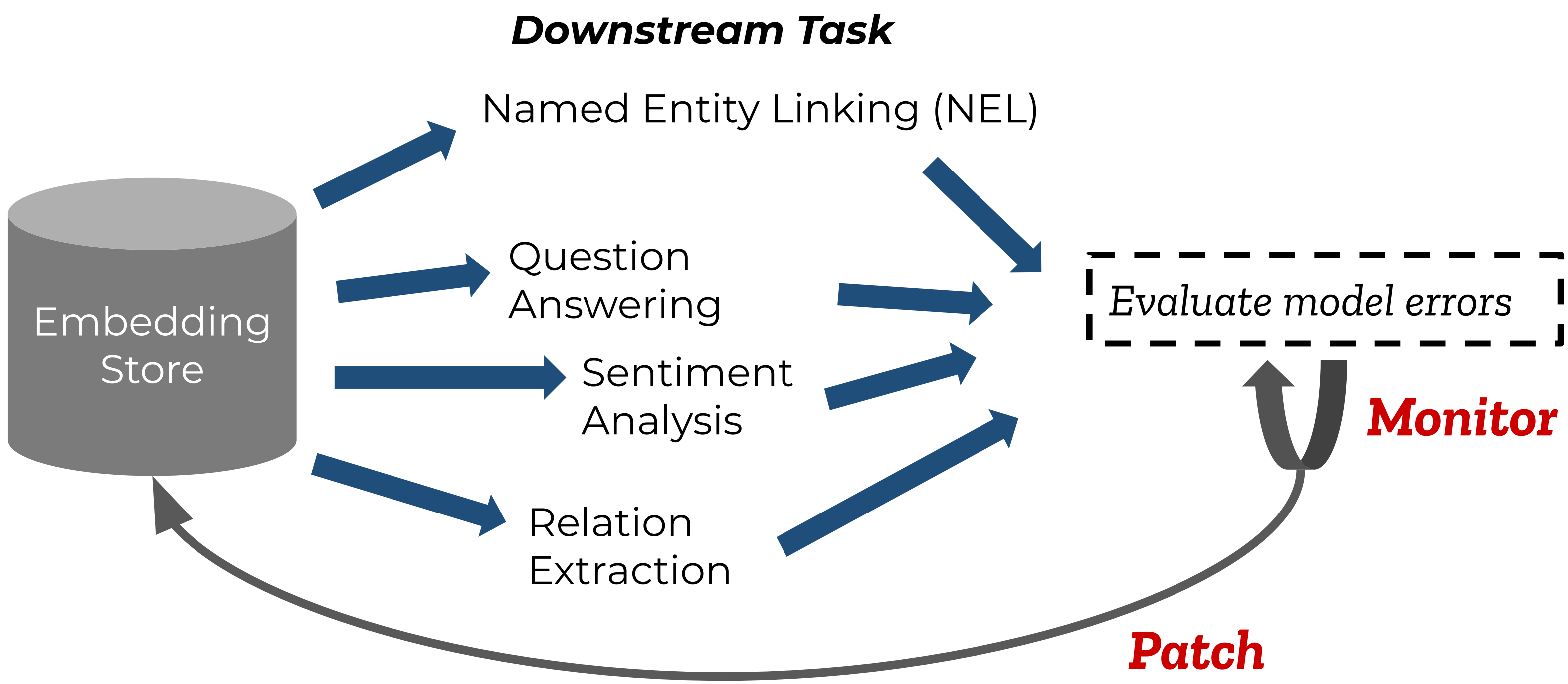
Remember: Embedding Store



Important: update **embeddings not downstream models →** changes propagate down to models!

Crucial Bottleneck: Evaluation

Can't monitor and patch embeddings without evaluation



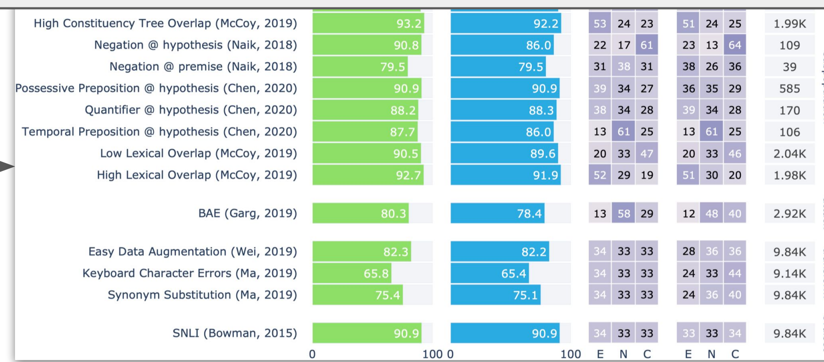
Crucial Bottleneck: Evaluation

Can't monitor and patch embeddings without evaluation

Many Evaluation Strategies

- Critical data slices
- Bias / fairness concerns
- Sensitivity to perturbations
- Invariance to transformations
- and more!

