Build a Data Annotation Tool using AL and TL

Final Presentation

Thursday, August 19, 2021

Presented by: Yongchao Zhou

VECTOR INSTITUTE

Motivation

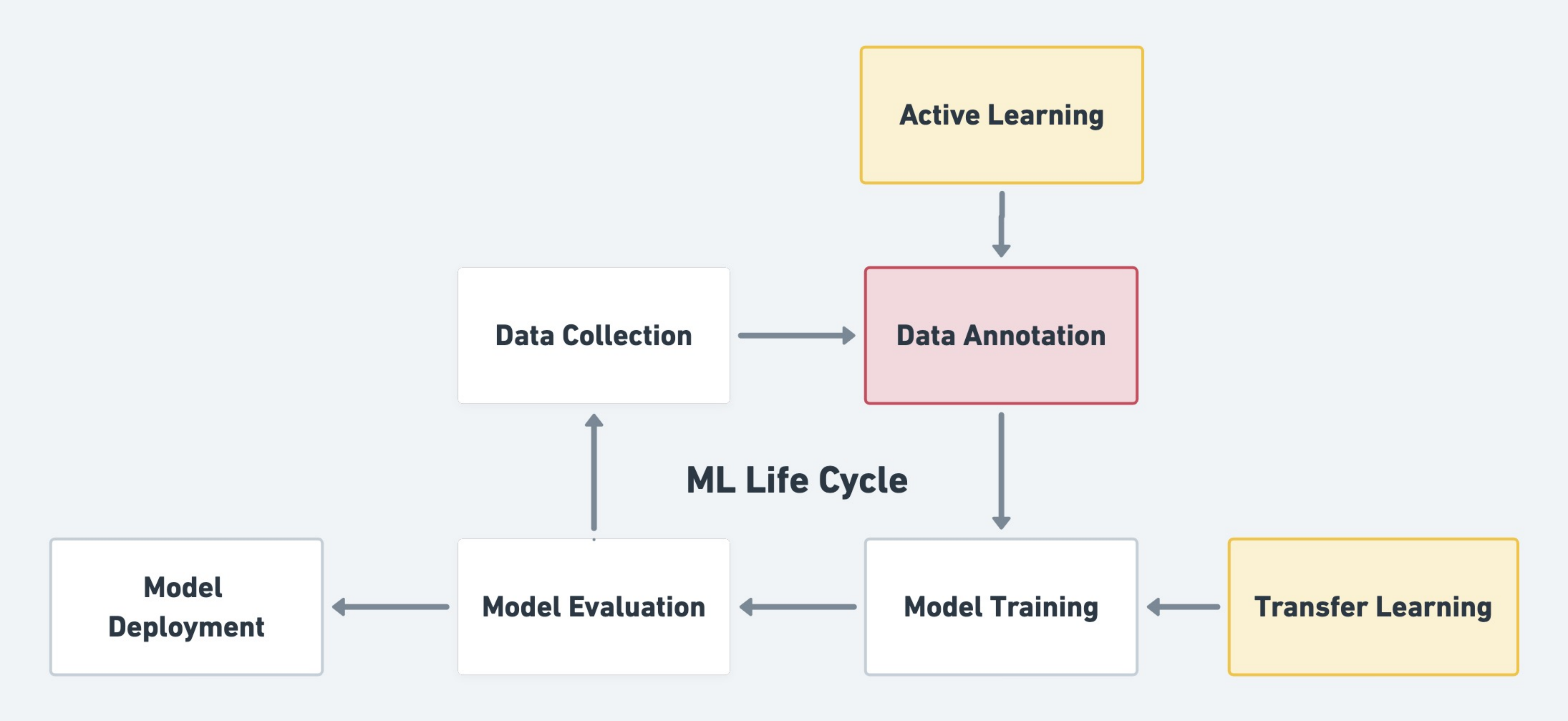

Figure 1: Machine Learning Life Cycle

Motivation

How to build a label-efficient ML System?

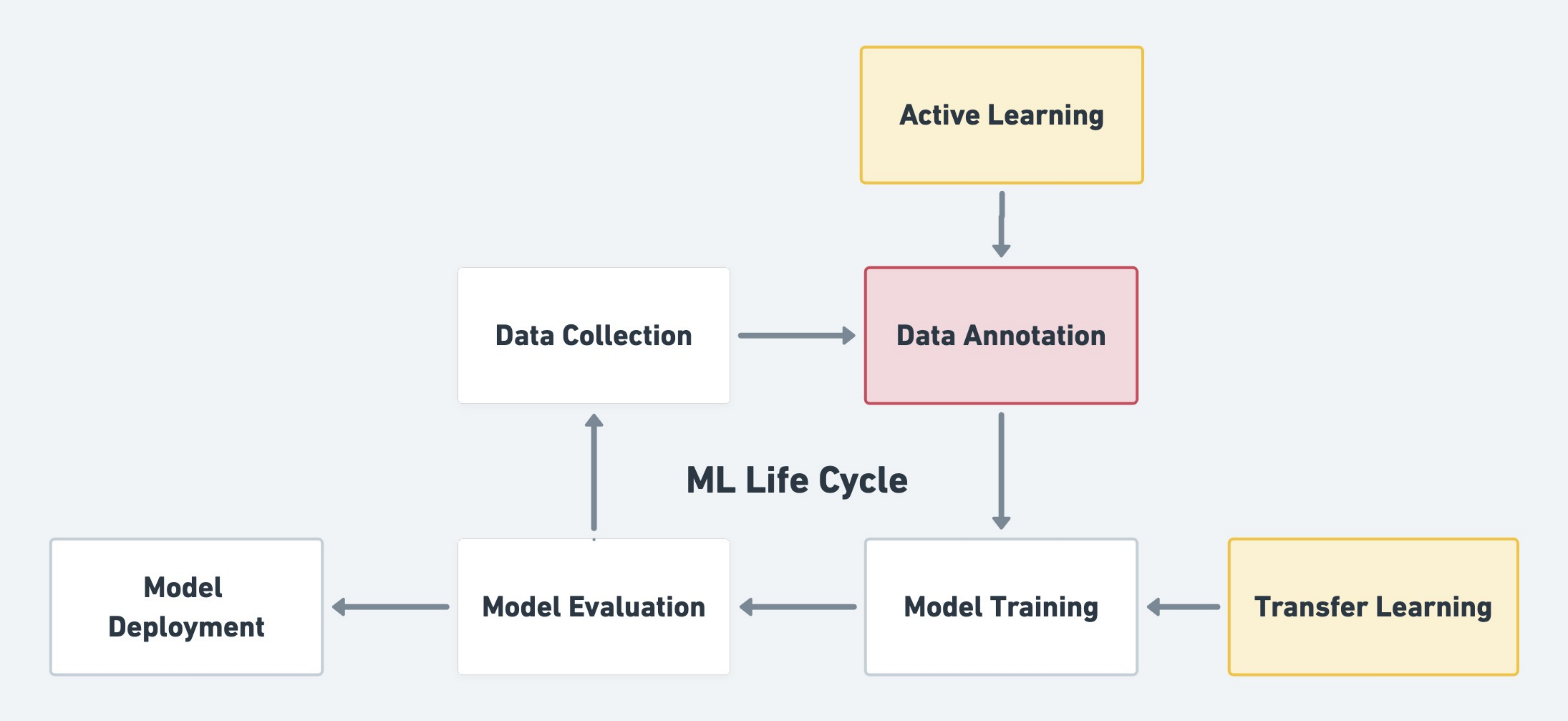

Figure 1: Machine Learning Life Cycle

Motivation

How to build a label-efficient ML System?

