VINBIGDATA

Real World Natural Language Processing Applications

QuangPH3
Language Representations Teams - NLP Research of VinBigdata

Agenda

1. Sơ lược về NLP
2. Machine Learning Workflows
3. Case Study

Agenda

1 Sơ lược về NLP

Nghiên cứu và học tập trong lĩnh vực NLP

Kỹ thuật lập trình	Math, Statistics, and Probability	Text Preprocessing	Machine Learning Basics	NLP Core Techniques
- Python hoặc Java - Kỹ thuật lập trình - Cấu trúc dữ liệu và giải thuật	- Kiến thức về toán, thống kê, xác suất	- Hiểu được các kiểu biểu diễn ngôn ngữ văn bản, cách chuyển đổi từ ngôn ngữ tự nhiên sang dạng dữ liệu máy có thể hiểu - Kiến thức về ngôn ngữ học - Cách tiền xử lý, chuẩn hóa dữ liệu	- Các thuật toán cơ bản của học máy: Linear Regression, Logistic Regression, Decision Tree, SVM, Neural Network,... - Các mô hình học sâu (Deeplearning): CNN, LSTM, GRU, Transformers,... - Các framework: sklearn, tensorflow, pytorch,...	- Hiểu được các bài toán của NLP, có khả năng sử dụng các kỹ thuật lập trình để giải quyết các bài toán của NLP - Thiết kế và xây dựng các hệ thống sử dụng NLP

Nghiên cứu và học tập trong lĩnh vực NLP

Low Level

- Text Cleaning
- Text Normalization
- Word Segmentation
- Part-of-Speech Tagging
- Chunking
- Parsing
- Text Representation
- Spelling Correction
- Semantic Role Labeling

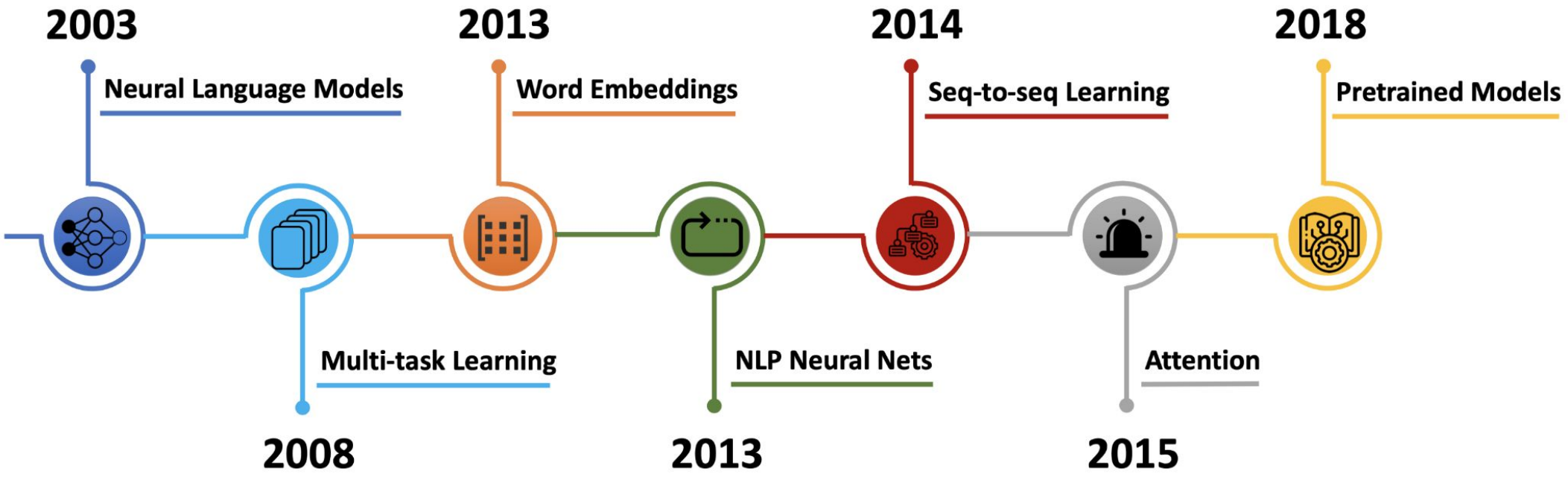
High Level

- Text Classification
- Sentiment Analysis
- Named Entity Recognition
- Language Detection
- Relationship extraction
- Event Extraction
- Keyword Extraction
- Information retrieval and extraction
- Coreference Resolution
- Entity Linking

Application

- Machine Translation
- Automatic Summarization
- Natural Language Generation
- Question Answering system
- Chatbot/VA
- Dialog Systems
- Knowledge Base Systems