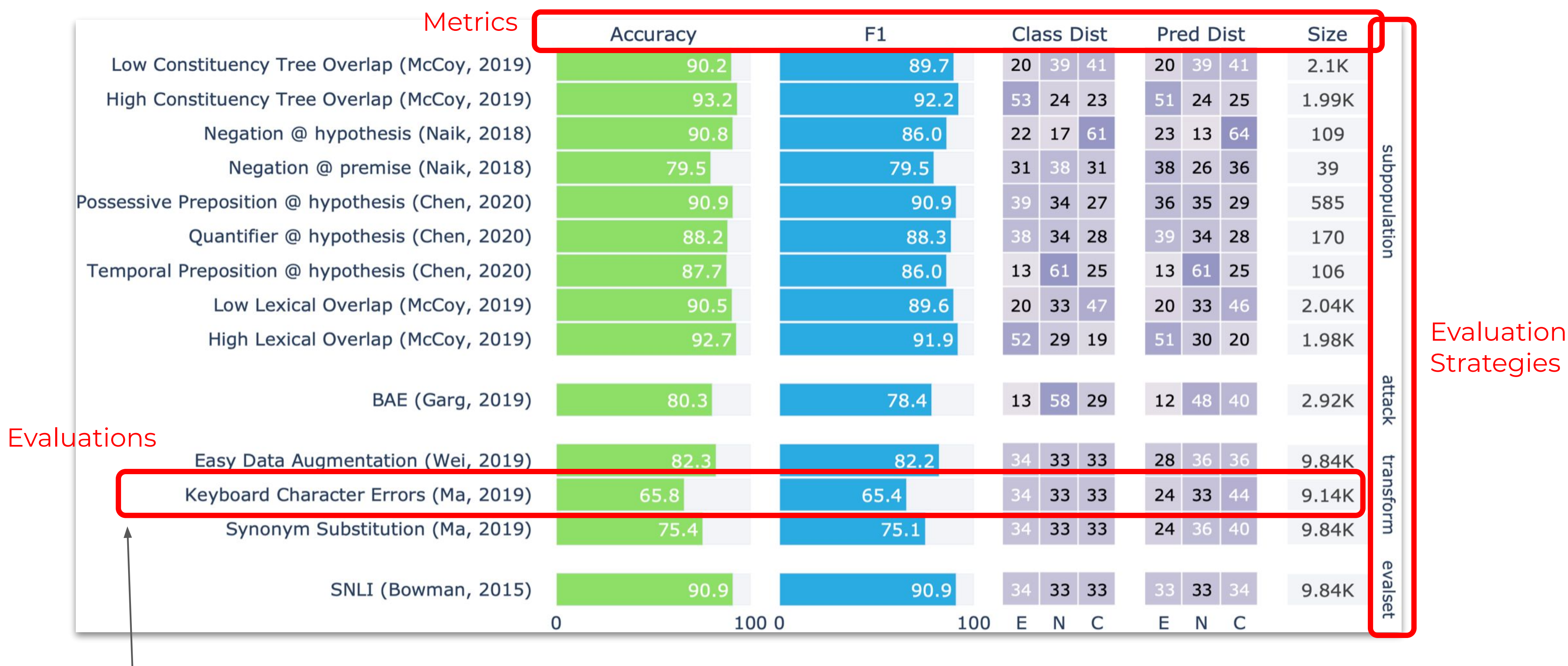
Fine-Grained Evaluation Metrics



Shift towards **fine-grained evaluation** with new tools (e.g. Robustness Gym, Dynabench)

Tool: Robustness Gym

Consolidates different evaluation strategies (slices, transformations) and metrics

