Guidelines and standard metrics for evaluating LLMs

INTRODUCTION TO LLMS IN PYTHON

Iván Palomares Carrascosa, PhD
Senior Data Science & AI Manager

datacamp

Evaluation metrics: classification accuracy

```
from transformers import pipeline
from sklearn.metrics import accuracy_score

sentiment_analysis = pipeline("sentiment-analysis")

test_examples = [
    {"text": "I love this product!", "label": 1},
    {"text": "The service was terrible.", "label": 0},
    {"text": "This movie is amazing.", "label": 1},
    {"text": "I'm disappointed with the quality.", "label": 0},
]

predictions = sentiment_analysis(
  [example["text"] for example in test_examples]
)
```

- Load **scikit-learn** accuracy score metric
- Load **sentiment analysis** pipeline

- Define four **test examples** with their associated **labels** (ground-truth)
- Pass them to the model for inference

Evaluation metrics: classification accuracy

```
true_labels = [example["label"] for example in test_examples]
predicted_labels = [1 if pred["label"] == "POSITIVE"
                    else 0 for pred in predictions]

accuracy = accuracy_score(true_labels, predicted_labels)

# Print results
print("Test Examples:")
for example, pred_label in zip(test_examples, predicted_labels):
    print(f"Text: {example['text']}, Prediction: {pred_label}")

print(f"Accuracy: {accuracy:.2%}")
```

```
Test Examples:
Text: I love this product!, Predicted Label: 1
Text: The service was terrible., Predicted Label: 0
Text: This movie is amazing., Predicted Label: 1
Text: I'm disappointed with the quality., Predicted Label: 0
Accuracy: 100.00%
```

Accuracy: % of correctly made (class) predictions

- Acc. = (TP + TN) / (TP + TN + FP + FN)

The evaluate library

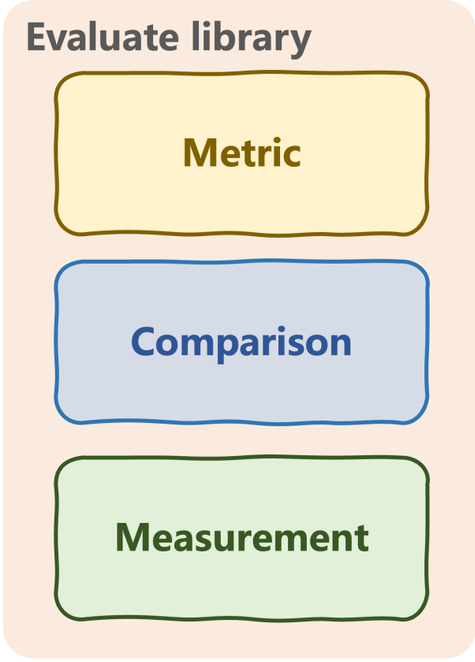
```
import evaluate
accuracy = evaluate.load("accuracy")
print(accuracy.description)
```

```
Accuracy is the proportion of correct predictions among the
total number of cases processed. It can be computed with:
Accuracy = (TP + TN) / (TP + TN + FP + FN) Where:
TP: True positive
TN: True negative
FP: False positive
FN: False negative
```

```
print(evaluate.load("f1").description)
```

```
The F1 score is the harmonic mean of the precision and
recall. It can be computed with the equation:
F1 = 2 * (precision * recall) / (precision + recall)
```

- **Metric:** evaluate model performance based on ground truth
- **Comparison:** compare two models
- **Measurement:** insight on dataset properties



The evaluate library

```
print(accuracy.features)
```

```
{'predictions': Value(dtype='int32', id=None),
 'references': Value(dtype='int32', id=None)}
```

```
f1 = evaluate.load("f1")
print(f1.features)
```

```
{'predictions': Value(dtype='int32', id=None),
 'references': Value(dtype='int32', id=None)}
```

```
pearson_corr = evaluate.load("pearsonr")
print(pearson_corr.features)
```

```
{'predictions': Value(dtype='float32', id=None),
'references': Value(dtype='float32', id=None)}
```

Inspecting required inputs by a metric

- `'predictions'` and `'references'` are two collections: *model outputs* vs *ground truth*
  - `.features` : indicates the type supported, e.g. `'int32'` for class labels
  - Certain metrics like **Pearson correlation** operate on real values, `'float32'`

