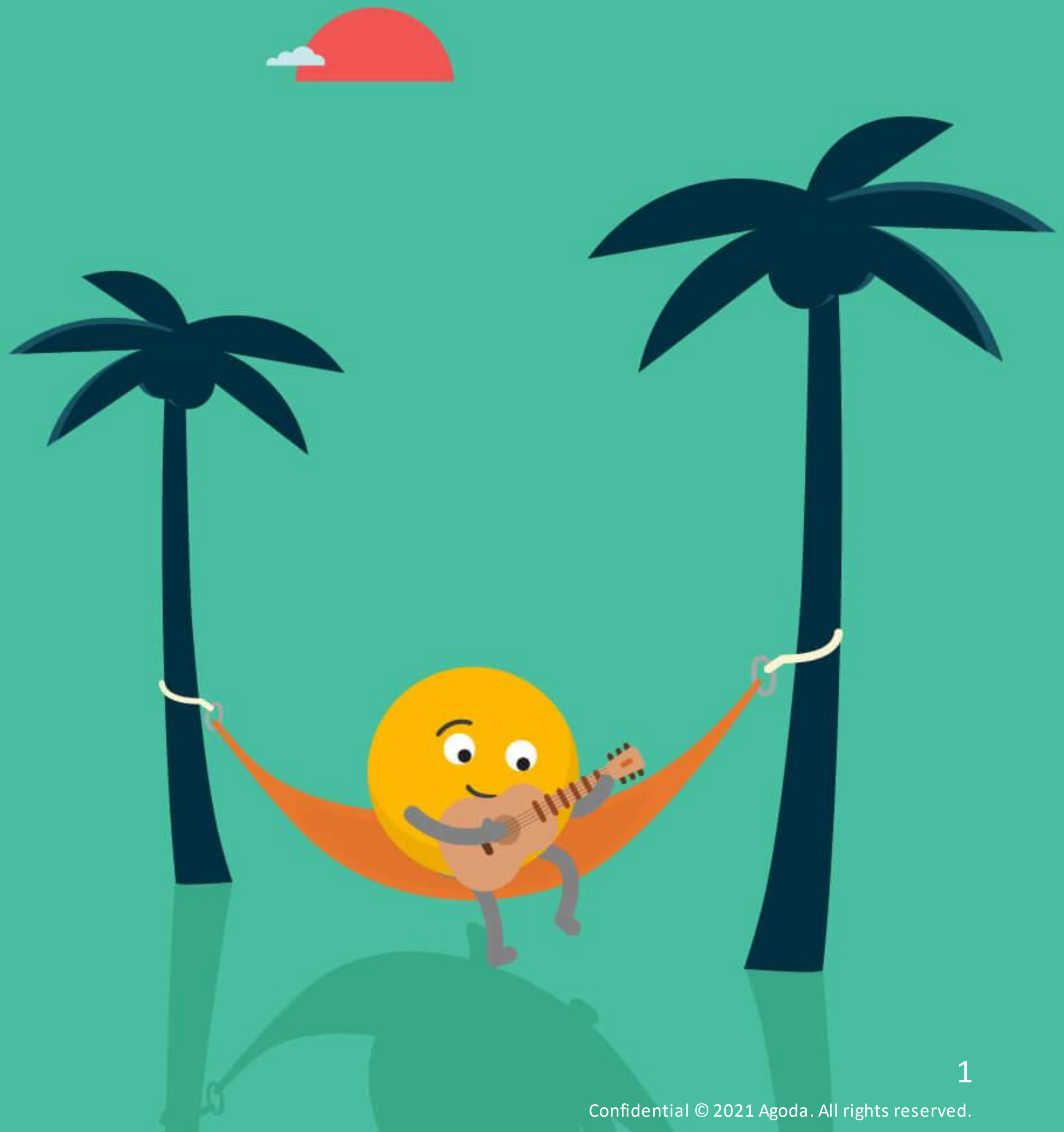
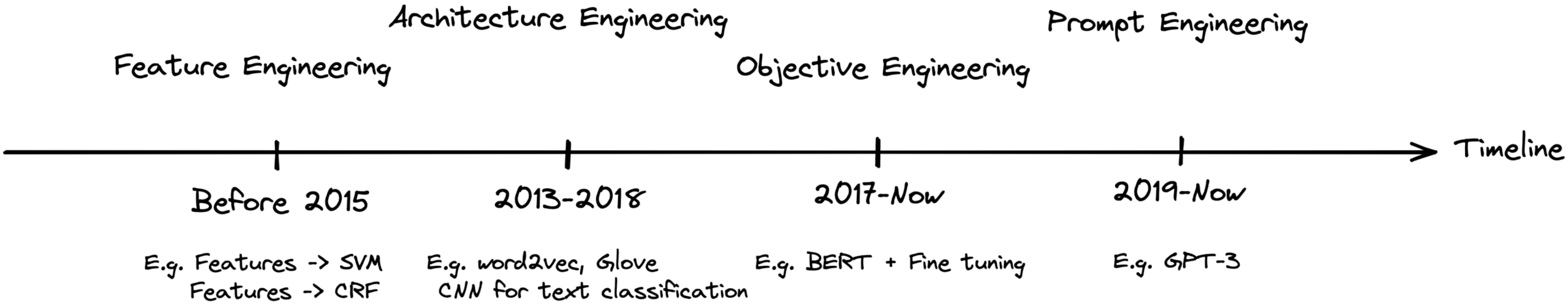
# Prompt-based Learning Intro