AI in Industry: Xu hướng nghiên cứu



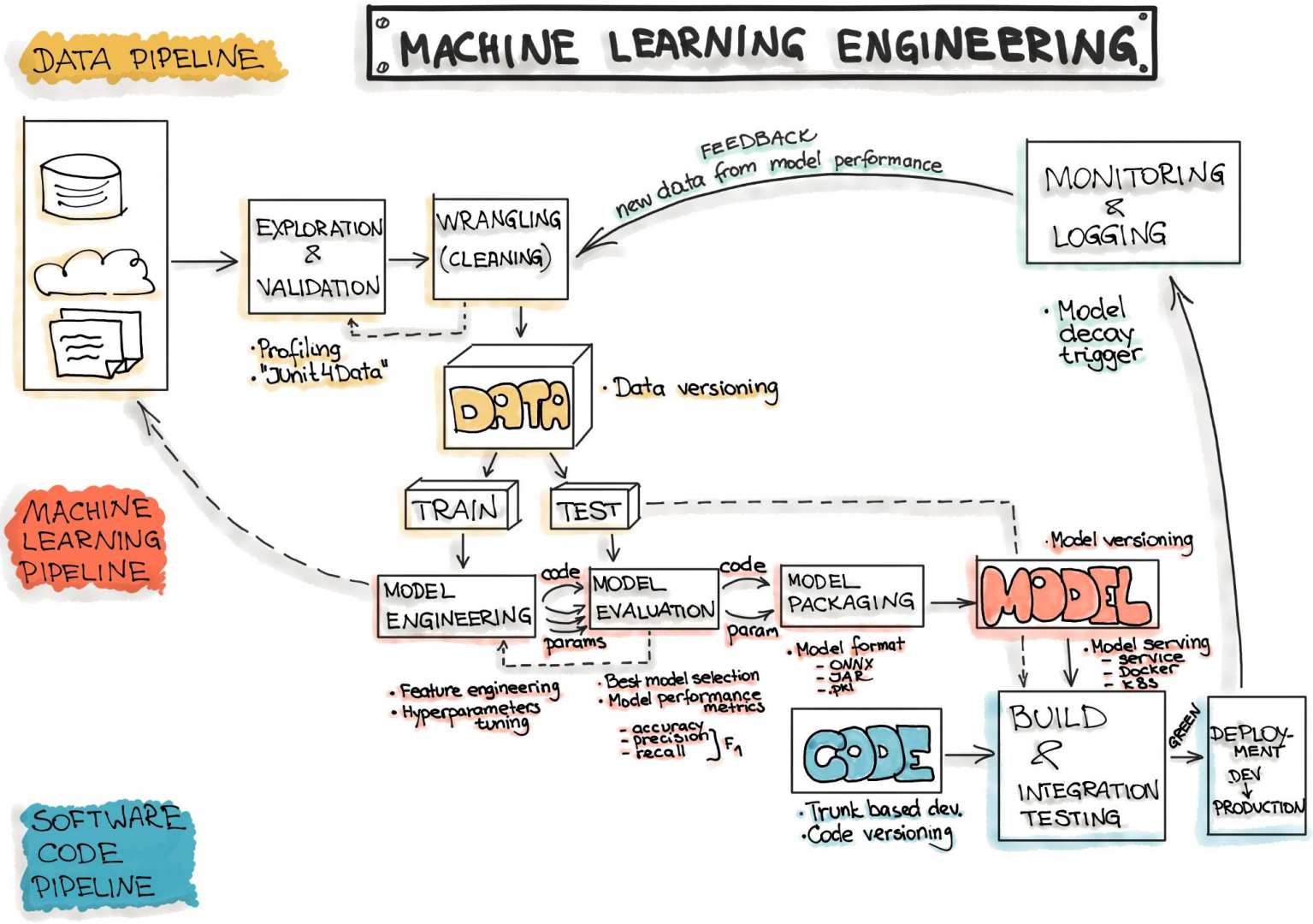
AI in Industry: Xu hướng nghiên cứu

01	Self-Supervised Learning and Transfer Learning	• Representation Learning • Large Language Models • Multilingual/Cross-lingual Language Modeling • Low-resource language / Domain Specific language
02	Multi-Modality	• Multimodal • Modality agnostic models • AI and the Arts, Generative models
03	Embodied learning	• Reinforcement learning • Learn from interaction
04	AIoT	• Artificial Intelligence of Things • Model Optimization

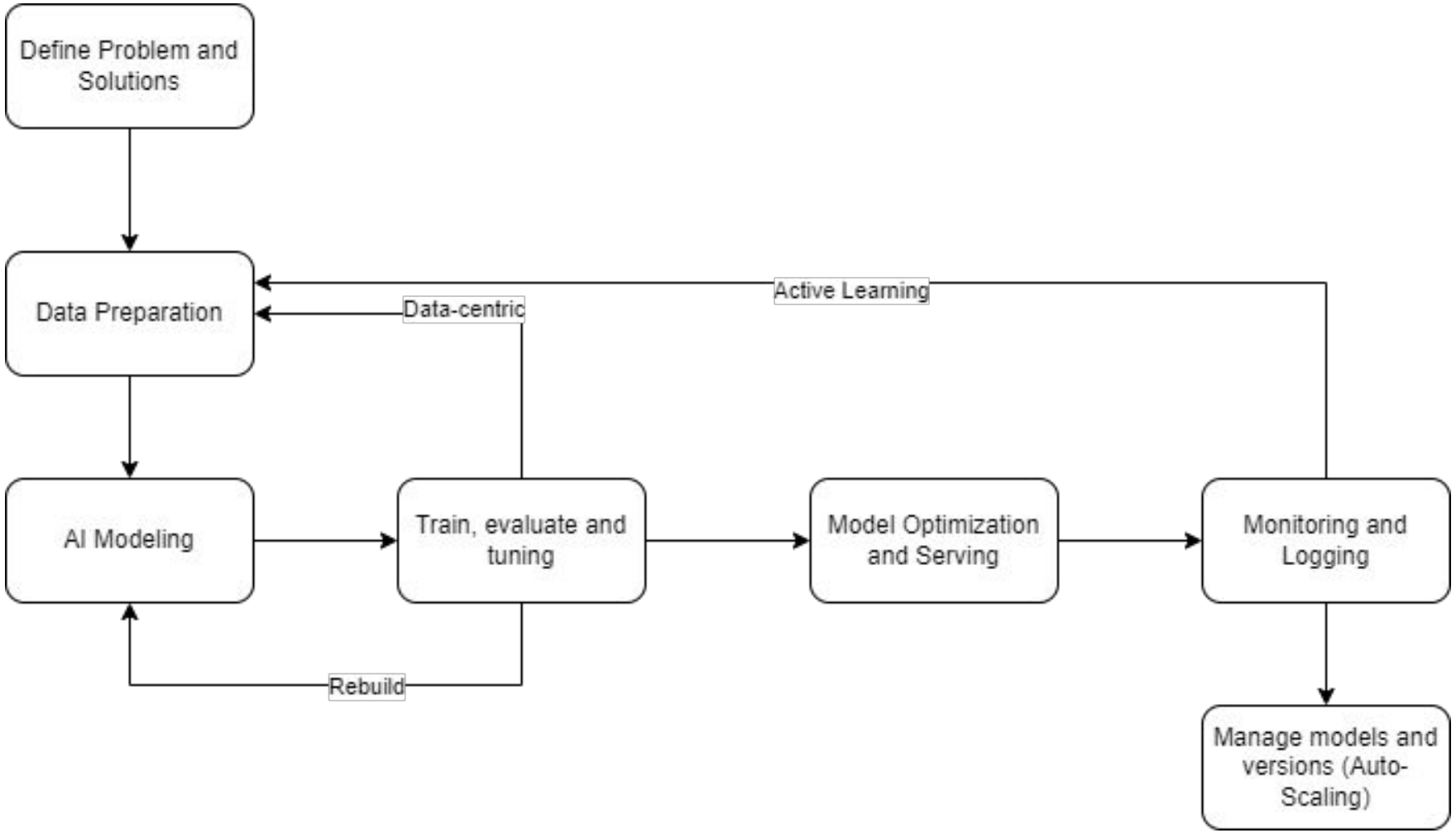
Agenda

1. Sơ lược về NLP
2. Machine Learning Workflows

Machine Learning Workflows



Machine Learning Workflows



Machine Learning Workflows: Define Problem and Solutions

Data

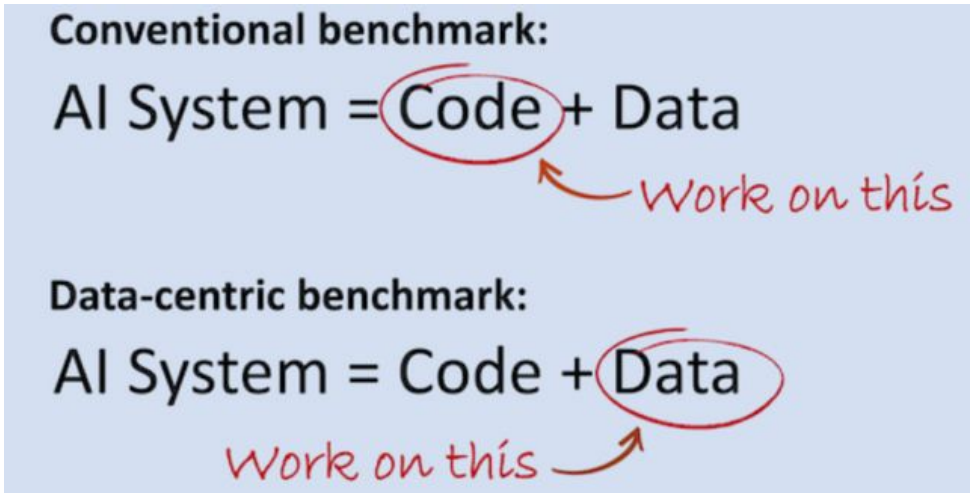
+ Nguồn dữ liệu
+ Thu thập
+ Gán nhãn
+ Tiền xử lý,...