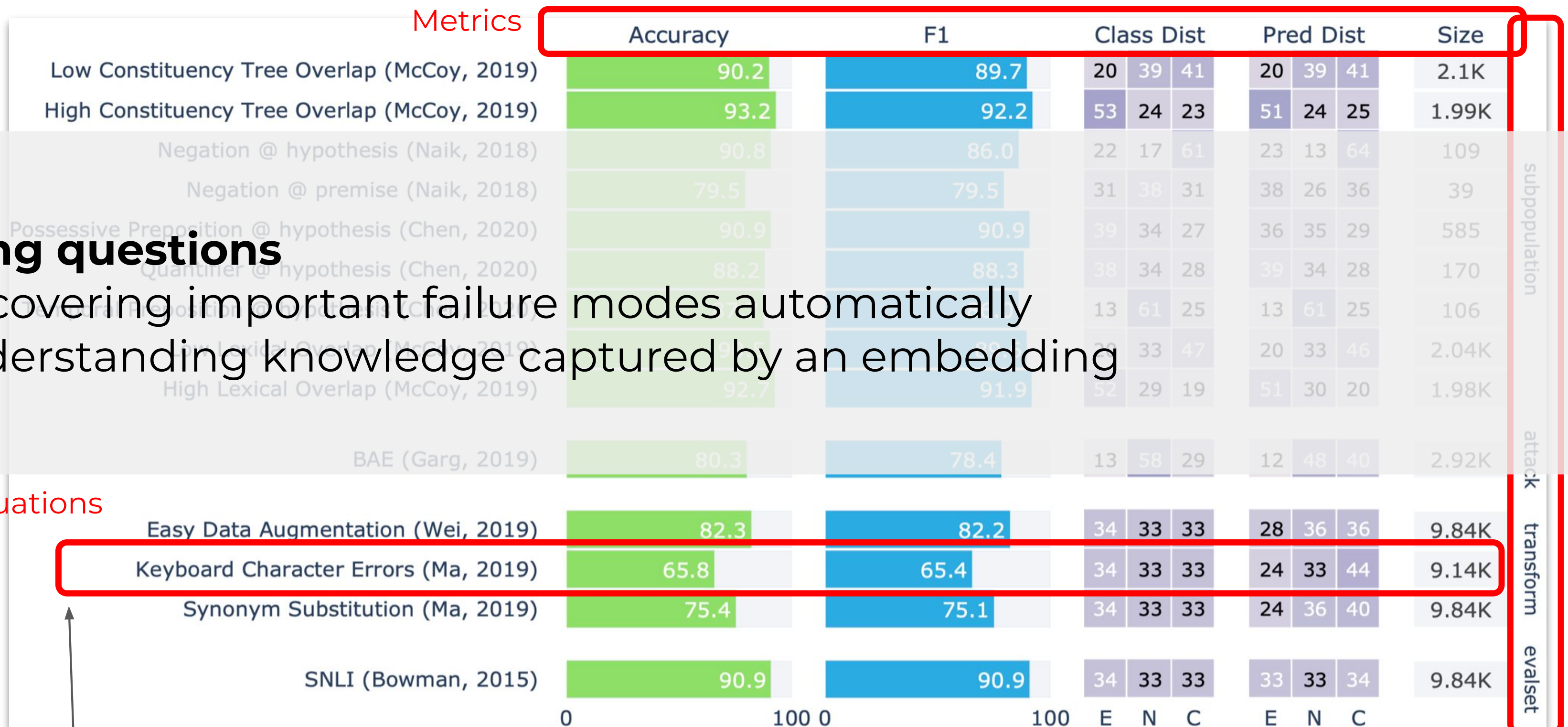


Example: BERT embeddings are sensitive to character errors

Robustness Gym: Unifying the NLP Evaluation Landscape. Goel et al. *NAACL Demo 2021.*

Tool: Robustness Gym

Consolidates many different evaluation types (subpopulations, transformations) and metrics



Emerging questions

- Discovering important failure modes automatically
- Understanding knowledge captured by an embedding

Example: BERT embeddings are sensitive to character errors

Robustness Gym: Unifying the NLP Evaluation Landscape. Goel et al. *NAACL Demo 2021.*

Important Evaluation Strategy: Slice-Based Evaluation

A type of fine-grained evaluation

→ Measure fine-grained performance on critical subpopulations (filtering)

Example:

short passages (< 50 words) in a text dataset

Example: Named Entity Linking

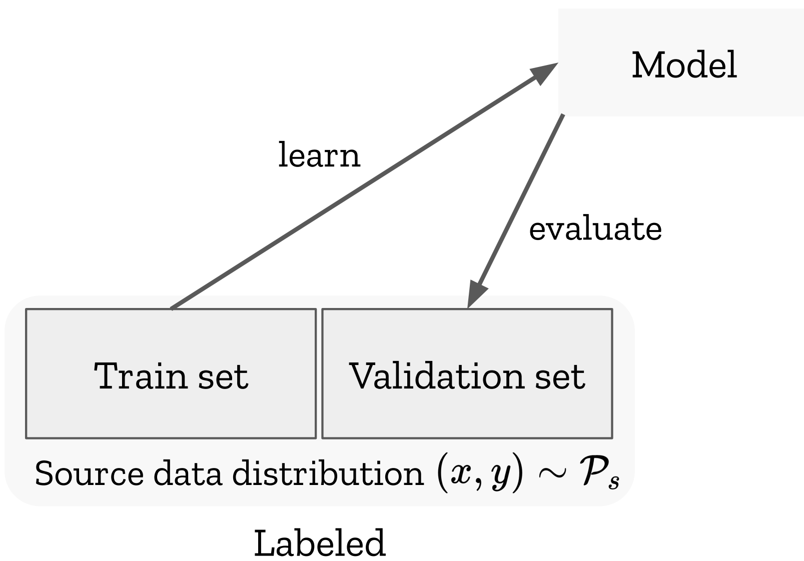
Robustness Gym: Unifying the NLP Evaluation Landscape. *NAACL Demo 2021.*