The evaluate library

```
accuracy = evaluate.load("accuracy")
precision = evaluate.load("precision")
recall = evaluate.load("recall")
f1 = evaluate.load("f1")
real_labels = [0,1,0,1,1]
predicted_labels = [0,0,0,1,1]
```

`.compute()` **method:** calculate metric score given **predictions** and **ground-truth labels**

```
print(accuracy.compute(references=real_labels, predictions=predicted_labels))
print(precision.compute(references=real_labels, predictions=predicted_labels))
print(recall.compute(references=real_labels, predictions=predicted_labels))
print(f1.compute(references=real_labels, predictions=predicted_labels))
```

```
{'accuracy': 0.8}
{'precision': 1.0}
{'recall': 0.6666666666666666}
{'f1': 0.8}
```

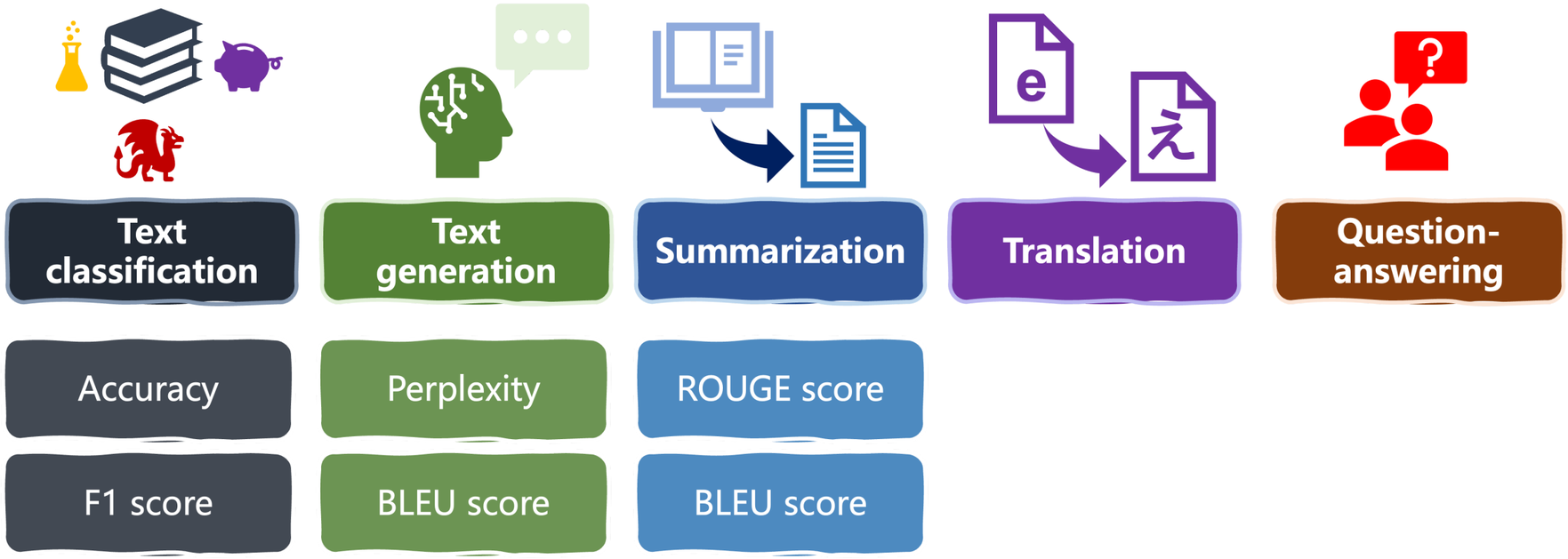
LLM tasks and metrics

LLM tasks and metrics

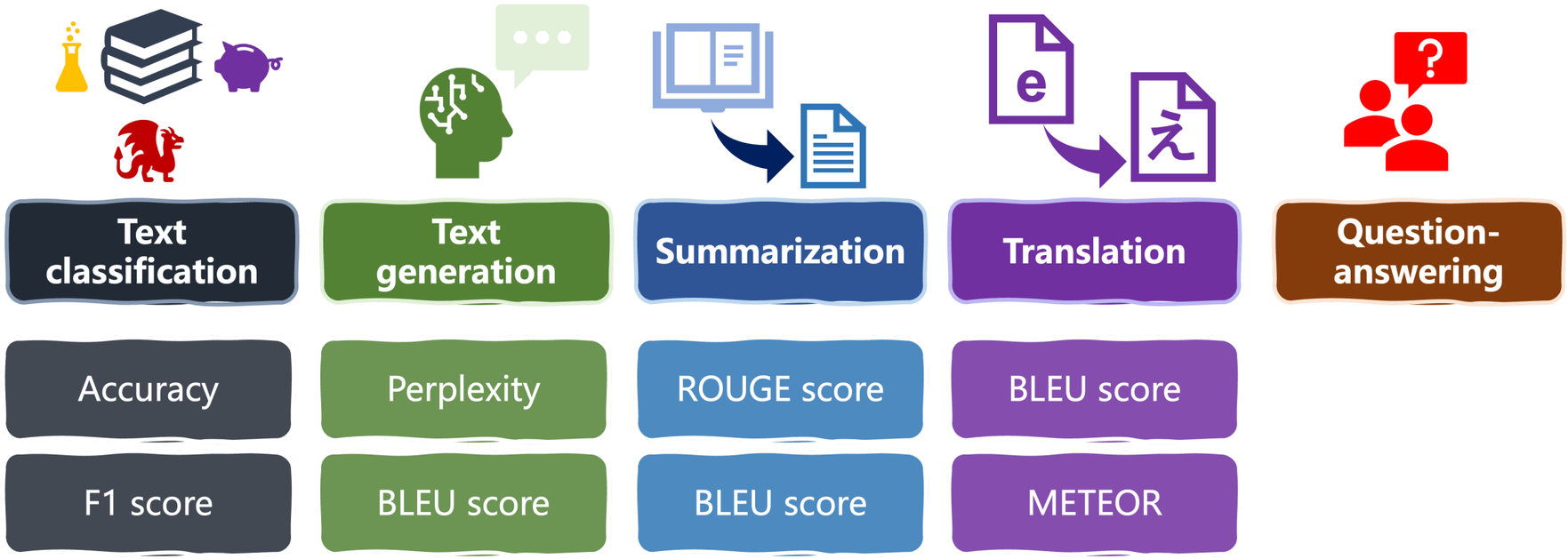
- Accuracy
- F1 score

LLM tasks and metrics

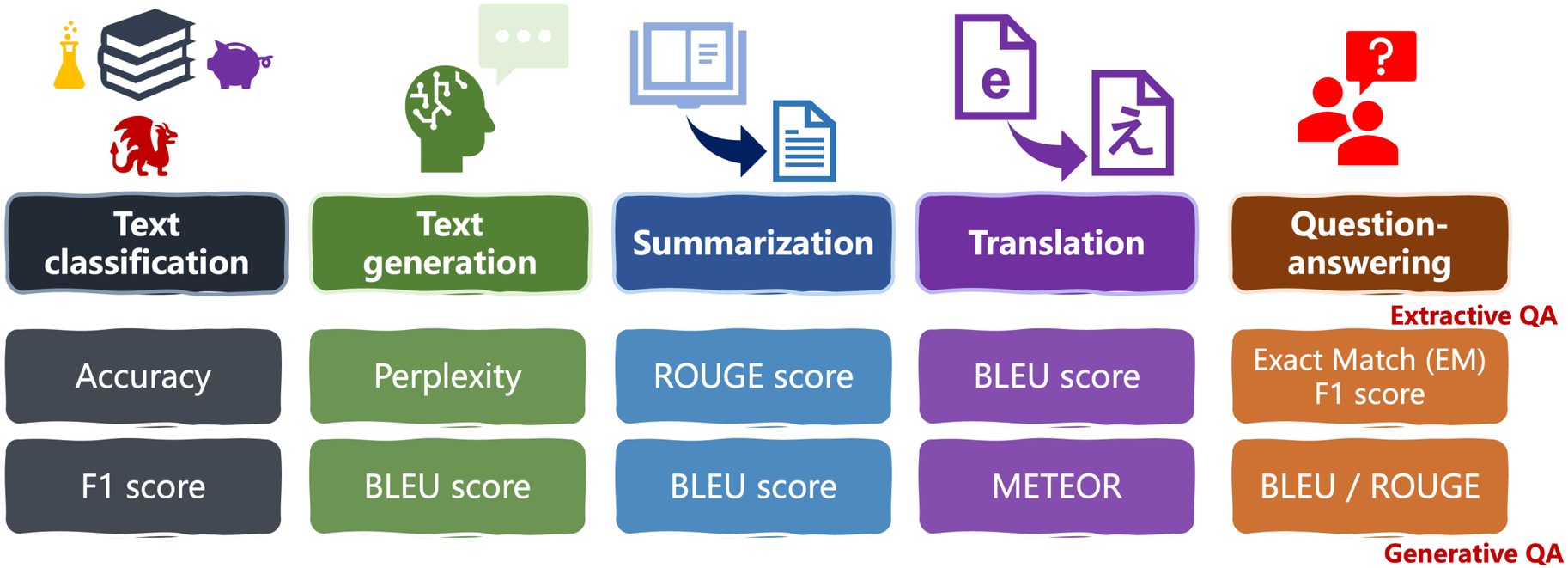
LLM tasks and metrics



LLM tasks and metrics



LLM tasks and metrics



- Be **aware:** each metric brings its own *insights*, but they also have their *limitations*
- Be **realistic**: a pre-trained book genre classifier may not accurately classify sentiments
- Be **comprehensive**: use a *combination of metrics* (and domain-specific *KPIs* where possible)

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

Specialized metrics for language tasks

INTRODUCTION TO LLMS IN PYTHON

Iván Palomares Carrascosa, PhD
Senior Data Science & AI Manager

datacamp

Perplexity in text generation