AI Algorithms

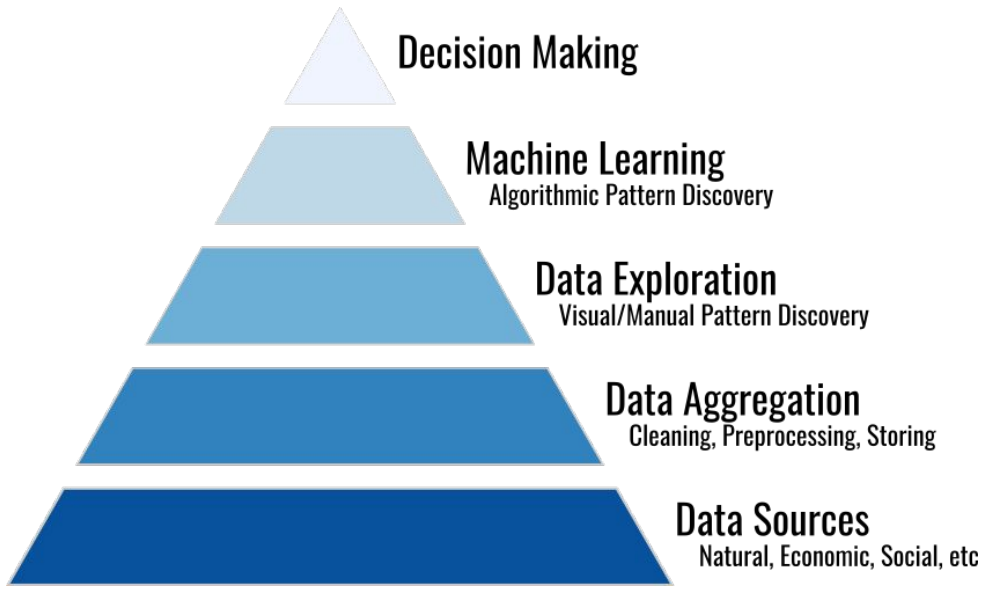
+ Mô hình hóa (đầu vào, đầu ra, encoder, decoder)
+ Hàm mục tiêu, hàm tối ưu

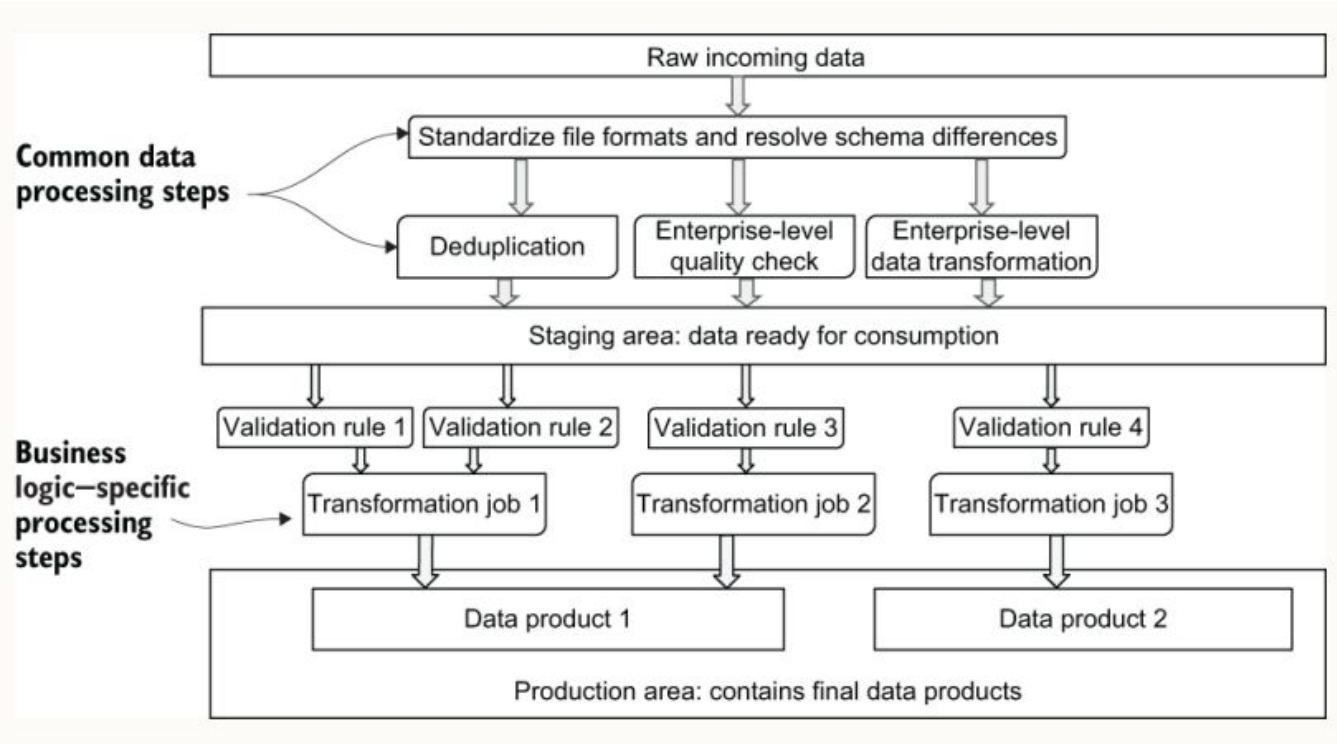
AI Model

+ Huấn luyện mô hình
+ Tối ưu siêu tham số
+ Serving và giám sát



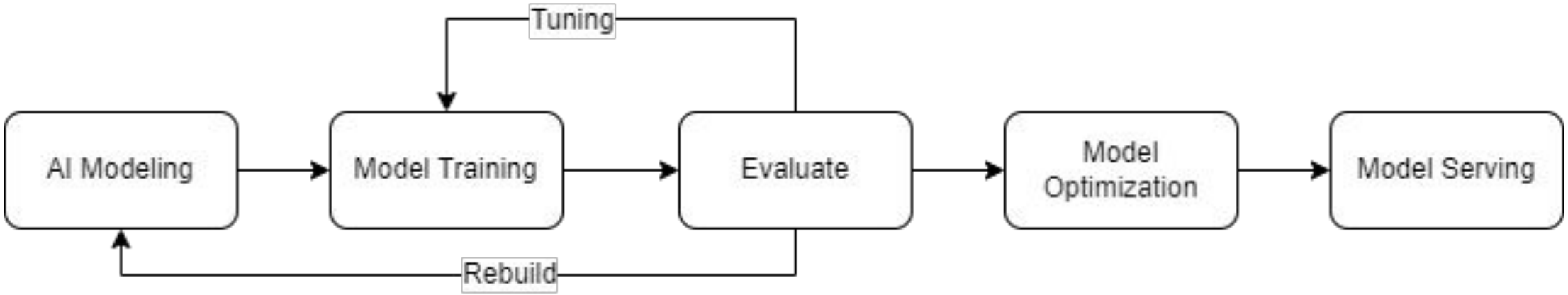
Machine Learning Workflows: Data Engineer