Shawn, SG DS Team
20.04.2022



# 4 Paradigms of NLP



# Feature Engineering

- Fully Supervised Learning
  - Non-neural Network
- Popular until 2015
- Non-NN ML model + Manual Features
- E.g. Manual Features + SVM or CRF
- Agoda use case: Cupid

How do we represent text in numeric format?

```
vectorizer = CountVectorizer()
vectorizer.fit(sentences_train)

X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)
```

# Architecture Engineering

- Fully Supervised Learning
- 2013-2018
- Rely on Neural networks
- Do not need manual features
- Need to modify network structure
  - LSTM, CNN
- Sometimes use pre-trained LMs, but often shallow features like embeddings
  - Word2vec, Glove
- E.g. CNN for text classification
- Agoda use case: Snippet Sentiment

```
df['Processed_Reviews'] = df.review.apply(lambda x: clean_text(x))
tokenizer.fit_on_texts(df['Processed_Reviews'])
list_tokenized_train = tokenizer.texts_to_sequences(df['Processed_Reviews'])

X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
y = df['sentiment']

embed_size = 128
model = Sequential()
model.add(Embedding(max_features, embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

batch_size = 100
epochs = 3
model.fit(X_t,y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

agoda

# Objective Engineering

- Pre-train , Fine-tune
- 2017 – Now
- Pre-trained LMs (PLMs) used as starting point
  - Both shallow and deep features
- Less work on architecture design, but engineer objective functions
- E.g. BERT, RoBERTa, XLM-R -> Fine Tuning
- Agoda use case: IRIS Chat Intent

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained(
  "distilbert-base-uncased", num_labels=2)

tokenized_train = small_train_dataset.map(preprocess_function, batched=True)
tokenized_test = small_test_dataset.map(preprocess_function, batched=True)

repo_name = "finetuning-sentiment-model-3000-samples"

training_args = TrainingArguments(
   output_dir=repo_name,
   learning_rate=2e-5,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   num_train_epochs=2,
   weight_decay=0.01,
   save_strategy="epoch",
   push_to_hub=True,
)

trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_train,
   eval_dataset=tokenized_test,
   tokenizer=tokenizer,
   data_collator=data_collator,
   compute_metrics=compute_metrics,
)

trainer.train()

trainer.evaluate()
```