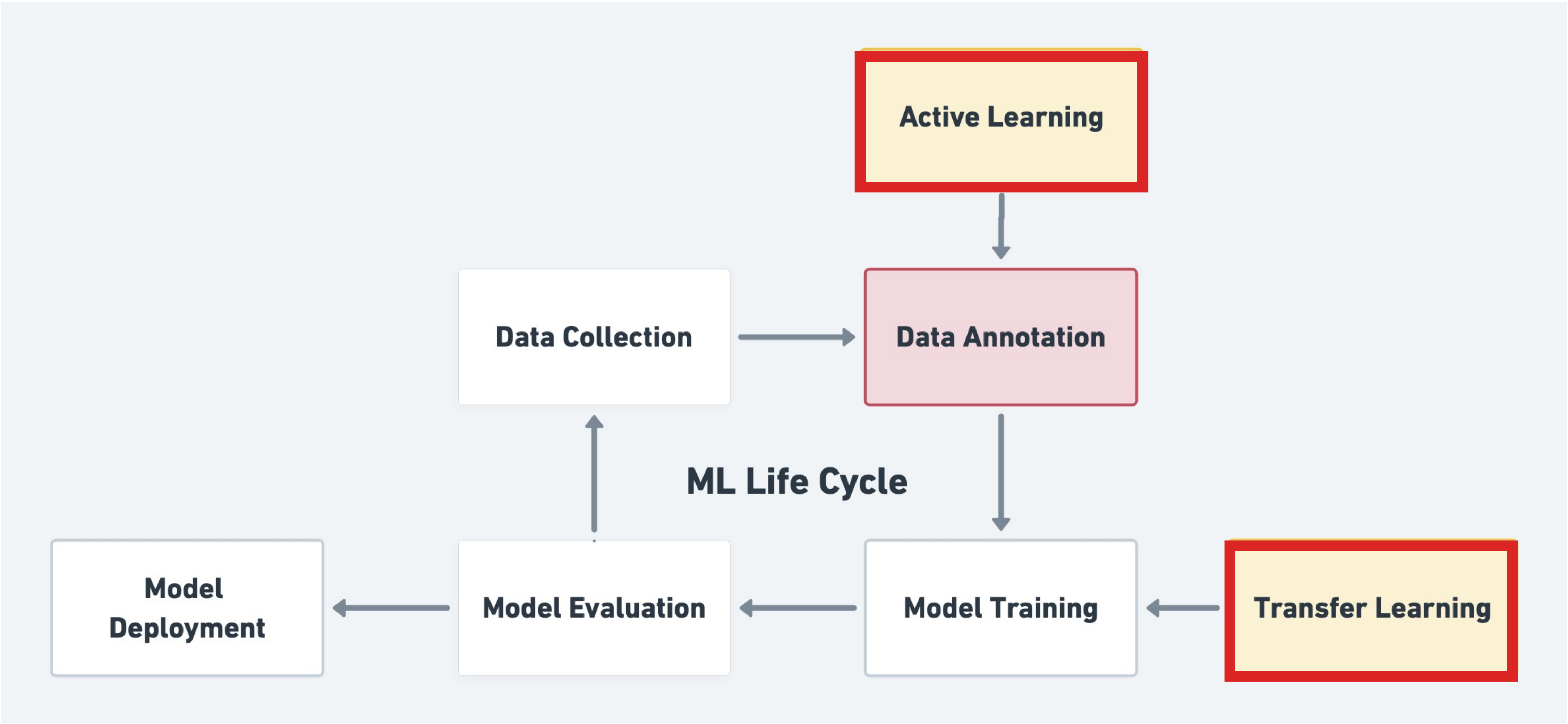

Figure 1: Machine Learning Life Cycle

Why AL + TL ?

Table 1: Comparison of Active Learning and Transfer Learning

	Active Learning (AL)	Transfer Learning (TL)
Goal	Select most informative example	Reuse knowlege learned elsewhere
Effect	Reduce number of labeled data	Reduce model training time Reduce number of labeled data
Target	Data	Model