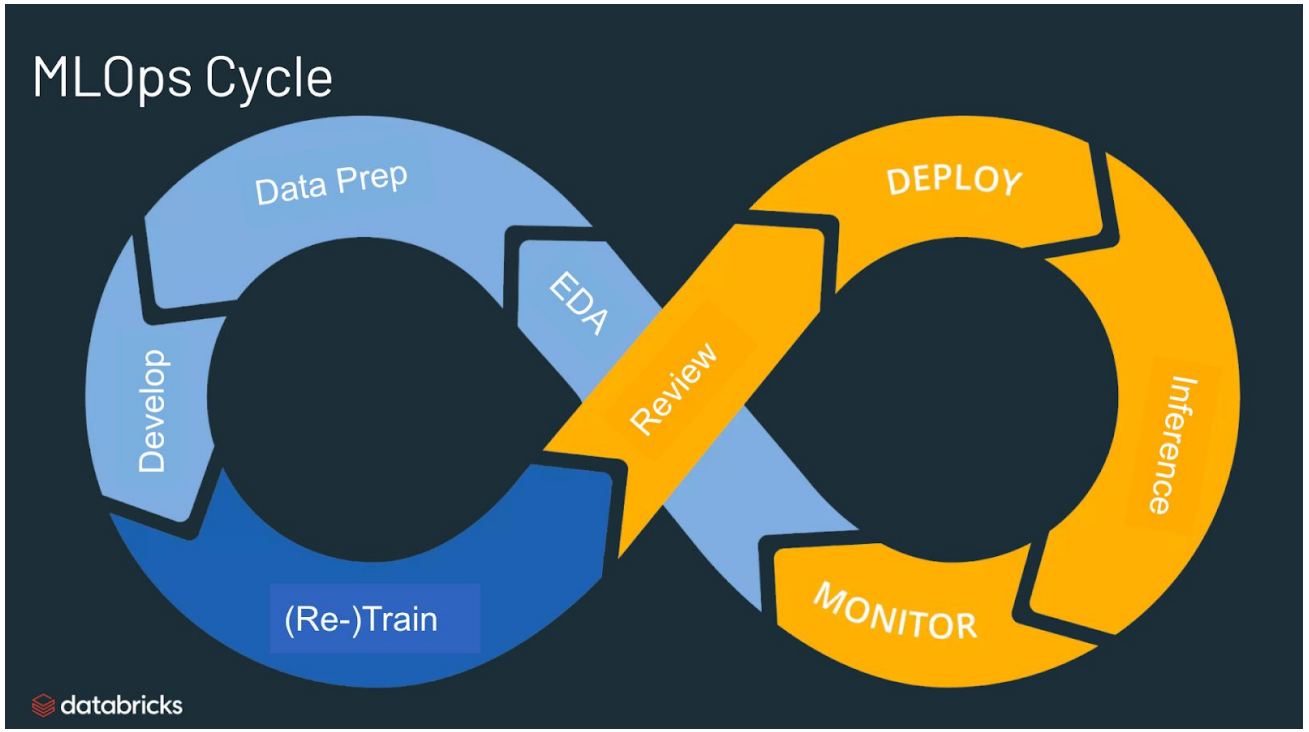


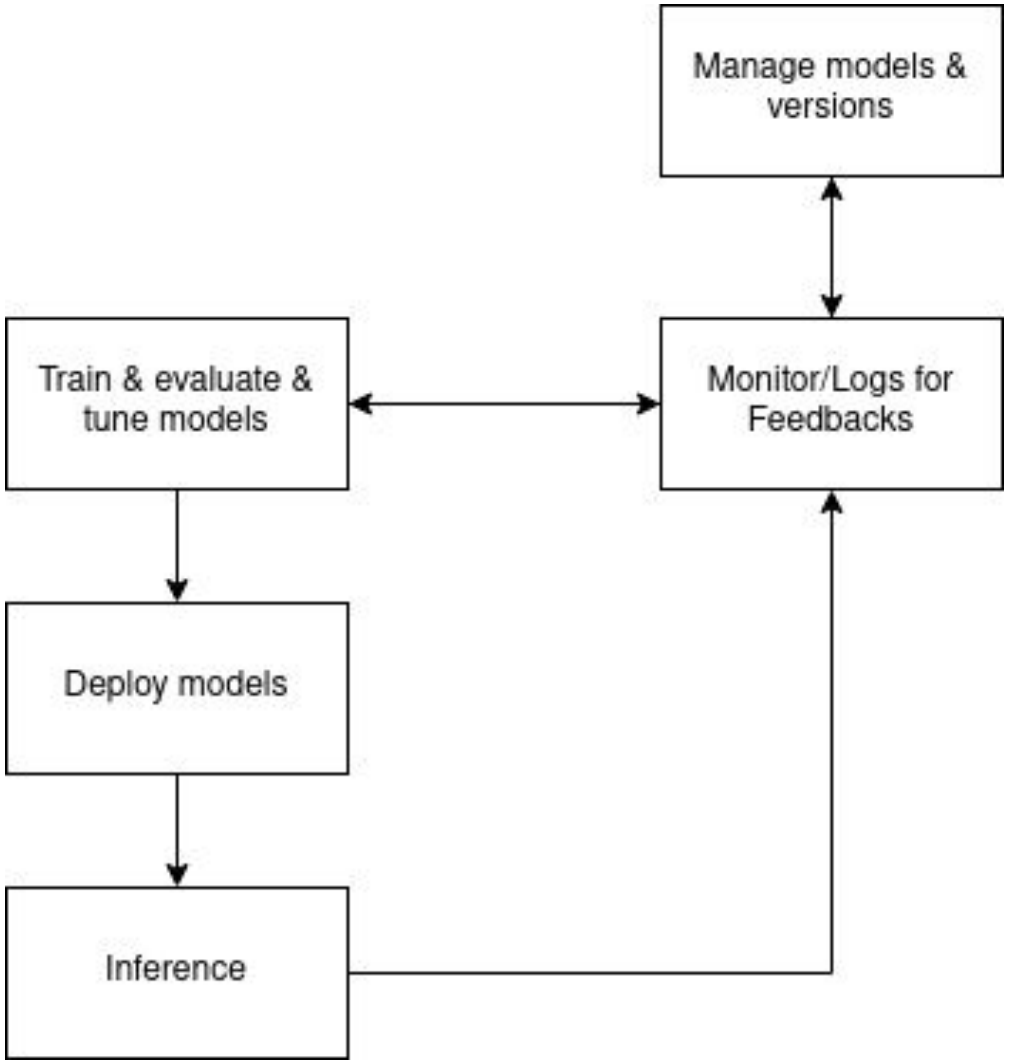
- Đảm nhận các công việc liên quan tới thu gom, tổ chức, quản lý, xử lý dữ liệu để tìm các insight giúp người dùng đưa ra các quyết định tốt hơn trên nhiều phương diện liên quan tới sản phẩm và công việc kinh doanh.
- Chuẩn bị data trên 1 database thuộc Serving Layer