```
model_name = "gpt2"
# ... Load model and tokenizer ...

prompt = "Latest research findings in Antarctica show"
prompt_ids = tokenizer.encode(test_sentence, return_tensors="pt")
output = model.generate(prompt_ids, max_length=17)
generated_text = tokenizer.decode(
  output[0], skip_special_tokens=True)
print(generated_text)
```

```
Latest research findings in Antarctica show that the ice sheet
is melting faster than previously thought.
```

```
perplexity = evaluate.load("perplexity", module_type="metric")
results = perplexity.compute(model_id='gpt2',
                             predictions=generated_text)
print(results['mean_perplexity'])
```

```
2867.6985334123883
```

> **Perplexity:** model's ability to predict the next word accurately and confidently
>
> - Calculated upon model's output logits
> - range `[0, inf)` : lower is better

Example: generate a few tokens upon a prompt and calculate perplexity of sequence

- Likelihood that the model generated the sequence
- Average sequences' perplexity: `'mean_perplexity'`

ROUGE score in text summarization

```
rouge = evaluate.load("rouge")
predictions = ["""as we learn more about the frequency and
        size distribution of exoplanets ,we are discovering
        that terrestrial planets are exceedingly common."""]
references = ["""The more we learn about the frequency and
        size distribution of exoplantes, the more confident we
        are that they are exceedingly common."""]

results = rouge.compute(predictions=predictions,