Live Demo - DANER

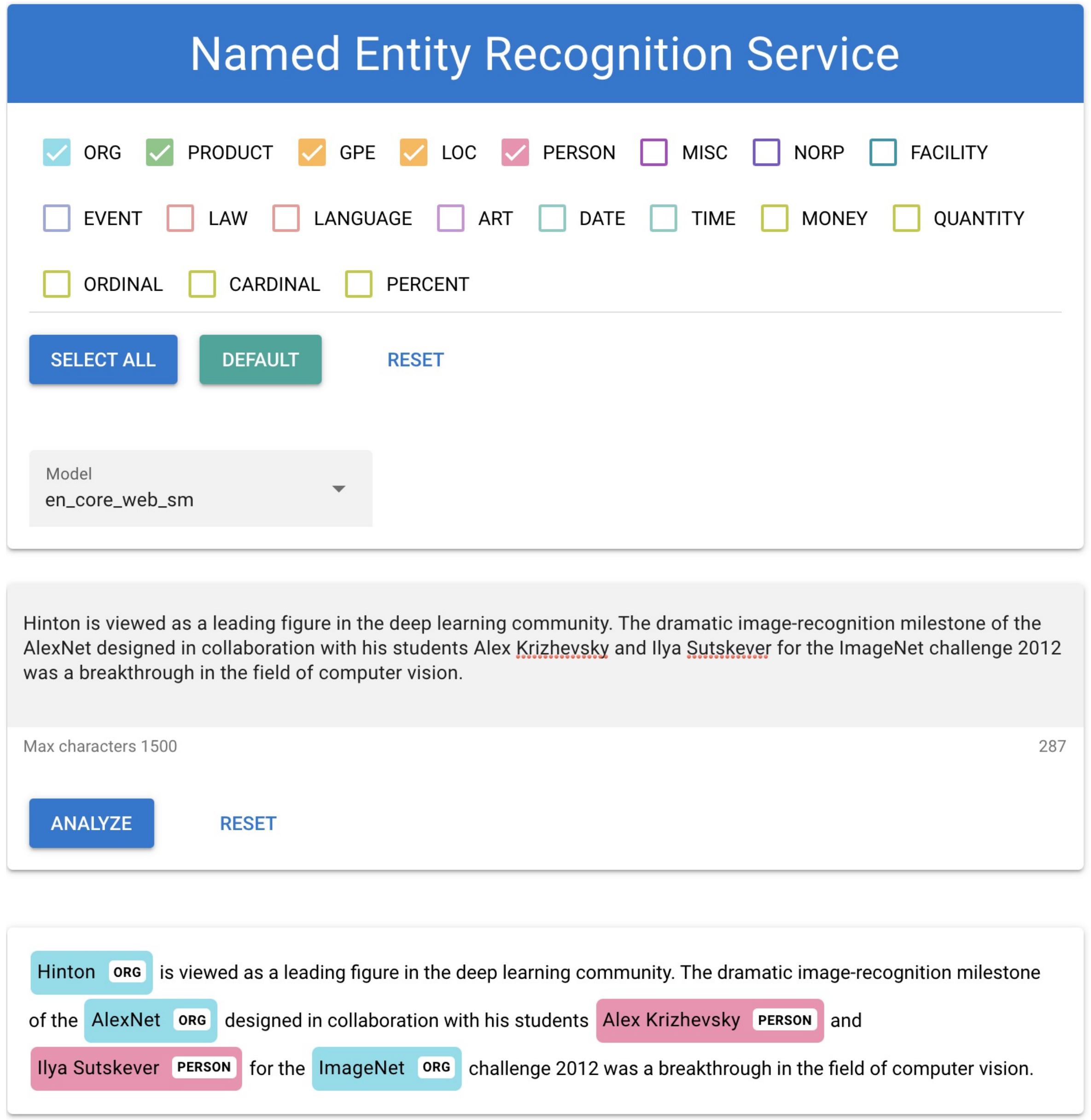

Figure 2: NER Inference Service

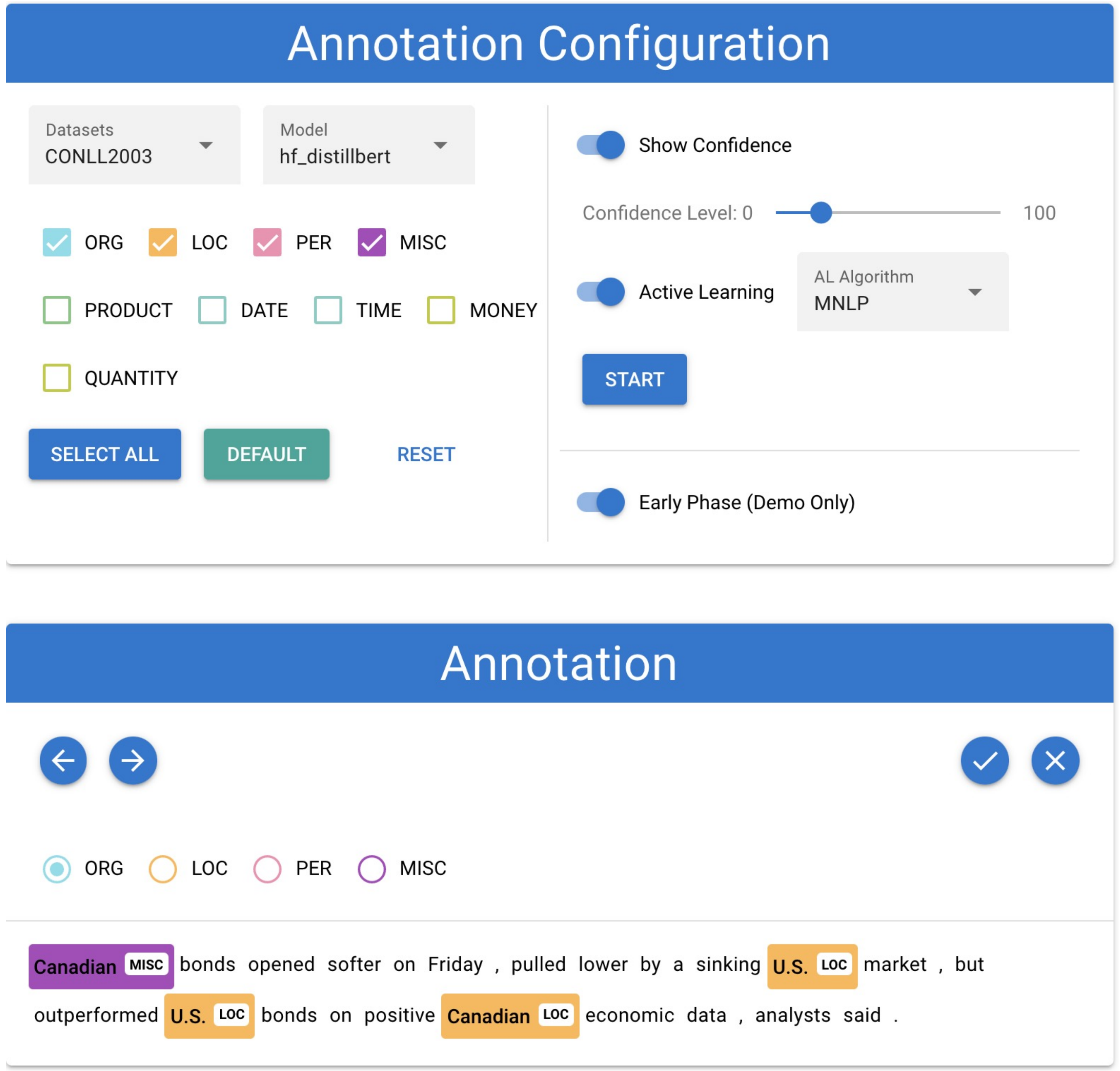

Figure 3: NER Annotation Service

How to implement?