Machine Learning Workflows: AI Engineer

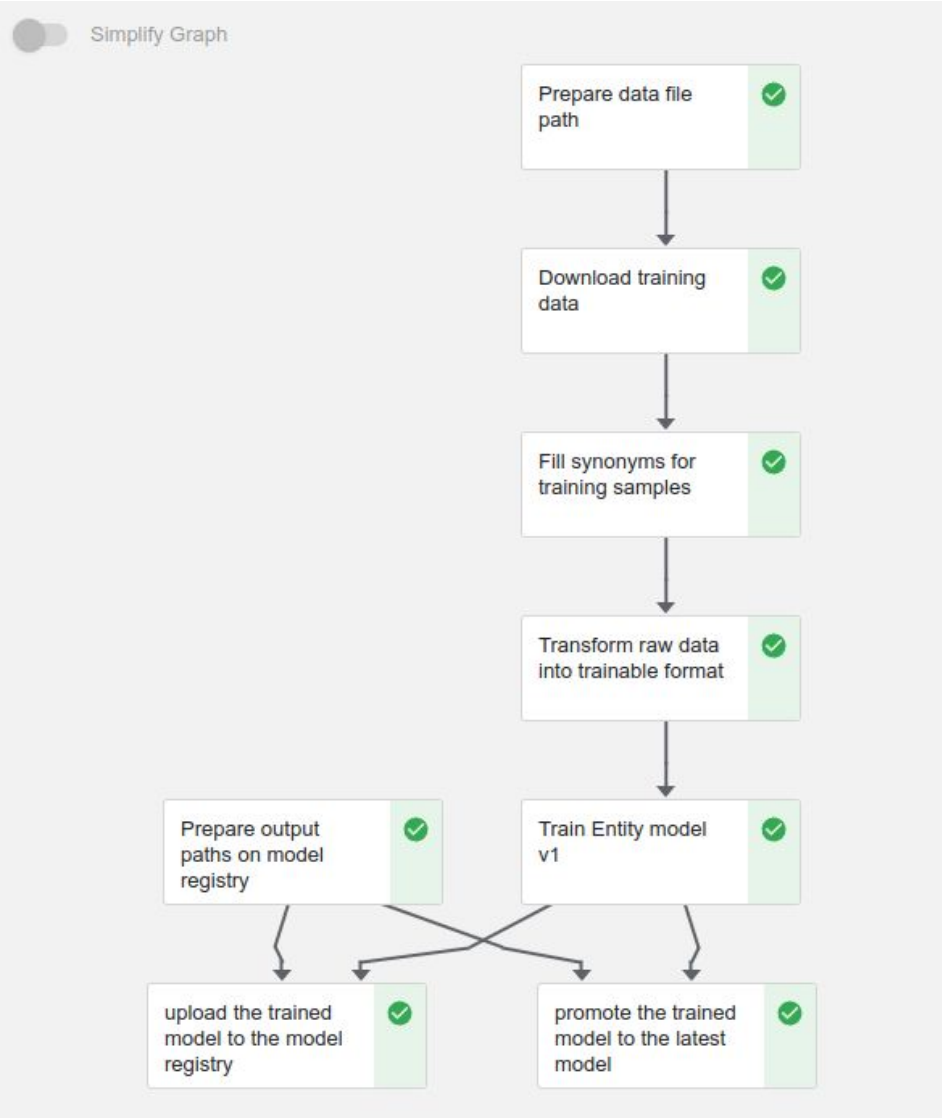


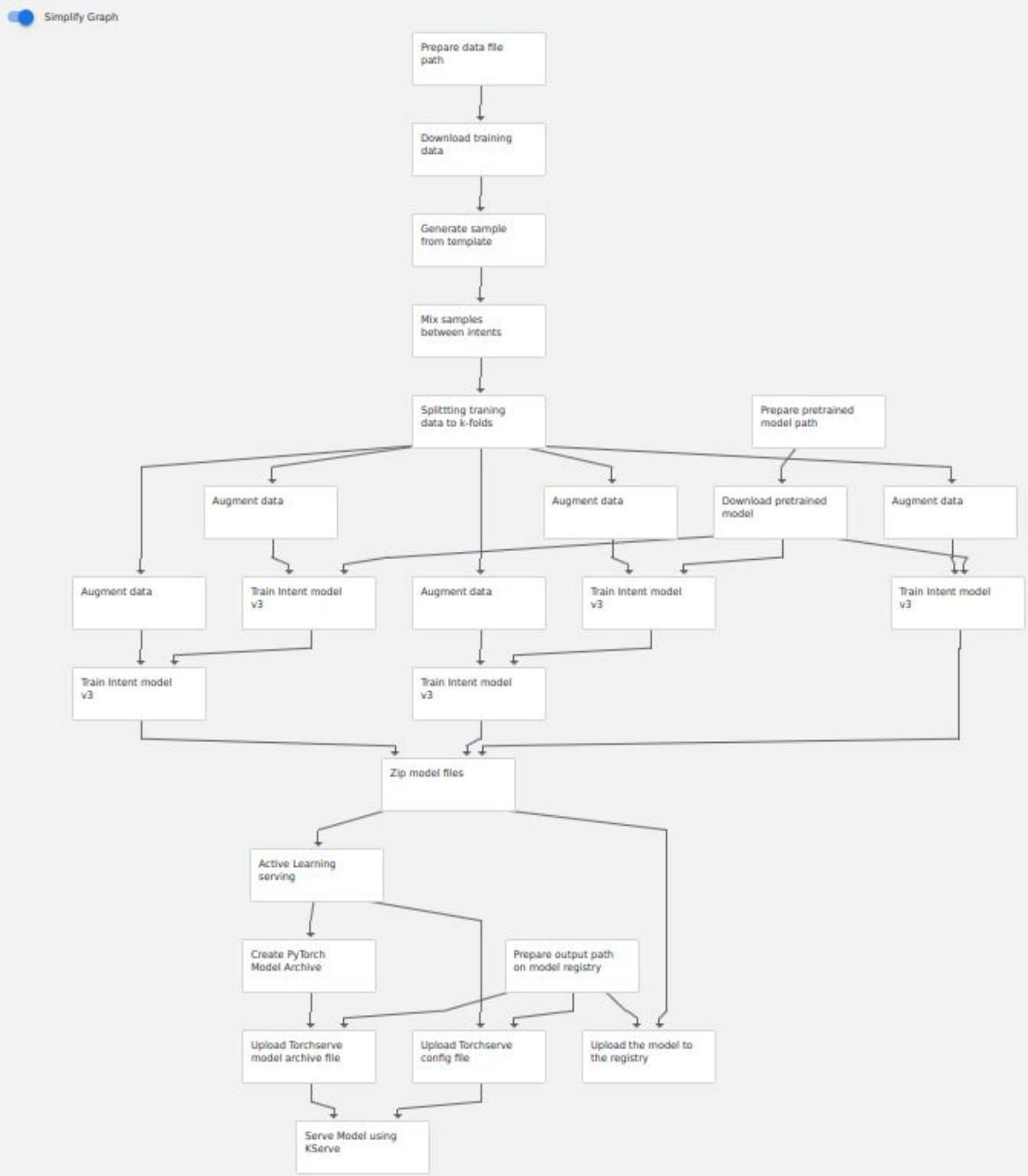
Machine Learning Workflows: MLOps Engineer