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. *NAACL Industry 2021.*

Most Named Entity Linking systems are poor on rare entities

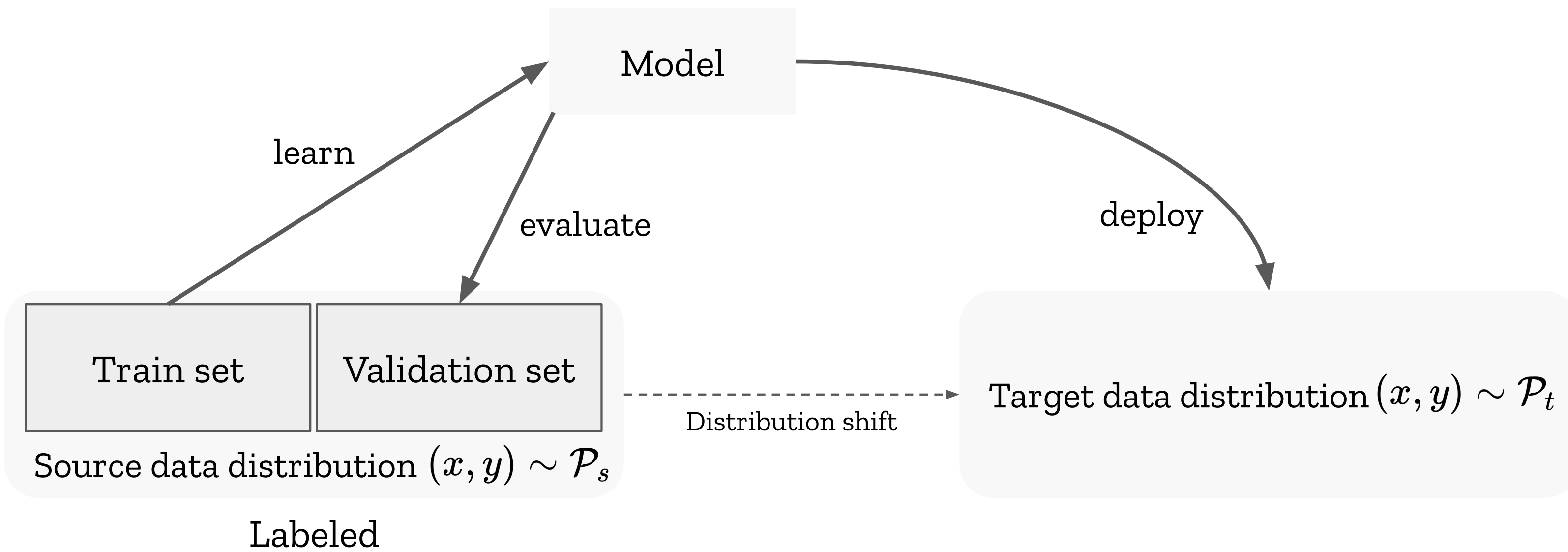
Evaluation over Time: **Monitoring**

Continually evaluate as the world changes



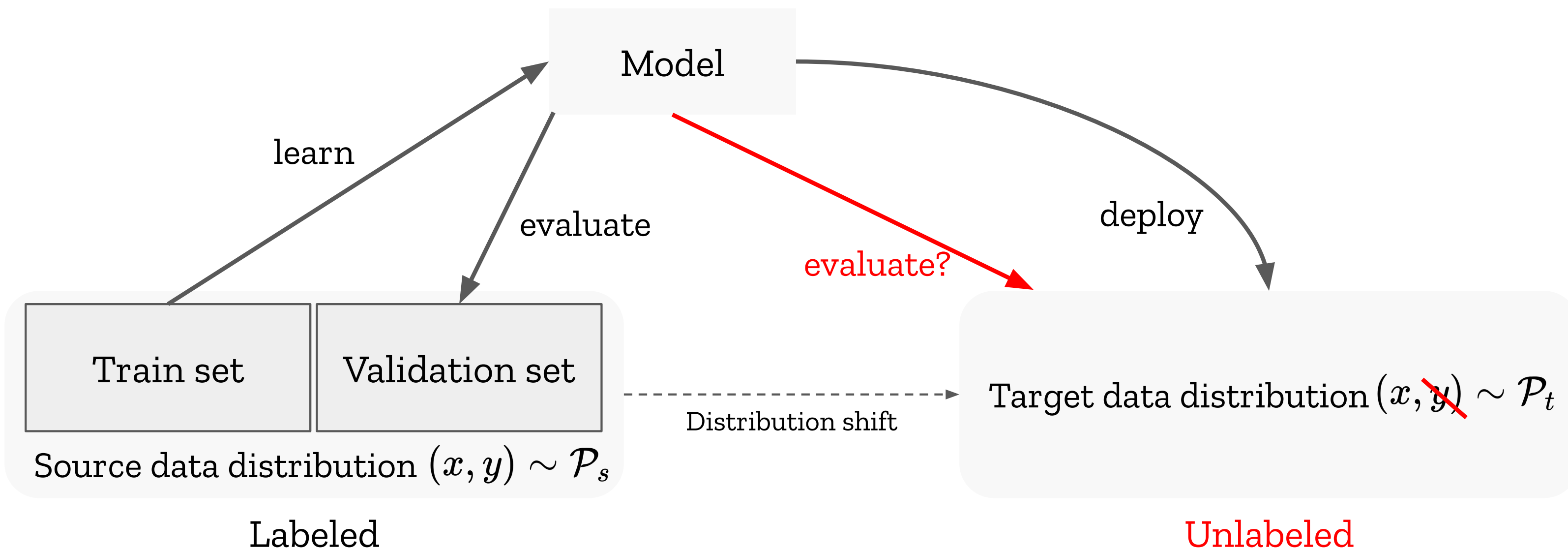
Evaluation over Time: **Monitoring**

Continually evaluate as the world changes



Evaluation over Time: **Monitoring**

Continually evaluate as the world changes



Need to monitor model performance on unlabeled data

Approach: Importance Weighting

Estimate metrics on incoming data

Upweight examples in our dev set more likely to be seen in the future

Theoretical Foundations

Density ratio estimation (Sugiyama, 2012)

Recent work: accurately estimate performance with *slice-based evaluation* + *importance weighting*

Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021.*

Mandoline: Slice-based reweighting framework

Slice: user-defined grouping of data $g(x) \in \{-1, 1\}$

negation
contains not, n't
$g_1(x)$

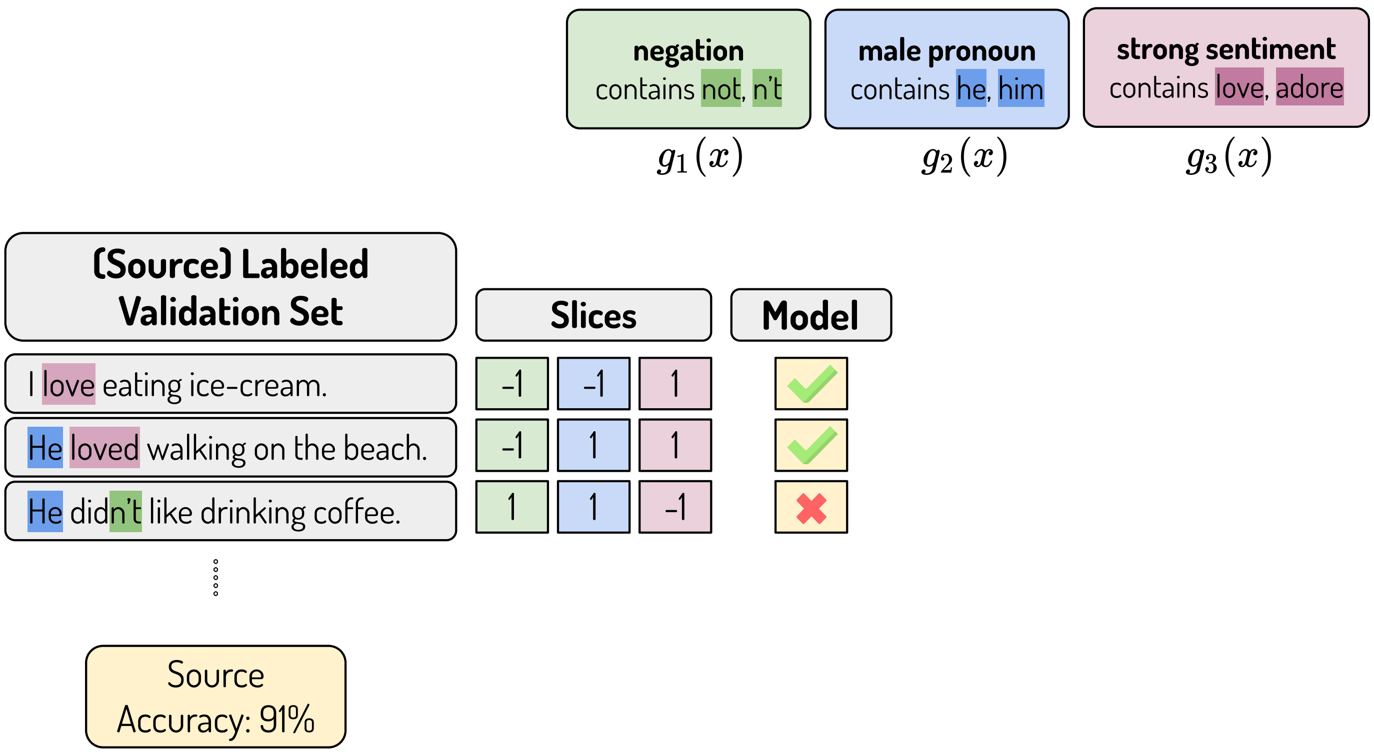
male pronoun
contains he, him
$g_2(x)$

strong sentiment
contains love, adore
$g_3(x)$

Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021.*

Mandoline: Slice-based reweighting framework

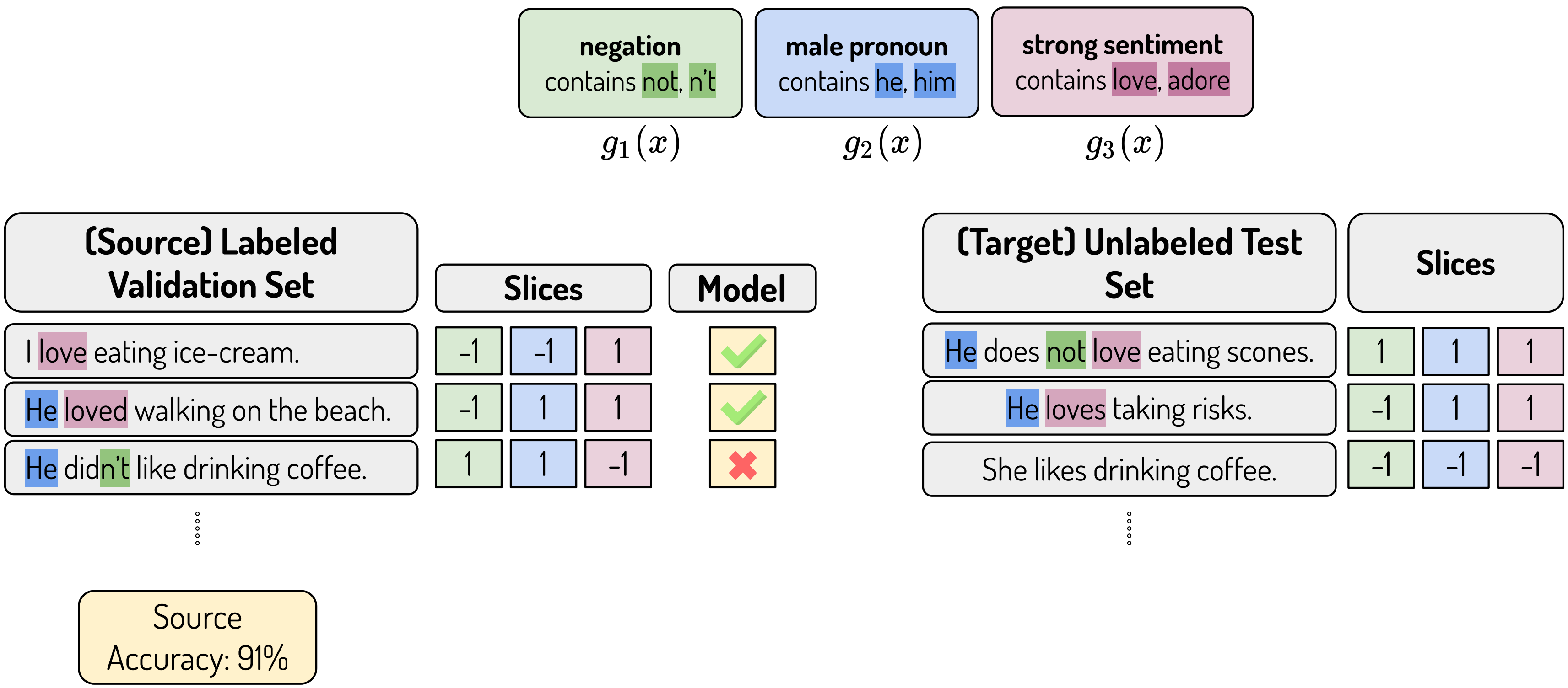
Slice: user-defined grouping of data $g(x) \in \{-1, 1\}$



Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021*.