# Huggingface

- More than 215 sentiment analysis model on huggingface hubs
- Easy to use for common NLP tasks
  - Sequence Classification
  - Extractive Question Answering
  - Language Modeling
  - Named Entity Recognition
  - Summarization
  - Translation
  - …

```
pip install -q transformers
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["I love you", "I hate you"]
sentiment_pipeline(data)

specific_model = pipeline(model="cardiffnlp/twitter-roberta-base-sentiment")
specific_model(data)
```

agoda

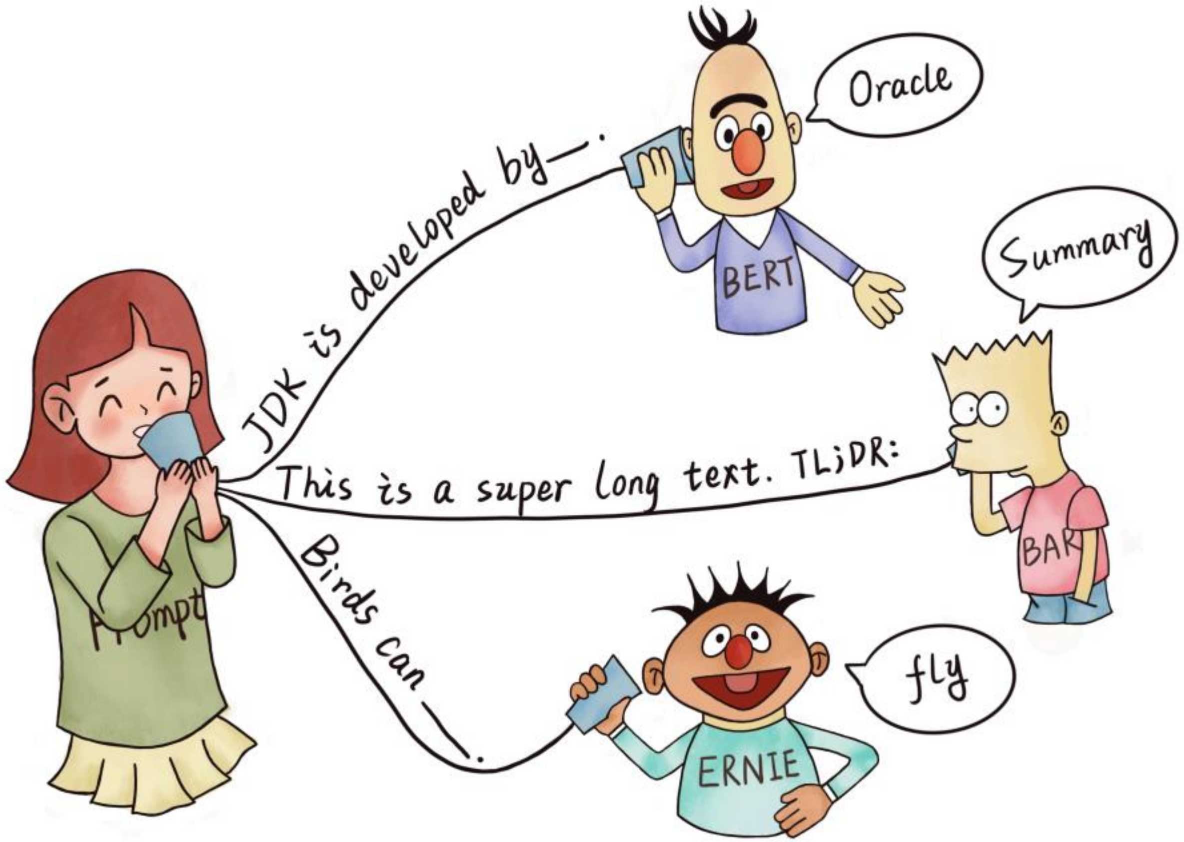
# Prompt Engineering

- Pre-train, Prompt, Predict
- 2019 – Now
- NLP tasks are modeled entirely by relying on LMs
- The task of shallow & deep features extraction, and prediction of data are all given to LM
- Engineering of prompts is required
- E.g. GPT-3

```
plm, tokenizer, model_config, WrapperClass = load_plm("t5", "t5-base")

promptTemplate = ManualTemplate(
    text = '{"placeholder":"text_a"} It was {"mask"}',
    tokenizer = tokenizer,
)

promptVerbalizer = ManualVerbalizer(
    classes = classes,
    label_words = {
        "negative": ["bad"],
        "positive": ["good", "wonderful", "great"],
    },
    tokenizer = tokenizer,
)

promptModel = PromptForClassification(
    template = promptTemplate,
    plm = plm,
    verbalizer = promptVerbalizer,
)

data_loader = PromptDataLoader(
    dataset = dataset,
    tokenizer = tokenizer,
    template = promptTemplate,
    tokenizer_wrapper_class=WrapperClass,
)

promptModel.eval()
with torch.no_grad():
    for batch in data_loader:
        logits = promptModel(batch)
        preds = torch.argmax(logits, dim = -1)
        print(classes[preds])
```