Machine Learning Workflows



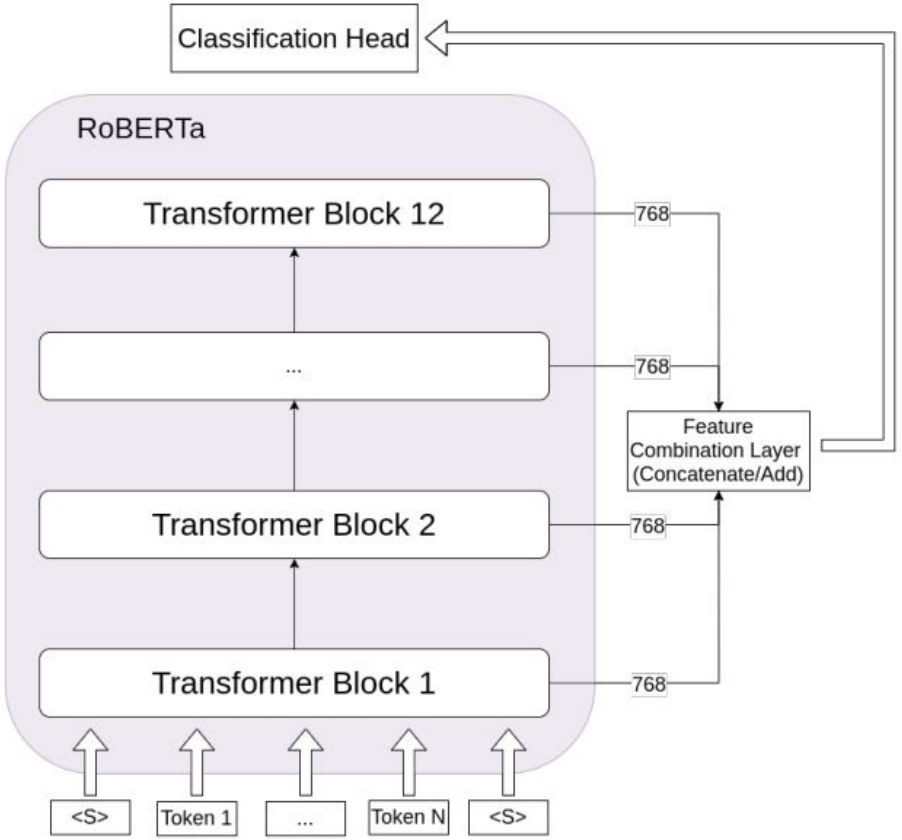


Agenda

1. Sơ lược về NLP
2. Machine Learning Workflows
3. Case Study

Case Study

- Sequence Classification
- Token Classification
- Masked Language Model
- Question Answering
- Dependency parsing / Relation Extraction



Case Study: Sequence Classification

```
# Sequence classification head
self.activation = nn.Tanh()
self.hidden_layer = nn.Linear(
    self.config.hidden_size * 6, self.config.hidden_size
)
self.dropout = nn.Dropout(0.2)
self.classifier = nn.Linear(
    self.config.hidden_size, self.num_classes
)
```

```
def forward(
    self,
    input_ids=None,
    attention_mask=None,
):
    bert_outputs = self.roberta(
        input_ids,
        attention_mask,
    )

    sequence_represent_last = torch.cat(
        bert_outputs[2][-6:], dim=-1
    )[:, 0, :]  # Batch-size, num_subword, bert_embedding_size

    hidden = self.hidden_layer(sequence_represent_last)
    hidden = self.activation(hidden)
    dropout = self.dropout(hidden)
    logits = self.classifier(dropout)  # [batch, sent_len, n_labels]

    return logits
```

Case Study: Token Classification

```
# Entity head
self.activation = nn.Tanh()
self.entity_hidden_layer = nn.Linear(
    self.config.hidden_size * 6, self.config.hidden_size
)
self.entity_dropout = nn.Dropout(0.2)
self.entity_classifier = nn.Linear(
    self.config.hidden_size, self.num_entity_classes
)
```

```
def forward(
    self,
    input_ids=None,
    attention_mask=None,
    word_matrix=None
):
    bert_outputs = self.roberta(
        input_ids,
        attention_mask,
    )

    sequence_represent_last = torch.cat(
        bert_outputs[2][-6:], dim=-1
    )  # Batch-size, num_subword, bert_embedding_size
    word_embedding_last = self.agg_bpe2word(
        sequence_represent_last, word_matrix, "sum"
    )  # Batch-size, num_word, bert_embedding_size

    # Entity classification
    entity_hidden = self.entity_hidden_layer(word_embedding_last)
    entity_hidden = self.activation(entity_hidden)
    entity_dropout = self.entity_dropout(entity_hidden)
    entity_logits = self.entity_classifier(entity_dropout)  # [batch, sent_len, n_labels]

    return entity_logits
```