                        references=references)
print(results)
```

```
{'rouge1': 0.7441860465116279, 'rouge2': 0.4878048780487805,
 'rougeL': 0.6976744186046512, 'rougeLsum': 0.6976744186046512}
```

> **ROUGE:** overlap between generated a summary and reference summaries.
>
> - n-grams, word overlap, ...

- `predictions:` LLM outputs
- `references` : human-provided summaries

ROUGE scores:

- `rouge1` : unigram overlap
- `rouge2` : bigram overlap
- `rougeL` : long overlapping subsequences

[1] ROUGE: Recall-Oriented Understudy for Gist Evaluation

BLEU score in translation

> **BLEU:** measures translation quality as the **correspondence between** an **LLM output** and one (or more) **human references.**

- `predictions:` LLM's output translation
- `references` : human translations

```
import evaluate
from transformers import pipeline
bleu = evaluate.load("bleu")

translator = pipeline(
  "translation", model="Helsinki-NLP/opus-mt-es-en")
input = "Qué hermoso día"
references = [["What a gorgeous day",
               "What a beautiful day"]]
```

```
translated_outputs = translator(input)
translated_sentence = translated_outputs[0]['translation_text']
print("Translation: ", translated_sentence)
```

```
Translation:  What a beautiful day.
```

```
results = bleu.compute(
  predictions=[translated_sentence], references=references)
print(results)
```

```
{'bleu': 0.7598356856515925,
 'precisions': [1.0, 1.0, 0.6666666666666666, 0.5],
 'brevity_penalty': 1.0, 'length_ratio': 1.0,
 'translation_length': 5, 'reference_length': 5}
```