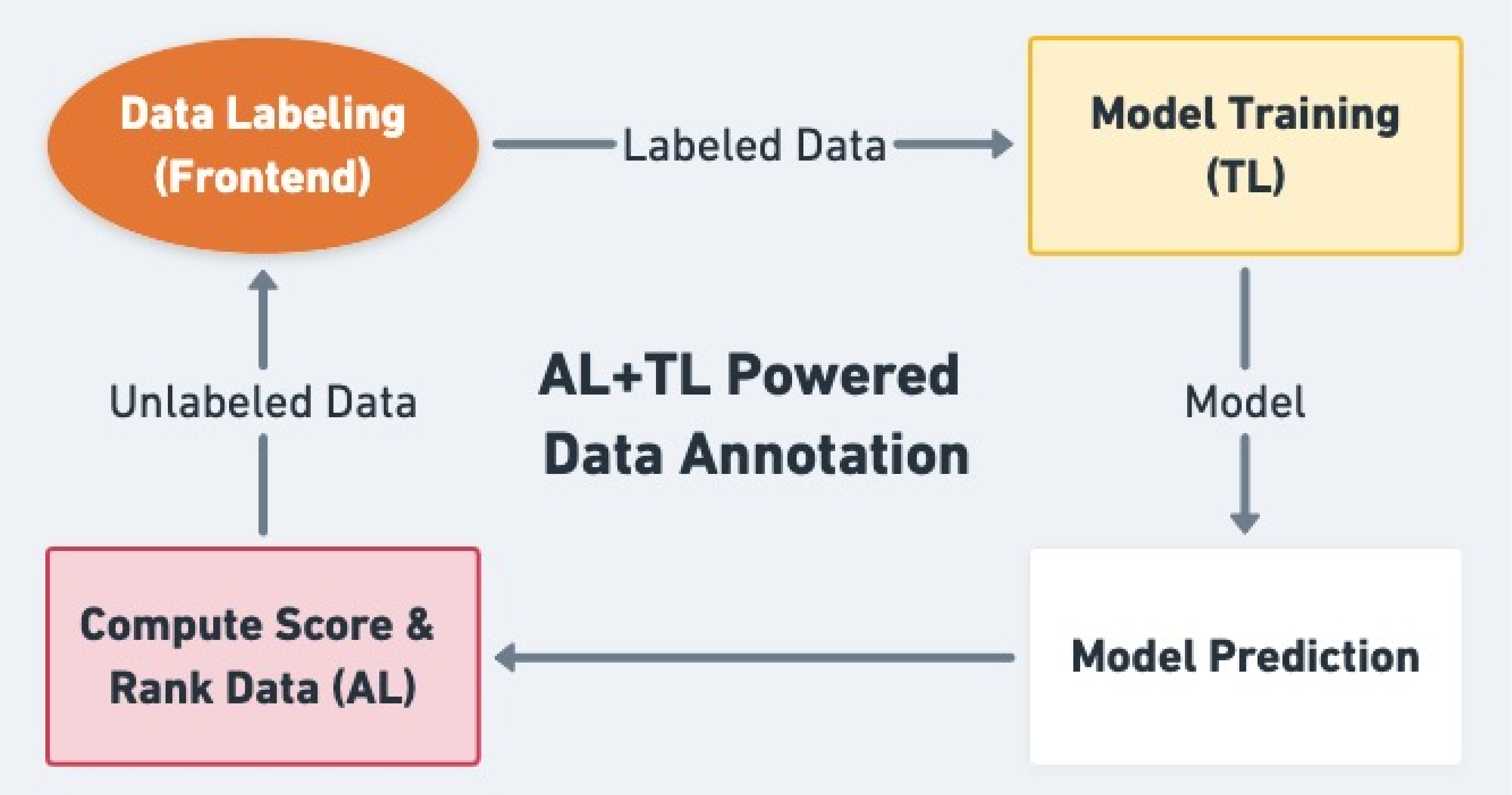

Figure 4: Data Annotation Process

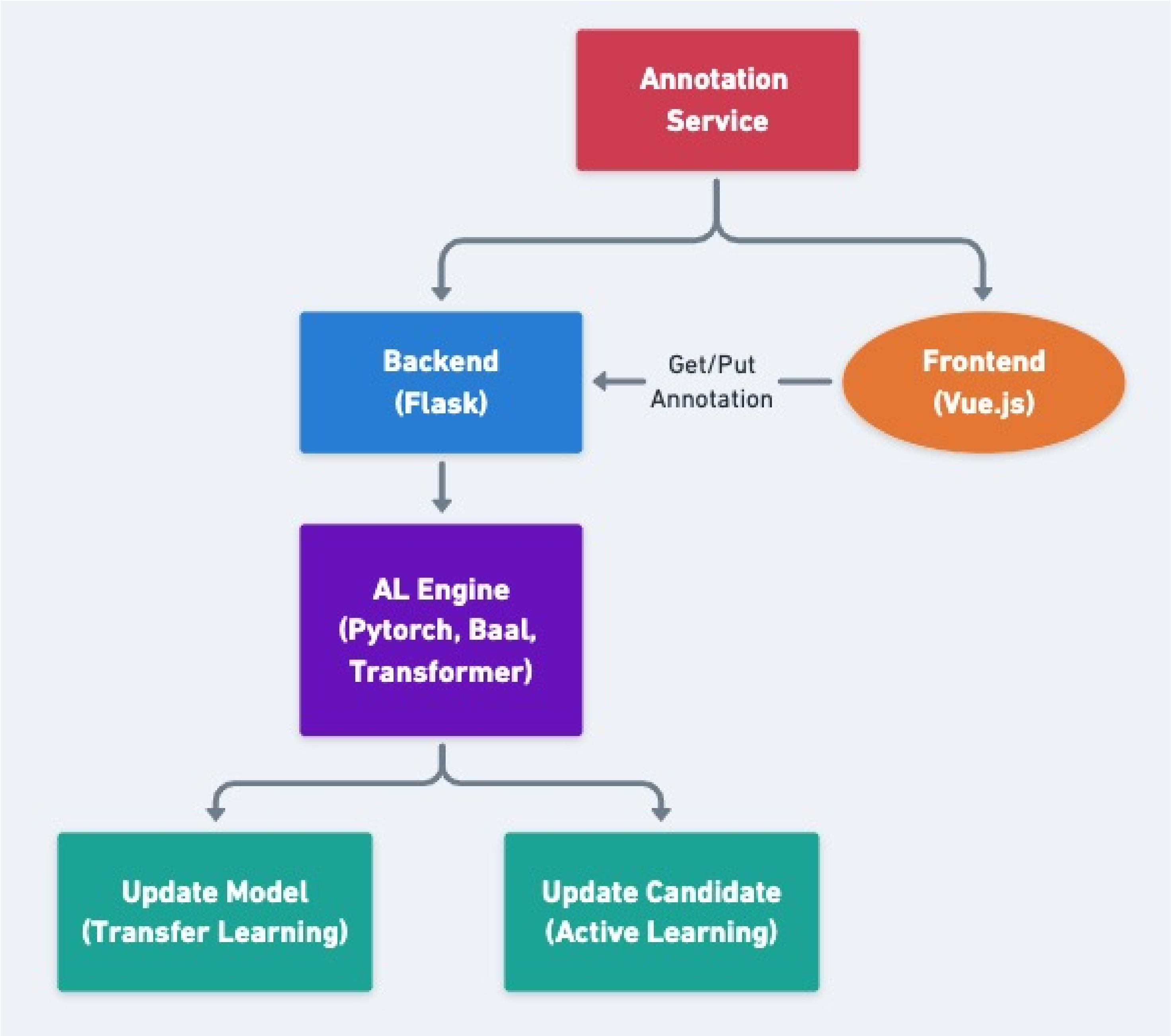

Figure 5: Annotation Service Structure Overview

How to implement?