Case Study: Masked Language Model

```python
def forward(
    self,
    input_ids=None,
    attention_mask=None,
    token_type_ids=None,
    position_ids=None,
    head_mask=None,
    inputs_embeds=None,
    encoder_hidden_states=None,
    encoder_attention_mask=None,
    labels=None,
    cls_labels=None
):

    outputs = self.roberta(
        input_ids,
        attention_mask=attention_mask,
        token_type_ids=token_type_ids,
        position_ids=position_ids,
        head_mask=head_mask,
        inputs_embeds=inputs_embeds,
        encoder_hidden_states=encoder_hidden_states,
        encoder_attention_mask=encoder_attention_mask,
    )
    sequence_output = outputs[0]

    # MLM head
    prediction_scores = self.lm_head(sequence_output)

    # Conver head
    cls_scores = self.conve_head(sequence_output[:, 0, :])
```

```python
class MaskedLMHead(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.layer_norm = nn.LayerNorm(config.hidden_size, eps=config.layer_norm_eps)

        self.decoder = nn.Linear(config.hidden_size, config.vocab_size)
        self.bias = nn.Parameter(torch.zeros(config.vocab_size))
        self.decoder.bias = self.bias

    def forward(self, features, **kwargs):
        x = self.dense(features)
        x = F.gelu(x)
        x = self.layer_norm(x)
        x = self.decoder(x)
        return x

    def _tie_weights(self):
        self.bias = self.decoder.bias
```

Case Study: Question Answering

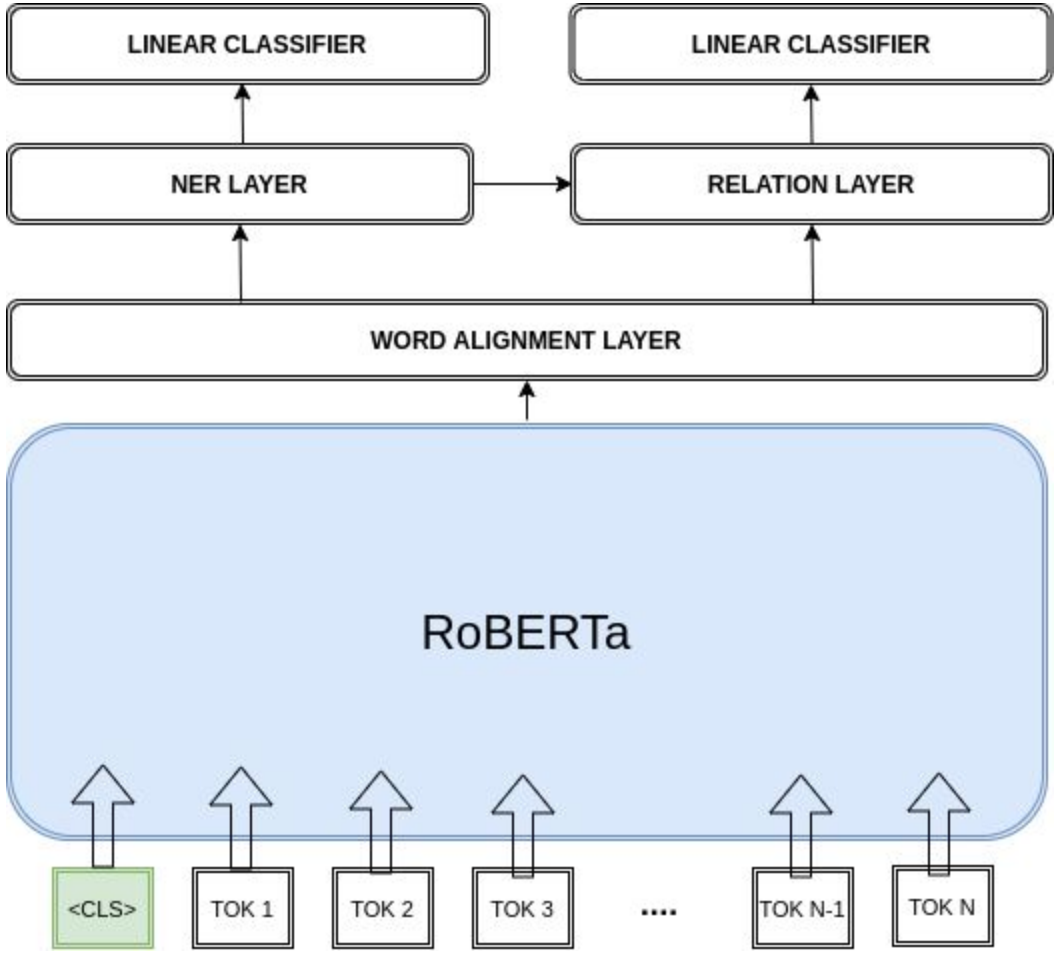
```
self.num_labels = 2
self.cls_num_labels = 2
self.encoder = BERTEncoder(self.model_name)
self.activation = nn.ReLU()
self.dropout = nn.Dropout(0.3)
self.se_hidden_layer = nn.Linear(
    self.encoder.embedding_dim, self.encoder.embedding_dim
)
self.se_classifier = nn.Linear(
    self.encoder.embedding_dim, self.num_labels
)

self.classifier = nn.Linear(
    self.encoder.embedding_dim, self.cls_num_labels
)
```

```
def forward(
    self,
    input_ids=None,
    attention_mask=None,
    cls_label=None,
    start_idx=None,
    end_idx=None,
):
    input_embedding = self.encoder(
        input_ids,
        attention_mask,
    )
    # Start/End of contact name classification
    hidden_state = self.dropout(input_embedding)
    hidden_state = self.se_hidden_layer(hidden_state)
    hidden_state = self.activation(hidden_state)
    hidden_state = self.dropout(hidden_state)
    se_logits = self.se_classifier(hidden_state)
    start_logits, end_logits = se_logits.split(1, dim=-1)
    start_logits = start_logits.squeeze(-1).contiguous()
    end_logits = end_logits.squeeze(-1).contiguous()

    # Sentence classification head
    cls_logit = self.classifier(input_embedding[:, 0, :])

    total_loss, se_loss, cls_loss = None, None, None
    if start_idx is not None and end_idx is not None:
        ignored_index = start_logits.size(1)
        start_idx = start_idx.clamp(0, ignored_index)
        end_idx = end_idx.clamp(0, ignored_index)

        se_loss_fct = nn.CrossEntropyLoss(ignore_index=ignored_index)
        start_loss = se_loss_fct(start_logits, start_idx)
        end_loss = se_loss_fct(end_logits, end_idx)
        se_loss = (start_loss + end_loss)/2
```

Case Study: Dependency parsing / Relation Extraction / Event Extraction

		O	O	O	B_PER	I_PER	O	O	O	O	B_ORG	O	O	O	O	O	O	B_PER	I_PER
		Không	có	việc	Diego	Costa	quay	lại	đội	hình	Chelsea	sau	cuộc	đụng	độ	với	HLV	Antonio	Conte
O	Không	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	có	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	việc	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
B_PER	Diego	x	x	x	x	x	x	x	x	x	NEG	x	x	x	x	x	x	NEG	x
I_PER	Costa	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	quay	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	lại	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	đội	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	hình	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
B_ORG	Chelsea	x	x	x	AFF	x	x	x	x	x	x	x	x	x	x	x	x	NEG	x
O	sau	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	cuộc	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	đụng	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	độ	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	với	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
O	HLV	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
B_PER	Antonio	x	x	x	NEG	x	x	x	x	x	NEG	x	x	x	x	x	x	x	x
I_PER	Conte	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Case Study: Dependency parsing / Relation Extraction / Event Extraction