# What is Prompting

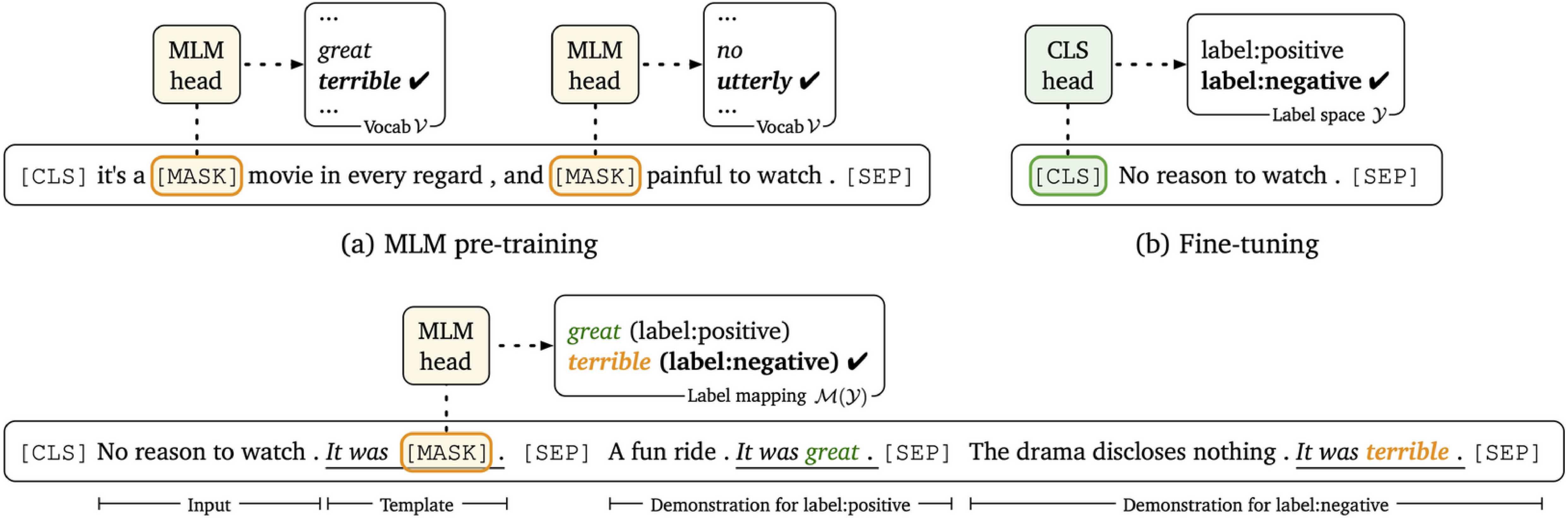
- Encouraging a pretrained model to make particular predictions by providing a “prompt” specifying the task to be done



# What is Prompt

- Prompt is a piece of text inserted in the input examples, so that the original task can be formulated as a (masked) language modeling problem.

## Why Prompts?




(c) Prompt-based fine-tuning with demonstrations (our approach)