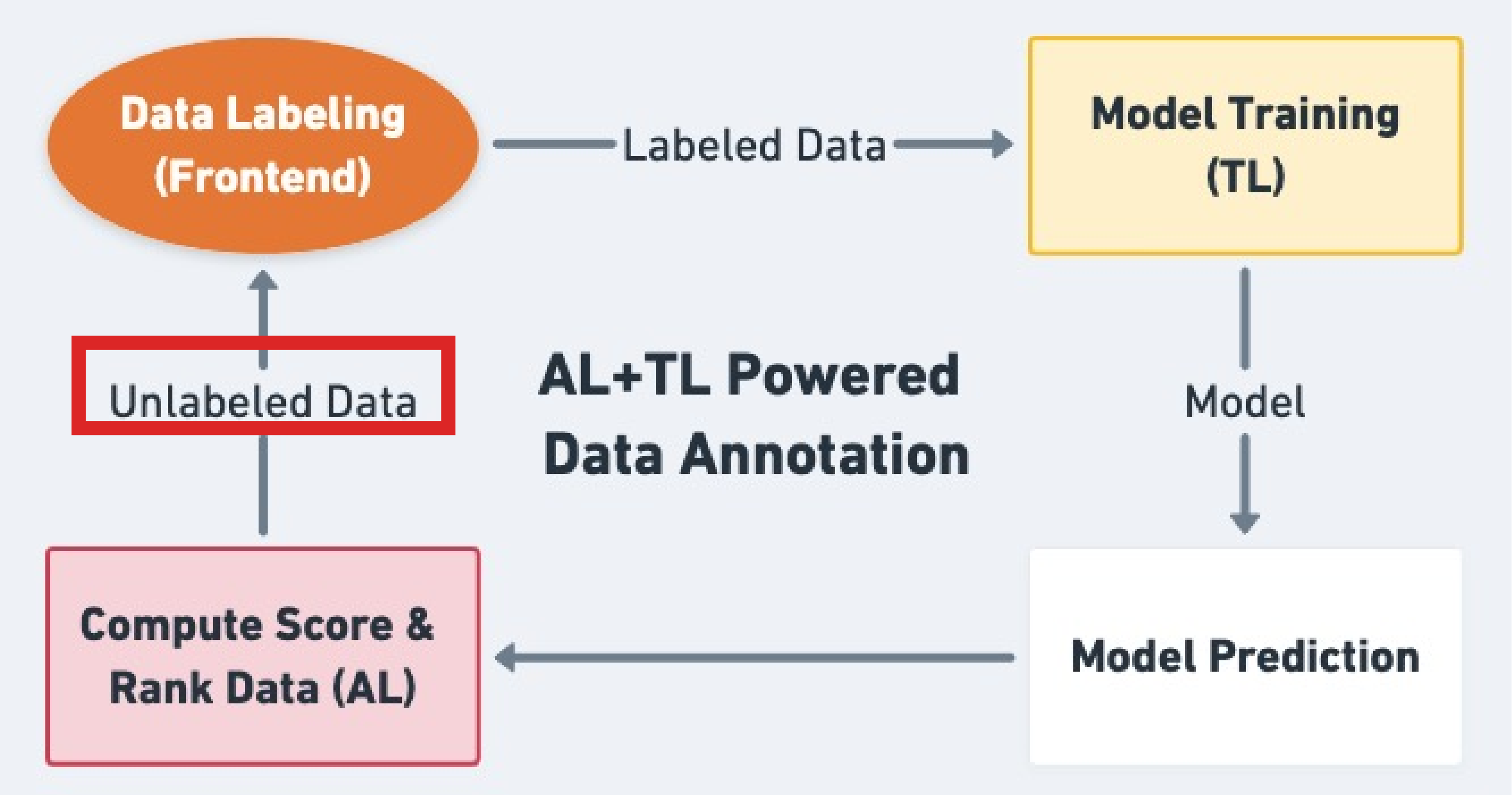

Figure 4: Data Annotation Process

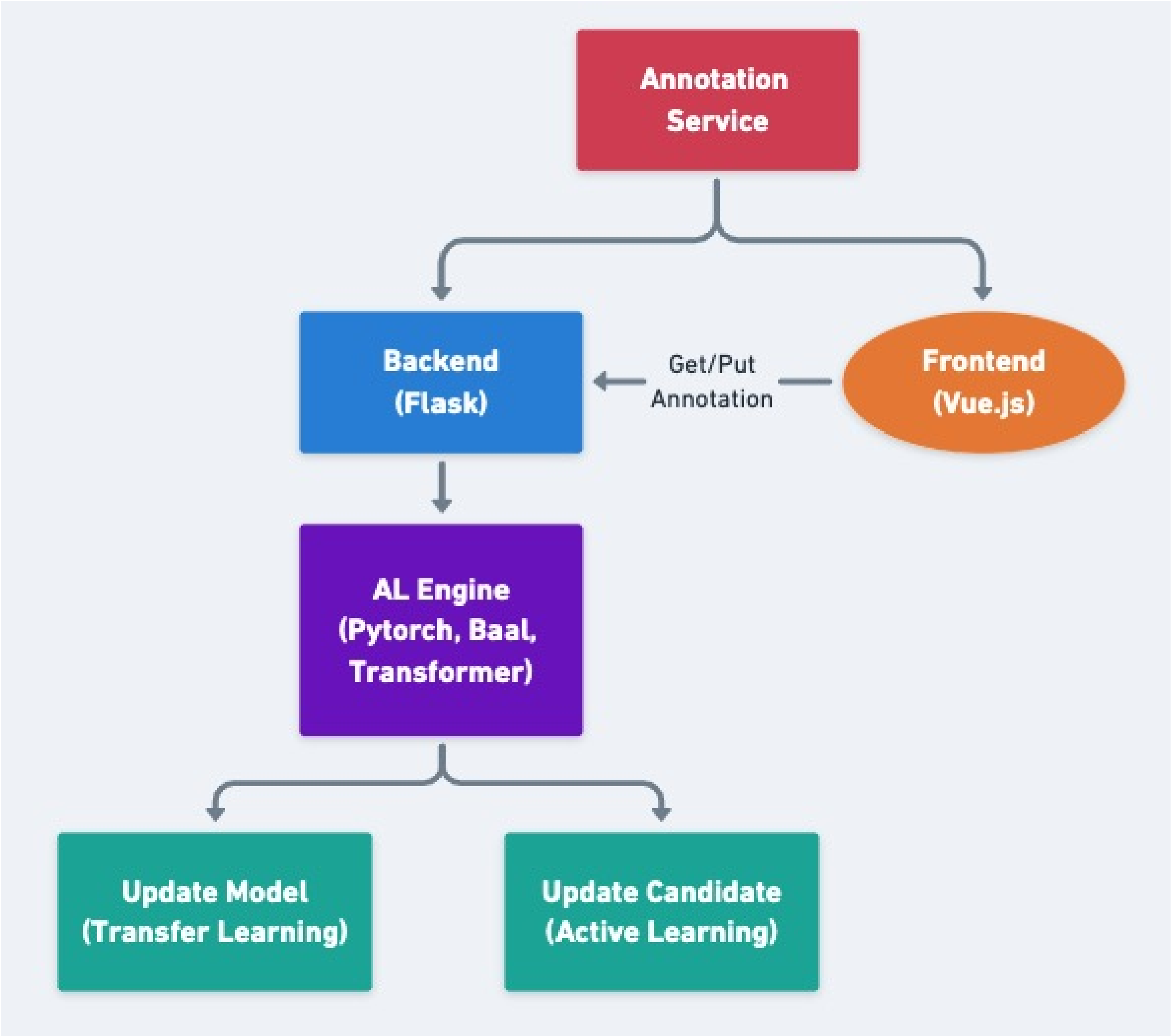

Figure 5: Annotation Service Structure Overview

How to implement?