Mandoline: Slice-based reweighting framework

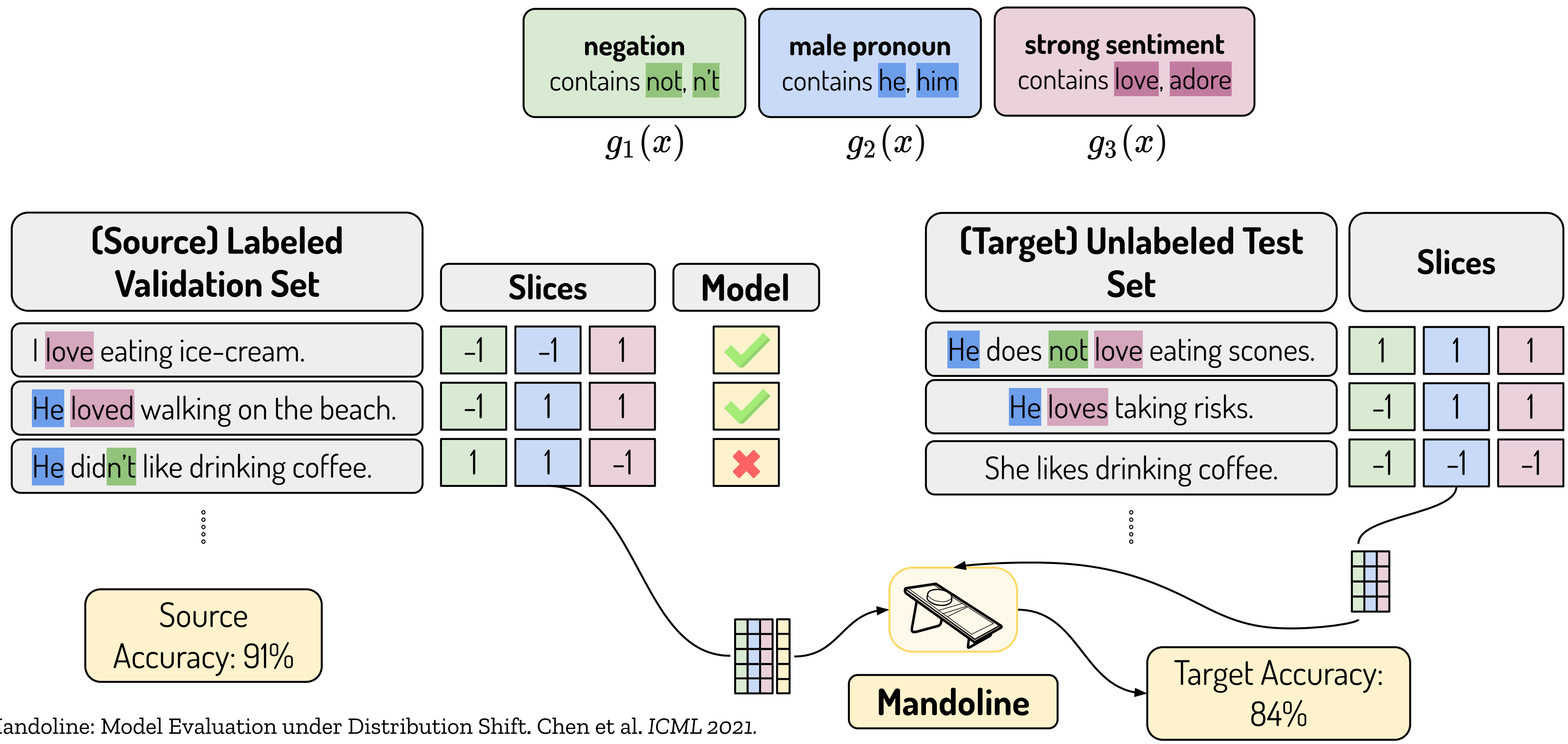
Slice: user-defined grouping of data $g(x) \in \{-1, 1\}$



Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021.*

Mandoline: Slice-based reweighting framework

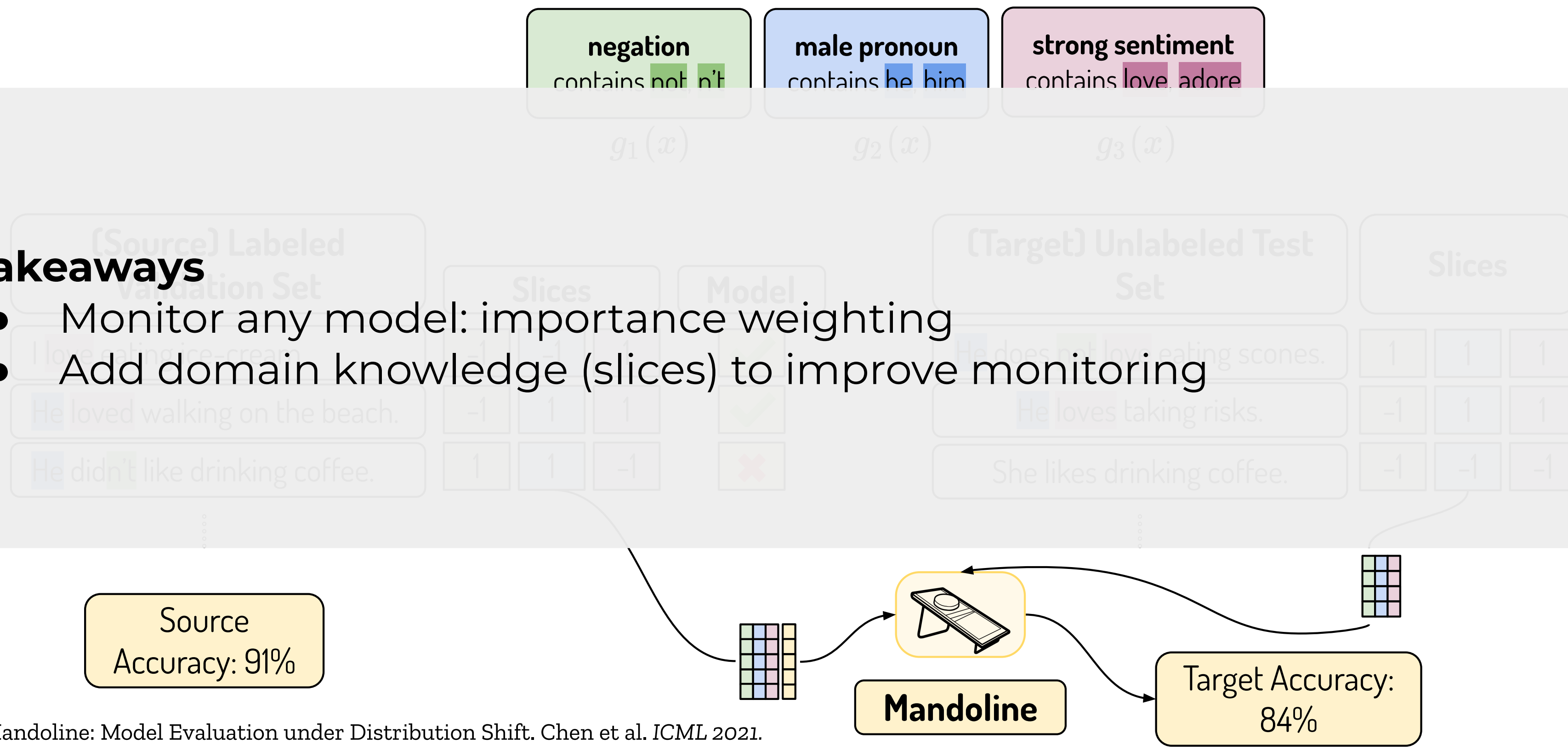
Slice: user-defined grouping of data $g(x) \in \{-1, 1\}$



Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021*.

Mandoline: Slice-based reweighting framework

Slice: user-defined grouping of data $g(x) \in \{-1, 1\}$



Takeaways

- Monitor any model: importance weighting
- Add domain knowledge (slices) to improve monitoring

Mandoline: Model Evaluation under Distribution Shift. Chen et al. *ICML 2021*.

Embedding Model **Patching**

Once errors are identified, need to retrain or update embeddings

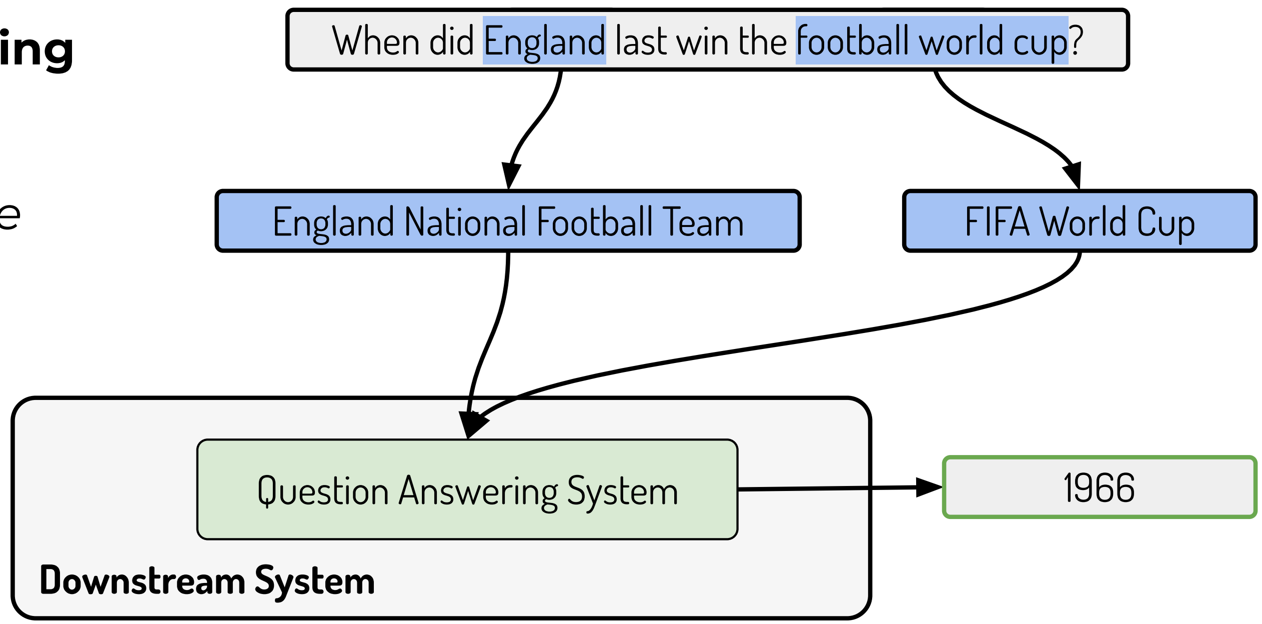
Many Approaches

Data-Centric	**Model-Centric**
Data Augmentation	Training Paradigm
Data Collection	Training Algorithm
Active Sampling	Features
Weak Labeling	Architecture
Data Preprocessing	

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. Goel et al. *NAACL Industry 2021.*

Named Entity Linking

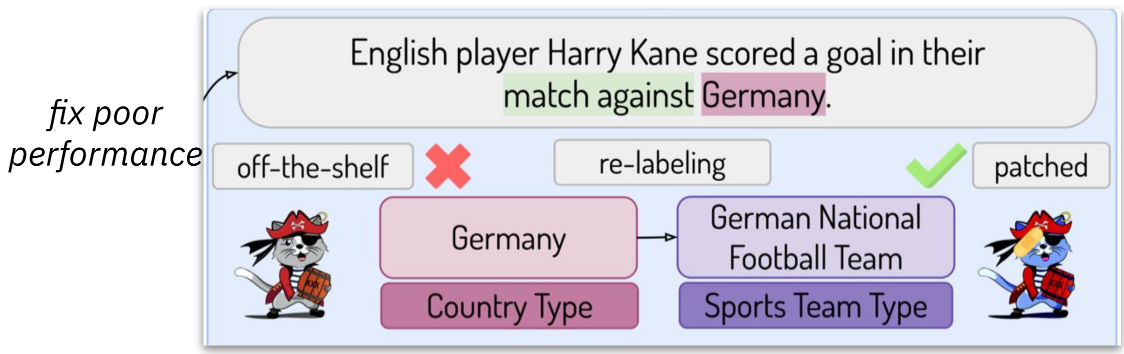
Named Entity Linking
map "strings" to "things" in a knowledge base like Wikipedia



A correct NEL is required for the downstream system!