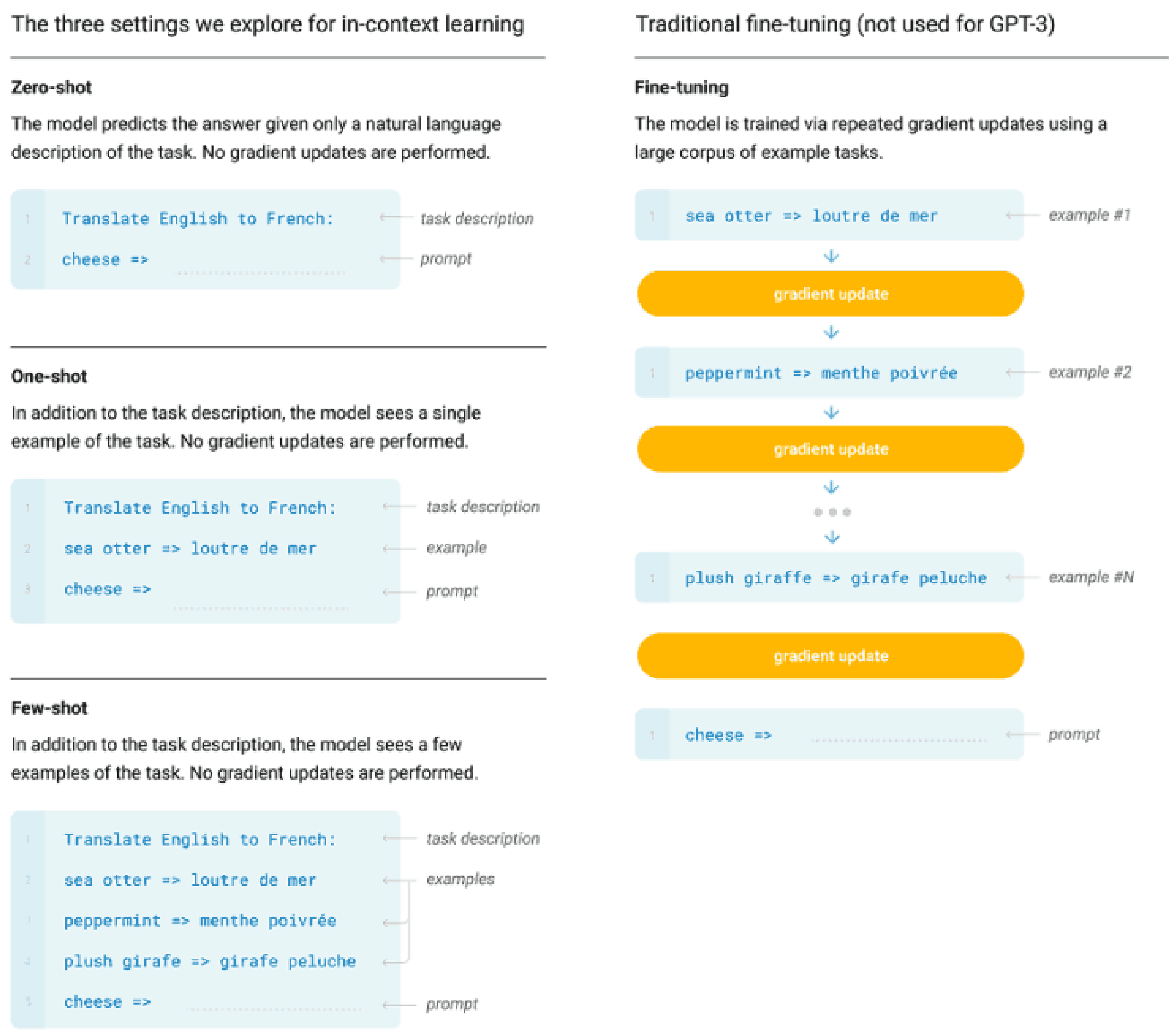
# Workflow of Prompting

- Prompt Addition
  - Transform input x into prompt x'
    - Define template: input [x] and answer [z]
    - Fill in the input slot [x]
- Answer Prediction
  - Using pretrained Language Model to predict
    - Fill [z]
- Answer-Label Mapping
  - Map the answer to a class label

- Input x
  - "I love this movie"
- Template
  - [x] Overall, it was a [z] movie
- Prompting x'
  - "I love this movie, Overall it was a [z] movie."
- Prediction x'
  - "I love this movie, Overall it was a fantastic movie."
- Mapping
  - fantastic = Positive

Utilizing the pretrained Language Model without Fine-tuning, good for few-shot cases.