Case Study: Dependency parsing / Relation Extraction / Event Extraction

```
self.activation = nn.Tanh()

# Head classifier
self.head_hidden_layer_u = nn.Linear(config.hidden_size * 2, config.hidden_size)
self.head_dropout_u = nn.Dropout(config.hidden_dropout_prob)

self.head_hidden_layer_v = nn.Linear(config.hidden_size * 2, config.hidden_size)
self.head_dropout_v = nn.Dropout(config.hidden_dropout_prob)

self.head_classifier = Biaffine(config.hidden_size, bias_x=True, bias_y=False)

# Rel classifier
self.rel_hidden_layer_u = nn.Linear(config.hidden_size * 2 + self.num_steps, config.hidden_size)
self.rel_dropout_u = nn.Dropout(config.hidden_dropout_prob)

self.rel_hidden_layer_v = nn.Linear(config.hidden_size * 2 + self.num_steps, config.hidden_size)
self.rel_dropout_v = nn.Dropout(config.hidden_dropout_prob)

self.rel_classifier = Biaffine(config.hidden_size, self.num_rel_labels, bias_x=True, bias_y=True)

self.head_loss_fct = CrossEntropyLoss(ignore_index=-100)
self.rel_loss_fct = CrossEntropyLoss(ignore_index=-100)
```

Case Study: Dependency parsing / Relation Extraction / Event Extraction

```
bert_outputs = self.roberta(
    input_ids,
    attention_mask=attention_mask,
)

sequence_represent_0 = torch.cat(
    bert_outputs[2][6:8], axis=-1
)  # Batch-size, num_subword, bert_embedding_size
word_embedding_0 = self.agg_bpe2word(
    sequence_represent_0, word_matrix, "sum"
)  # Batch-size, num_word, bert_embedding_size

sequence_represent = torch.cat(
    bert_outputs[2][-2:], axis=-1
)  # Batch-size, num_subword, bert_embedding_size
word_embedding = self.agg_bpe2word(
    sequence_represent, word_matrix, "sum"
)  # Batch-size, num_word, bert_embedding_size

# Trigger classification
trigger_hidden = self.trigger_hidden_layer(word_embedding_0)
trigger_hidden = self.activation(trigger_hidden)
trigger_dropout = self.trigger_dropout(trigger_hidden)
trigger_logits = self.trigger_classifier(trigger_dropout)  # [batch, sent_len, n_labels]

# Argument classification
B, L, H = word_embedding.size()
u = self.activation(self.arg_hidden_u(word_embedding)).unsqueeze(1).expand(B, L, L, -1)
# Batch_size, num_steps, num_steps, rel_emb_size // embed_bert + embed_tags(H)
v = self.activation(self.arg_hidden_u(word_embedding)).unsqueeze(2).expand(B, L, L, -1)
# Batch_size, num_steps, num_steps, rel_emb_size // embed_bert + embed_tags(H)
uv = self.activation(self.selection_uv(torch.cat((u, v), dim=-1))) # Batch_size, num_steps, num_steps , rel_emb_size
arg_logits = torch.einsum('bijh,rh->bijr', uv,
                                    self.arg_emb.weight) # Batch_size, num_steps, num_steps, num_rel_labels
arg_logits = self.activation(arg_logits)
arg_logits = self.arg_projection(arg_logits)  # Add for fun
trigger_labels = torch.argmax(trigger_labels, axis=-1)
trigger_labels = (trigger_labels != 0).type(torch.float)
att_mask = torch.bmm(torch.unsqueeze(trigger_labels, axis=-1), torch.ones(torch.unsqueeze(trigger_labels, axis=1).shape).to(self.device))

if mode == 'train':
    arg_logits = arg_logits * torch.unsqueeze(att_mask, axis=-1)
```

Case Study: Dependency parsing / Relation Extraction / Event Extraction

```
bert_outputs = self.roberta(
    input_ids,
    attention_mask=attention_mask,
)

sequence_represent_0 = torch.cat(
    bert_outputs[2][6:8], axis=-1
)  # Batch-size, num_subword, bert_embedding_size
word_embedding_0 = self.agg_bpe2word(
    sequence_represent_0, word_matrix, "sum"
)  # Batch-size, num_word, bert_embedding_size

sequence_represent = torch.cat(
    bert_outputs[2][-2:], axis=-1
)  # Batch-size, num_subword, bert_embedding_size
word_embedding = self.agg_bpe2word(
    sequence_represent, word_matrix, "sum"
)  # Batch-size, num_word, bert_embedding_size

# Trigger classification
trigger_hidden = self.trigger_hidden_layer(word_embedding_0)
trigger_hidden = self.activation(trigger_hidden)
trigger_dropout = self.trigger_dropout(trigger_hidden)
trigger_logits = self.trigger_classifier(trigger_dropout)  # [batch, sent_len, n_labels]

# Argument classification
B, L, H = word_embedding.size()
u = self.activation(self.arg_hidden_u(word_embedding)).unsqueeze(1).expand(B, L, L, -1)
# Batch_size, num_steps, num_steps, rel_emb_size // embed_bert + embed_tags(H)
v = self.activation(self.arg_hidden_u(word_embedding)).unsqueeze(2).expand(B, L, L, -1)
# Batch_size, num_steps, num_steps, rel_emb_size // embed_bert + embed_tags(H)
uv = self.activation(self.selection_uv(torch.cat((u, v), dim=-1))) # Batch_size, num_steps, num_steps , rel_emb_size
arg_logits = torch.einsum('bijh,rh->bijr', uv,
                                    self.arg_emb.weight) # Batch_size, num_steps, num_steps, num_rel_labels
arg_logits = self.activation(arg_logits)
arg_logits = self.arg_projection(arg_logits)  # Add for fun
trigger_labels = torch.argmax(trigger_labels, axis=-1)
trigger_labels = (trigger_labels != 0).type(torch.float)
att_mask = torch.bmm(torch.unsqueeze(trigger_labels, axis=-1), torch.ones(torch.unsqueeze(trigger_labels, axis=1).shape).to(self.device))

if mode == 'train':
    arg_logits = arg_logits * torch.unsqueeze(att_mask, axis=-1)
```

Case Study: Head Type

- Linear DNN
- CNN
- LSTM / BiLSTM / RNN
- CRF
- Biaffine Attention
- Encoder Feature Extraction:
  - Final Hidden State of [CLS] Token
  - Final Hidden States of Sequence → Reshape
  - Concatenated Hidden States
  - Mean Pooling, Max Pooling,..
- Code example: Intent Classification, Tagging Model,...

VINBIGDATA

THANK YOU

(024) 32 09 78 88 info@vinbigdata.com https://vinbigdata.com
Tầng 9, Century Tower, Times City, 458 Minh Khai, Hai Bà Trưng, Hà Nội, Việt Nam