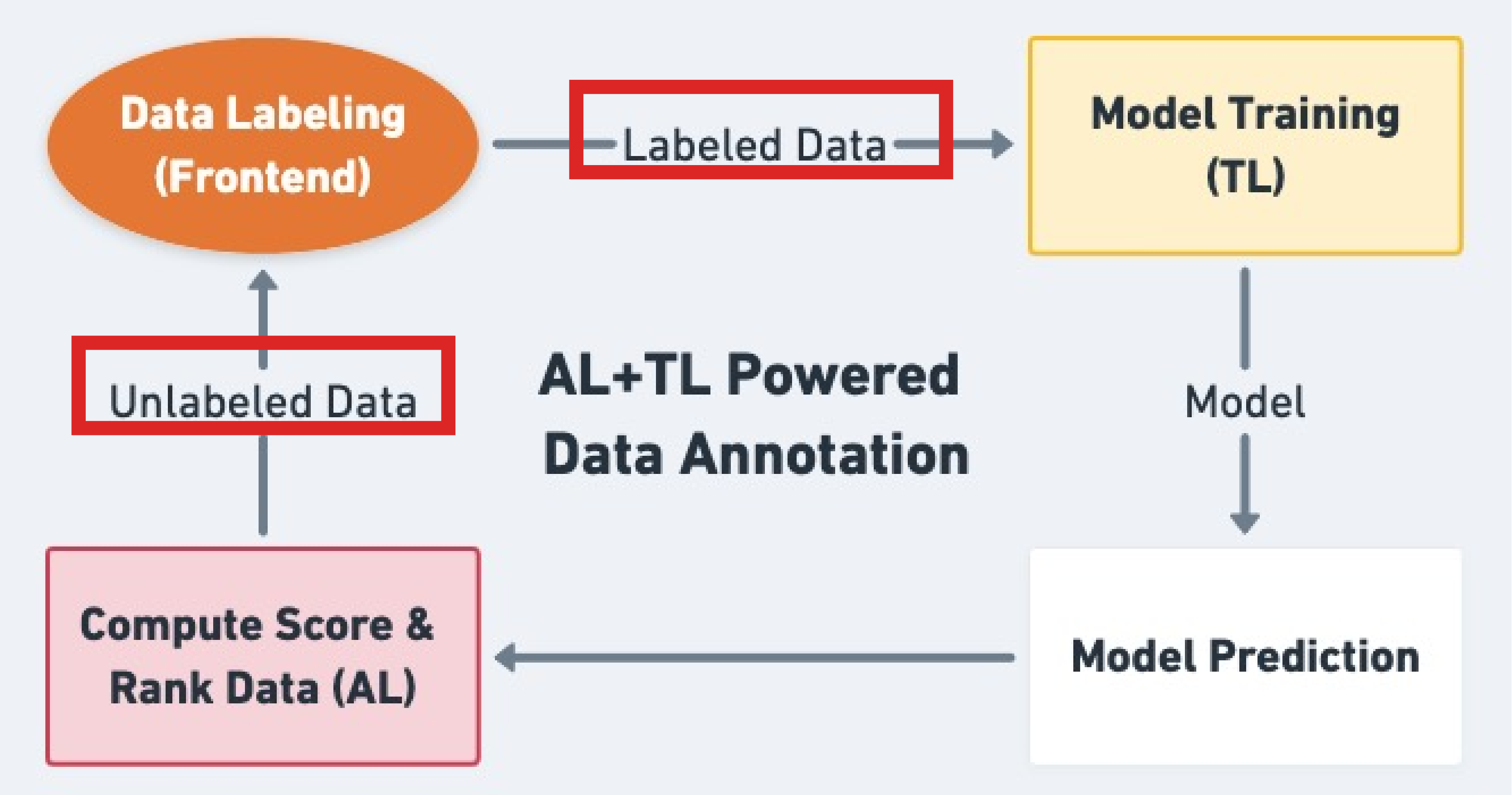

Figure 4: Data Annotation Process

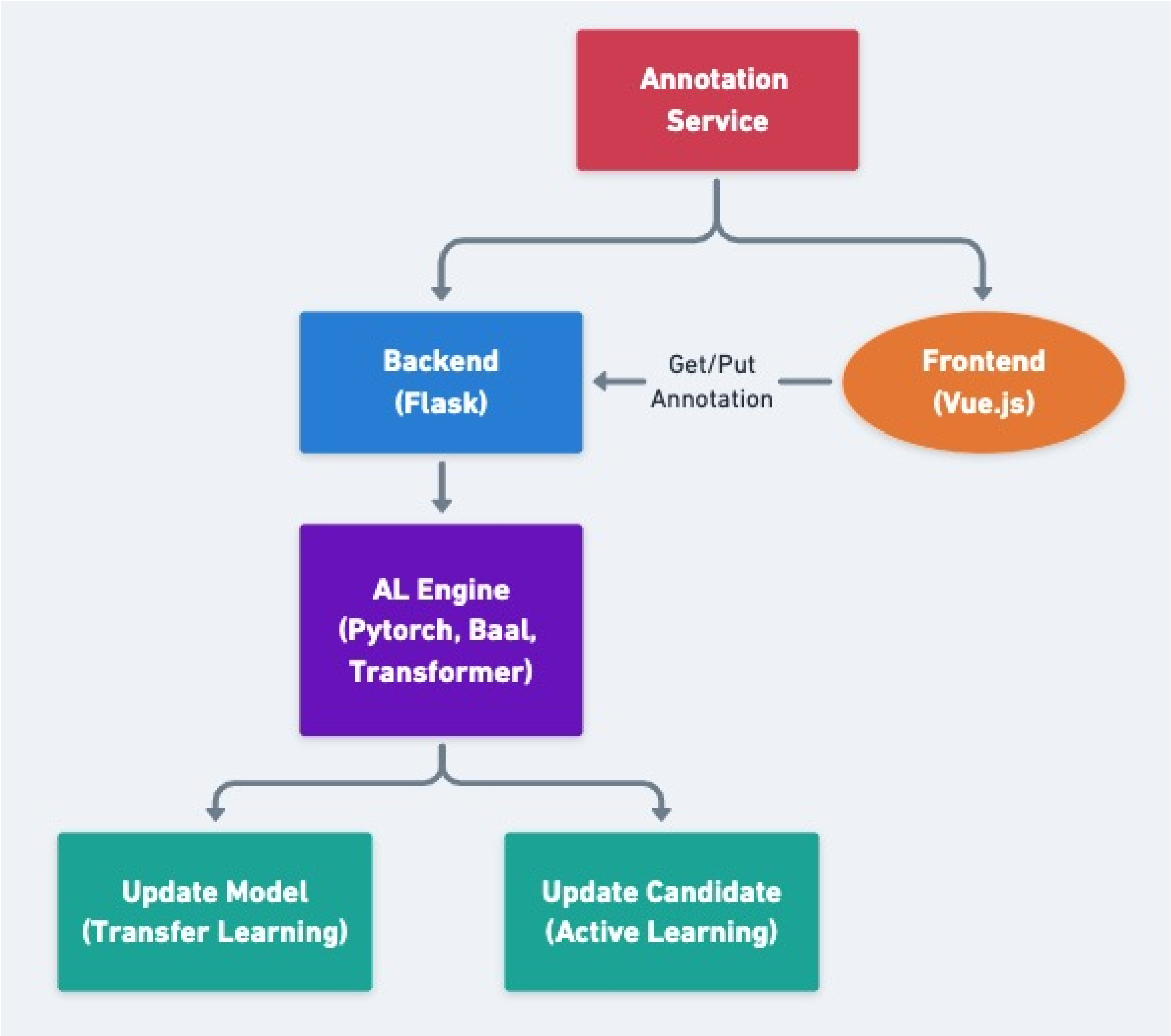

Figure 5: Annotation Service Structure Overview

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,               # total number of training steps per AL step
    per_device_train_batch_size=32,    # batch size per device during training
    per_device_eval_batch_size=64,     # batch size for evaluation
    weight_decay=0.01,                 # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                    # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,               # the tokenizer that is compatible with model
    args=training_args,                # training arguments, defined above
    train_dataset=active_set,          # AL dataset
)

trainer.train()
```

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,                # total number of training steps per AL step
    per_device_train_batch_size=32,     # batch size per device during training
    per_device_eval_batch_size=64,      # batch size for evaluation
    weight_decay=0.01,                  # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                     # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,                # the tokenizer that is compatible with model
    args=training_args,                 # training arguments, defined above
    train_dataset=active_set,           # AL dataset
)

trainer.train()
```