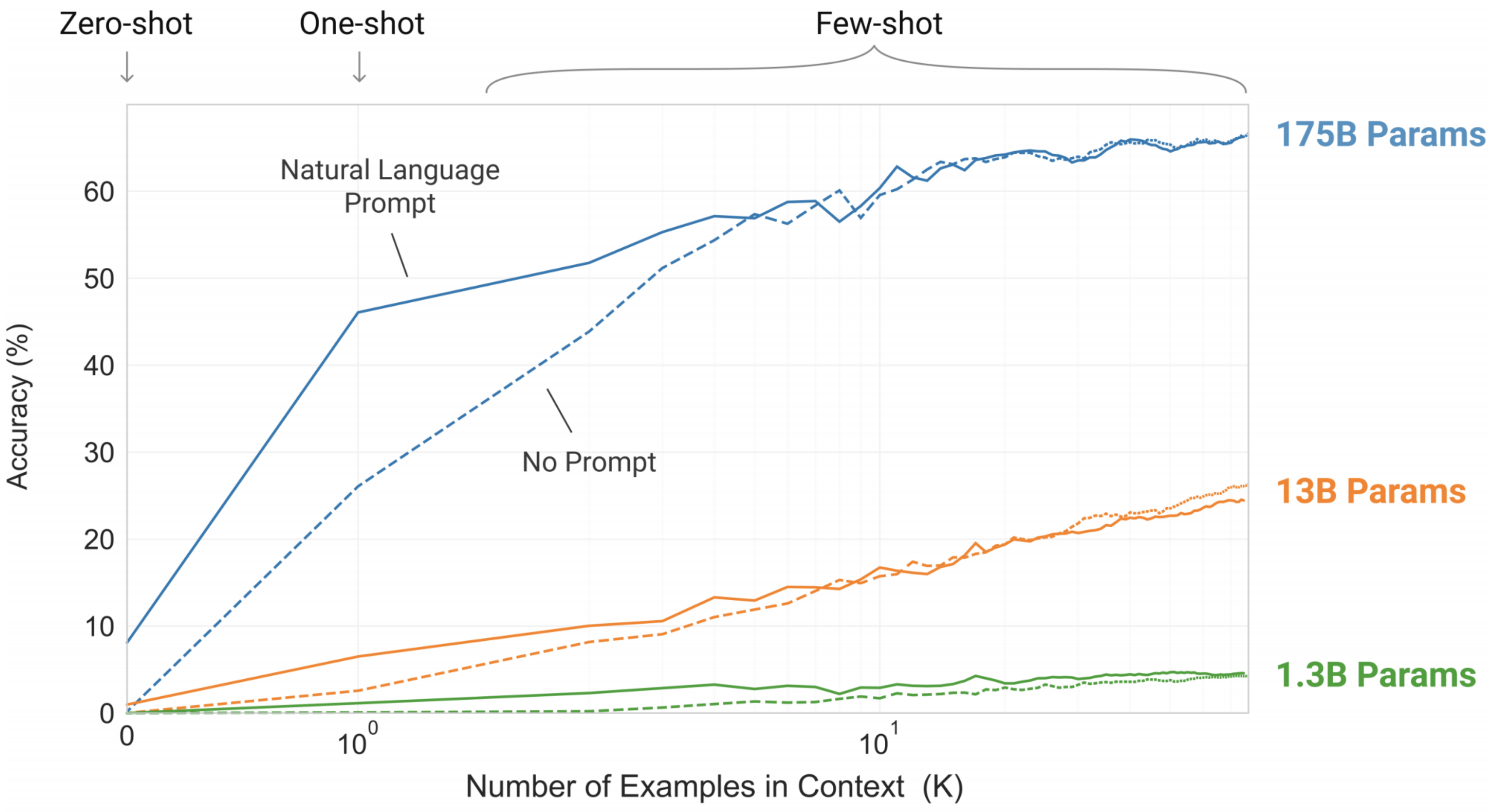
# What is zero-shot, one-shot, few-shot comparing with traditional fine-tuning