End to End Example: Named Entity Linking

Repurposing Bootleg NEL system to patch errors for sports QA



Sports QA: prefer if the model predicted the national sports team instead of the country!

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. Goel et al. *NAACL Industry 2021.*

End to End Example: Named Entity Linking

Repurposing Bootleg NEL system to patch errors for sports QA

fix poor performance

Relabel the training set with a simple heuristic

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. Goel et al. *NAACL Industry 2021.*

End to End Example: Named Entity Linking

Repurposing Bootleg NEL system to patch errors for sports QA

Wikipedia examples with mentions of countries and sports teams

Subpop.	Gold Label	Pred. Label	Size (Off-The-Shelf $\rightarrow$ Patched)
	Team	Country	$151 \rightarrow 106$ ($\downarrow$)
		Team	$3393 \rightarrow 3447$ ($\uparrow$)

25% absolute accuracy improvement in sports-related errors

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. Goel et al. *NAACL Industry 2021.*

Embedding Model **Patching**

Once errors are identified, need to retrain or update embeddings

Many Approaches

Data-Centric	Model-Centric
Data Augmentation	Training Paradigm
Data Collection	Training Algorithm
Active Sampling	Features
Weak Labeling	Architecture
Data Preprocessing	

New area of research!

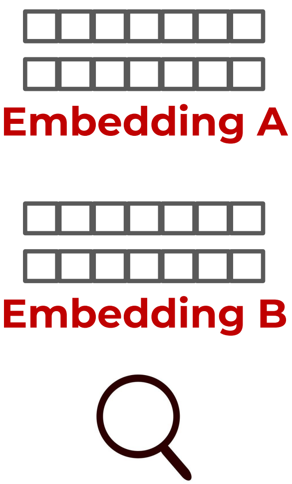
- Incremental and targeted embedding updates
- Backwards compatibility for updated embeddings e.g. stability
- Data-centric vs. model-centric updates
- Sample efficiency and efficacy of approaches
- Time-to-update and optimal cadence

Goodwill Hunting: Analyzing and Repurposing Off-the-Shelf Named Entity Linking Systems. Goel et al. *NAACL Industry* 2021.

Future Directions

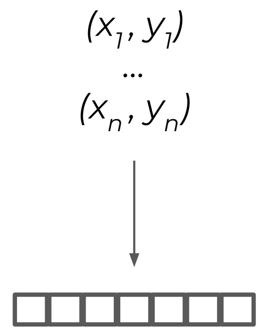
Embeddings as First Class Citizens

What is the right system for embedding management in ML pipelines?



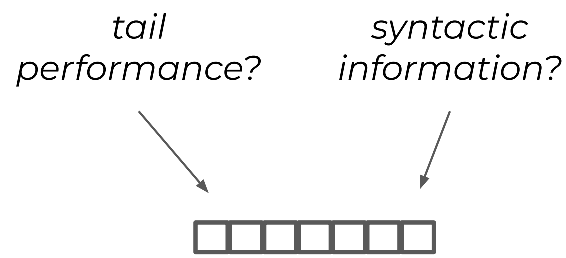
Search

What set of embeddings are best for a specific task?



Provenance

What data had the most “impact” on these embeddings?

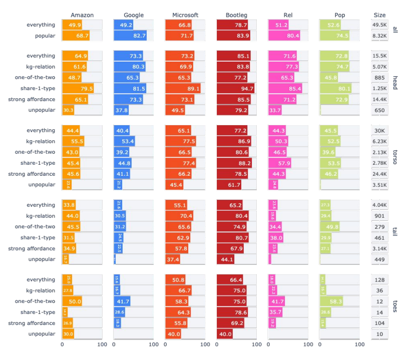


Quality

What are the right metrics/probes for embedding quality?

End-to-End Model Patching

How can we automate and provide guidance for embedding patching?



What are the current failure modes?

Data Augmentation

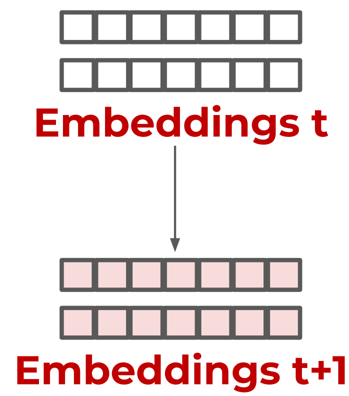
Data Collection

Active Sampling

Weak Labeling

Data Preprocessing

What data engineering strategy to use?



How do I update my models efficiently?

https://github.com/robustness-gym/meerkat

Interactive Machine Learning

How can we facilitate human interaction with model training and evaluation data?

Construct Data

Measure

Monitor

Maintain

How can we support the entire lifecycle?

Text

Images

Relational

Video

How to integrate multiple modalities?

meerkat

Data Panels for ML

Video, Image, Model Outputs, ...

Systems to store and manage models and data?