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,              # total number of training steps per AL step
    per_device_train_batch_size=32,   # batch size per device during training
    per_device_eval_batch_size=64,    # batch size for evaluation
    weight_decay=0.01,                # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                   # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,              # the tokenizer that is compatible with model
    args=training_args,               # training arguments, defined above
    train_dataset=active_set,         # AL dataset
)

trainer.train()
```


VECTOR INSTITUTE

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,                # total number of training steps per AL step
    per_device_train_batch_size=32,     # batch size per device during training
    per_device_eval_batch_size=64,      # batch size for evaluation
    weight_decay=0.01,                  # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                     # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,                # the tokenizer that is compatible with model
    args=training_args,                 # training arguments, defined above
    train_dataset=active_set,           # AL dataset
)

trainer.train()
```

Update Model - Transfer Learning

```python
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,                # total number of training steps per AL step
    per_device_train_batch_size=32,     # batch size per device during training
    per_device_eval_batch_size=64,      # batch size for evaluation
    weight_decay=0.01,                  # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                     # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,                # the tokenizer that is compatible with model
    args=training_args,                 # training arguments, defined above
    train_dataset=active_set,           # AL dataset
)

trainer.train()
```

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,               # total number of training steps per AL step
    per_device_train_batch_size=32,    # batch size per device during training
    per_device_eval_batch_size=64,     # batch size for evaluation
    weight_decay=0.01,                 # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                    # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,               # the tokenizer that is compatible with model
    args=training_args,                # training arguments, defined above
    train_dataset=active_set,          # AL dataset
)

trainer.train()
```


VECTOR INSTITUTE

Update Model - Transfer Learning

```
from transformers import AutoTokenizer, AutoModelForTokenClassification, Trainer, TrainingArguments
training_args = TrainingArguments(
    max_steps=MAX_STEPS,              # total number of training steps per AL step
    per_device_train_batch_size=32,   # batch size per device during training
    per_device_eval_batch_size=64,    # batch size for evaluation
    weight_decay=0.01,                # strength of weight decay
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
hf_model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_list))

trainer = Trainer(
    model=hf_model,                   # the instantiated 🤗 Transformers model to be trained
    tokenizer=tokenizer,              # the tokenizer that is compatible with model
    args=training_args,               # training arguments, defined above
    train_dataset=active_set,         # AL dataset
)