# Research Questions

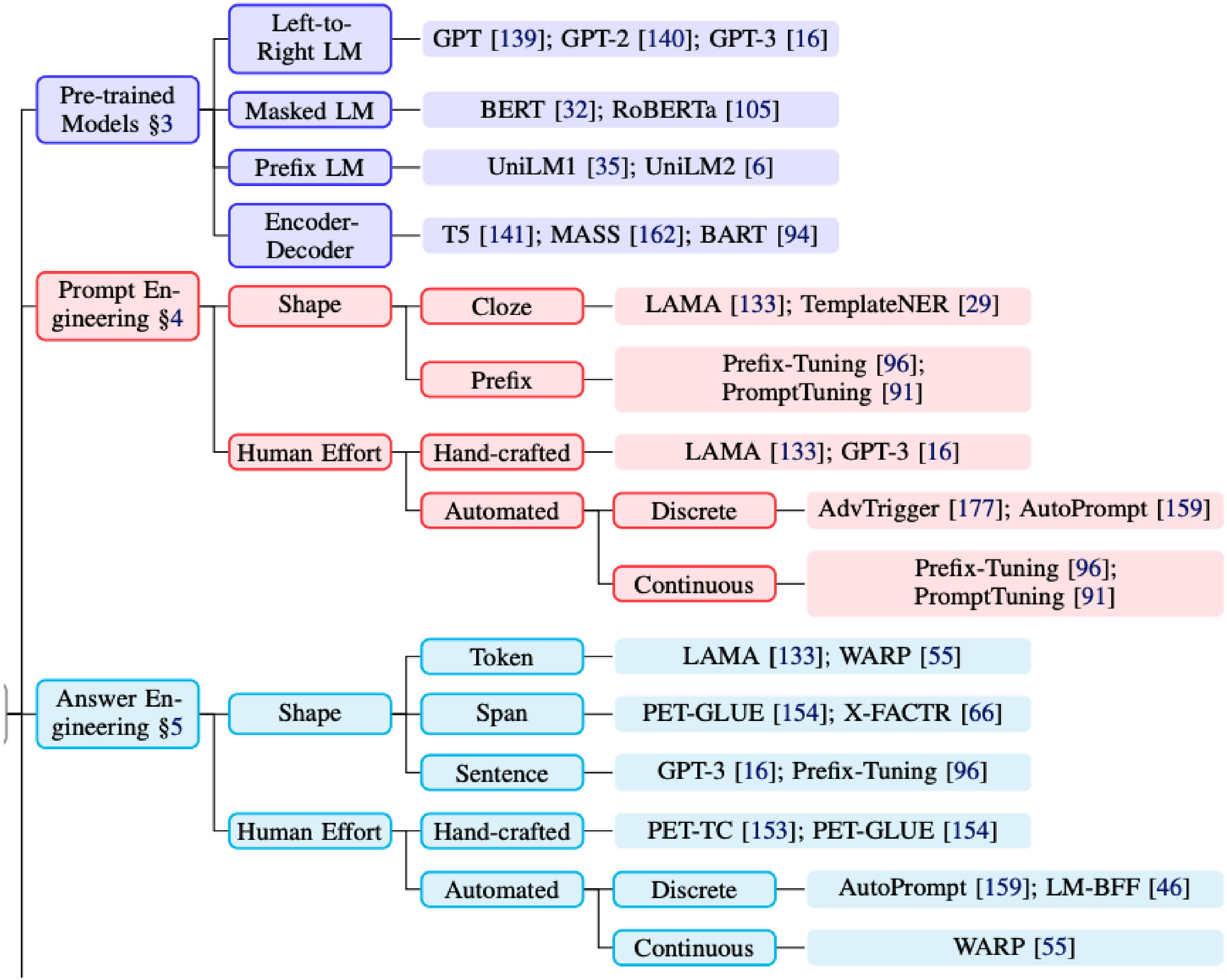
Parameters
- BERT/RoBERTa: 0.3 Billion
- GPT-3: **175 Billion**

Can we use prompt for smaller models?
How to define prompt?
How to find good prompt?
What is the role of demonstrations?
- LM-BFF (Gao et al. ACL 2021)
- OptiPrompt (Zhong et al. NAACL 2021)