trainer.train()
```

Update Candidate - Active Learning

- Maximum Normalized Log-Probability (MNLP)
  - Intuition: select the most uncertain points.
  - How to represent the uncertainty?
    - Let $\mathbf{x}_i$ be the i^{th} data point with n tokens.
    - The probability of model current prediction: $\max_{y_1,...,y_n} \mathbb{P}\left[y_1, \ldots, y_n \mid \mathbf{x}_i\right]$
    - The normalized log probability of model current prediction on $\mathbf{x}_i$ is computed as,

$$MNLP(\mathbf{x}_i) = \max_{y*1,...,y*n} \frac{1}{n} \sum *j = 1^n \log \mathbb{P}\left[y * j \mid \mathbf{x\_}i\right]$$

  - AL Data Candidate $\{\mathbf{x}\} = \arg\min_{\mathbf{x}_i} MNLP(\mathbf{x}_i)$

VECTOR INSTITUTE

Update Candidate - Active Learning

- Maximum Normalized Log-Probability (MNLP)
  - Intuition: select the most uncertain points.
  - How to represent the uncertainty?
    - Let $\mathbf{x}_i$ be the i^{th} data point with n tokens.
    - The probability of model current prediction: $\boxed{\max_{y_1,...,y_n} \mathbb{P}\left[y_1, \ldots, y_n \mid \mathbf{x}_i\right]}$
    - The normalized log probability of model current prediction on $\mathbf{x}_i$ is computed as,

$$MNLP(\mathbf{x}_i) = \max_{y*1,...,y*n} \frac{1}{n} \sum *j = 1^n \log \mathbb{P}\left[y*j \mid \mathbf{x}\_i\right]$$

  - AL Data Candidate $\{\mathbf{x}\} = \arg\min_{\mathbf{x}_i} MNLP(\mathbf{x}_i)$

Update Candidate - Active Learning

- Maximum Normalized Log-Probability (MNLP)
  - Intuition: select the most uncertain points.
  - How to represent the uncertainty?
    - Let $\mathbf{x}_i$ be the i^{th} data point with n tokens.
    - The probability of model current prediction: $\boxed{\max_{y_1,...,y_n} \mathbb{P}\left[y_1, \ldots, y_n \mid \mathbf{x}_i\right]}$
    - The normalized log probability of model current prediction on $\mathbf{x}_i$ is computed as,

$$\boxed{MNLP(\mathbf{x}_i) = \max_{y*1,...,y*n} \frac{1}{n} \sum *j = 1^n \log \mathbb{P}\left[y*j \mid \mathbf{x}\_i\right]}$$

  - AL Data Candidate $\{\mathbf{x}\} = \arg\min_{\mathbf{x}_i} MNLP(\mathbf{x}_i)$

Update Candidate - Active Learning

- Maximum Normalized Log-Probability (MNLP)
  - Intuition: select the most uncertain points.
  - How to represent the uncertainty?
    - Let $\mathbf{x}_i$ be the i^{th} data point with n tokens.
    - The probability of model current prediction: $\boxed{\max_{y_1,...,y_n} \mathbb{P}\left[y_1, \ldots, y_n \mid \mathbf{x}_i\right]}$
    - The normalized log probability of model current prediction on $\mathbf{x}_i$ is computed as,

$$\boxed{MNLP(\mathbf{x}_i) = \max_{y*1,...,y*n} \frac{1}{n} \sum *j = 1^n \log \mathbb{P}\left[y*j \mid \mathbf{x\_}i\right]}$$

  - AL Data Candidate $\{\mathbf{x}\} = \arg\min_{\mathbf{x}_i} MNLP(\mathbf{x}_i)$

Select data with small MNLP score!

Small score = Low Confidence = High Uncertainty

Limitation and Next Step

- Active Learning Performance Gap
  - Dataset? Architecture?
  - Transfer Learning?
  - More robust AL Algorithm is needed
- Noisy Human Label
  - Human label may not be reliable

User 1

Geoffrey Everest Hinton PERSON (born 6 CARDINAL December 1947 DATE) is a British NORP - Canadian cognitive psychologist and computer scientist , most noted for his work on artificial neural networks .

User 2

Geoffrey Everest Hinton (born 6 December 1947 DATE) is a British NORP - Canadian cognitive psychologist and computer scientist , most noted for his work on artificial neural networks .

  - Data Quality Control

Limitation and Next Step

- Active Learning Performance Gap
  - Dataset? Architecture?
  - Transfer Learning?
  - More robust AL Algorithm is needed
- Noisy Human Label
  - Human label may not be reliable

User 1

User 2

  - Data Quality Control

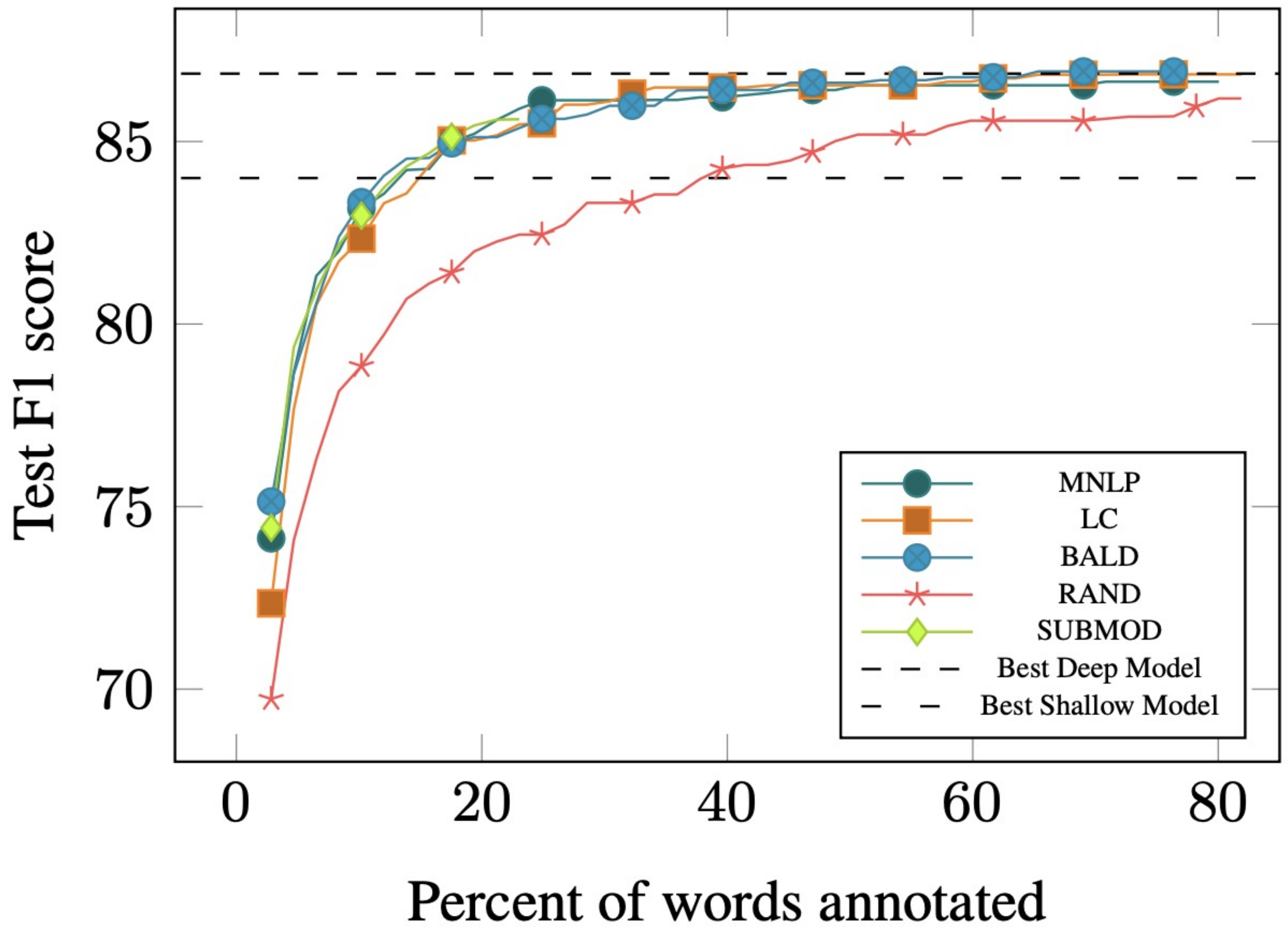

Figure 6: AL Performance from Literature [1]

[1] Deep Active Learning for Named Entity Recognition (ICLR2018)

VECTOR INSTITUTE

Take away

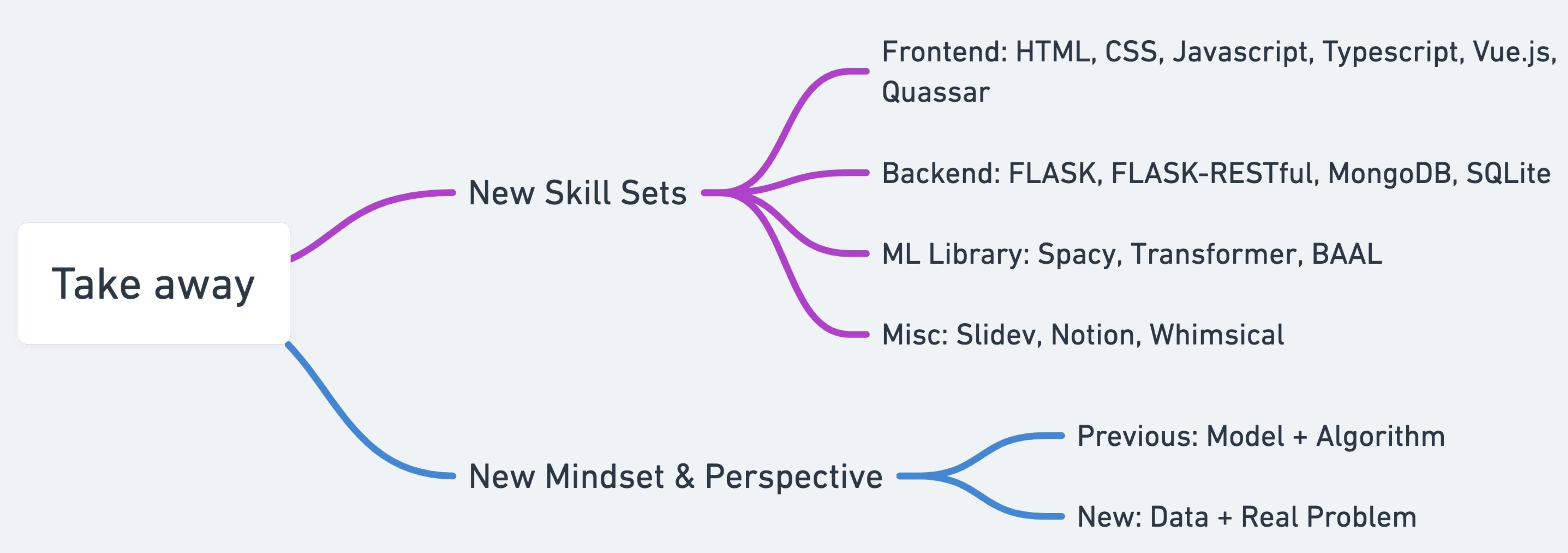

Figure 7: Internship Take Away