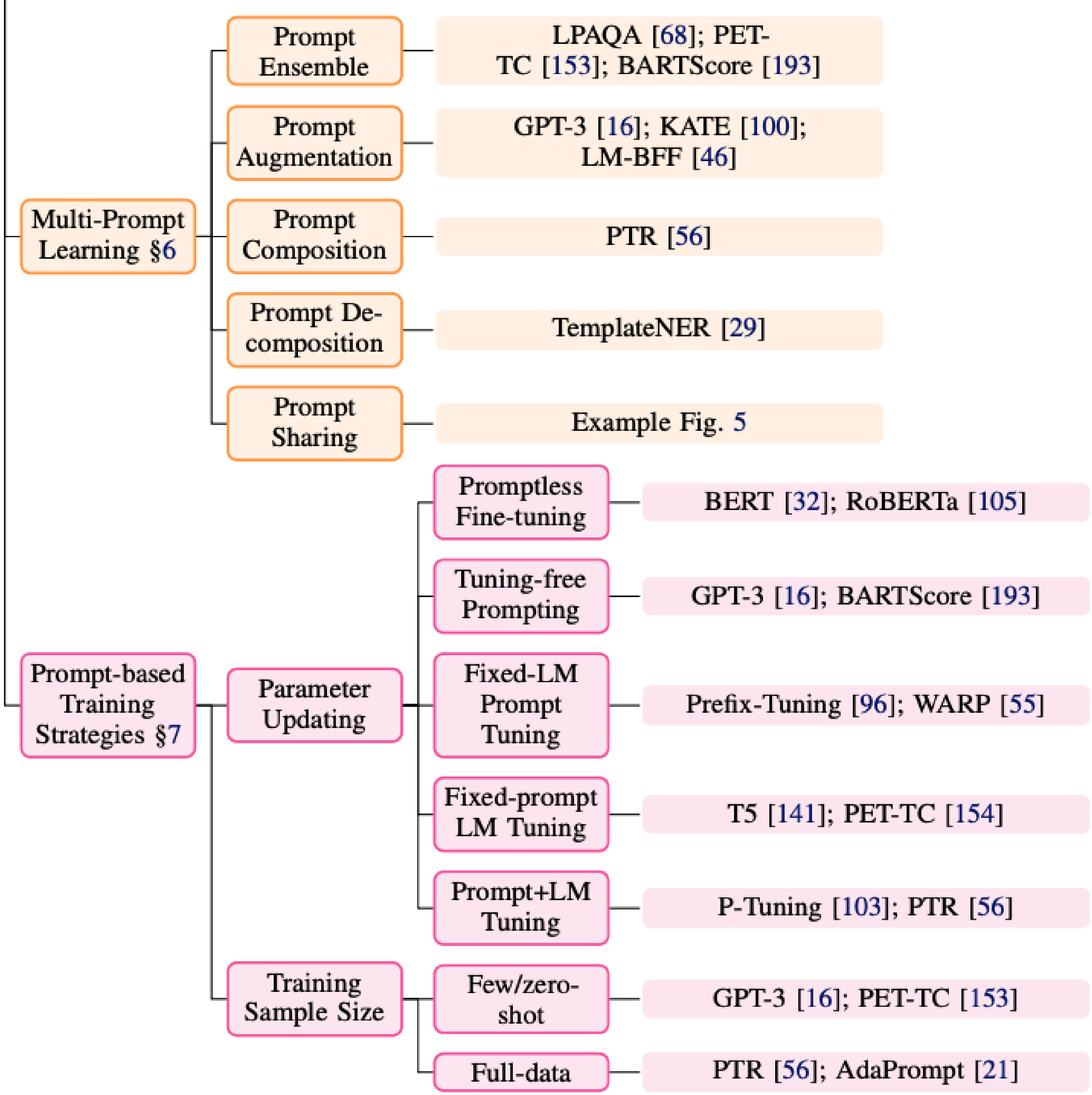
# Design Considerations

- **Pre-trained Model Choice**
- **Prompt Engineering**
- **Answer Engineering**
- Expanding the Paradigm
- Prompt-based Training Strategies



# Design Considerations

- Pre-trained Model Choice
- Prompt Engineering
- Answer Engineering
- **Expanding the Paradigm**
- **Prompt-based Training Strategies**



# Resources

1. CMU Advanced NLP 2021 (10): Prompting + Sequence-to-sequence Pre-training
    1. Youtube Link
2. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing
    1. Paper
    2. Youtube 1, Youtube 2, Youtube 3, Youtube 4, Youtube 5
3. Prompting: Better Ways of Using Language Models for NLP Tasks
    1. Blog
4. GPT-J-6B
    1. Github
    2. Colab Demo

# Q&A