`0-1` score, indicating similarity to references

`precisions` : mean n-gram precisions

[1] BLEU: BiLingual Evaluation Understudy

METEOR score in translation

```
bleu = evaluate.load("bleu")
meteor = evaluate.load("meteor")
pred = ["He thought it right and necessary to become a
        knight-errant, roaming the world in armor, seeking
        adventures and practicing the deeds he had read about
        in chivalric tales."]
ref = ["He believed it was proper and essential to transform
       into a knight-errant, traveling the world in armor,
       pursuing adventures, and enacting the heroic deeds he
       had encountered in tales of chivalry."]

results_b = bleu.compute(predictions=pred, references=ref)
results_m = meteor.compute(predictions=pred, references=ref)
print("Bleu: ", results_bleu['bleu'])
print("Meteor: ", results_meteor['meteor'])
```

```
Bleu:  0.190888841781992524
Meteor:  0.5350702240481536
```

METEOR: overcome ROUGE and BLEU limitations. It comprehensively assesses:

- N-gram overlap, precision, recall, synonyms, stemming, word order
- More computationally expensive

- Overall quality of LLM-generated text
- `0-1` score: higher is better

[1] METEOR: Metric for Evaluation of Translation with Explicit ORdering

Exact Match (EM) in question answering

```
from evaluate import load
em_metric = load("exact_match")

exact_match = evaluate.load("exact_match")
predictions = ["The cat sat on the mat.",
               "Theaters are great.",
               "It's like comparing oranges and apples."]
references = ["The cat sat on the mat?",
              "Theaters are great.",
              "It's like comparing apples and oranges."]

results = exact_match.compute(
  references=references, predictions=predictions)
print(results)
```

```
{'exact_match': 0.3333333333333333}
```

Exact Match (EM):

- EM = 1 if an LLM's output exactly matches its reference answer
- EM = 0 otherwise

- Normally used in conjunction with **F1 score**

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

Model fine-tuning using human feedback

INTRODUCTION TO LLMS IN PYTHON

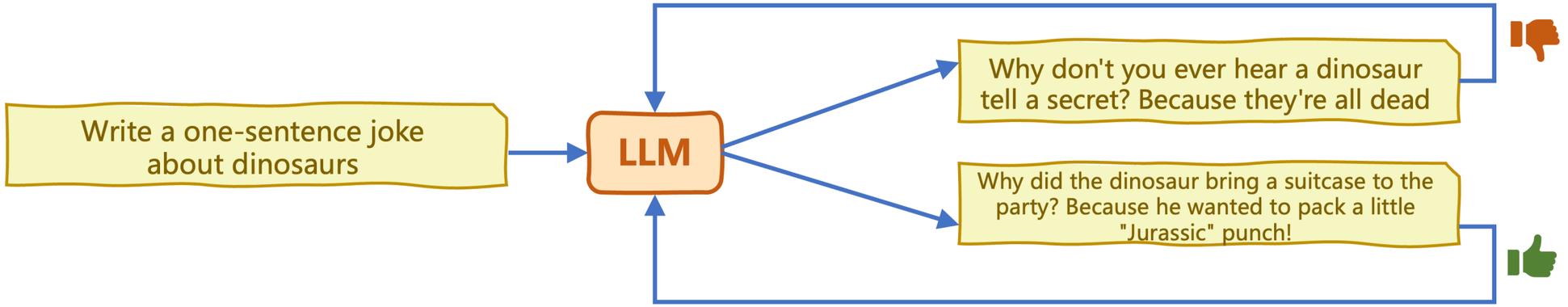
Iván Palomares Carrascosa, PhD
Senior Data Science & AI Manager

datacamp

Why human feedback in LLMs

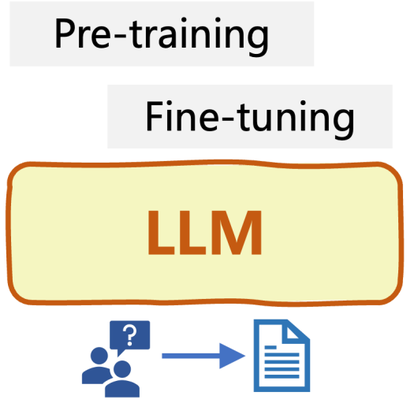
> "*What makes an LLM **good?***", "*What is the LLM **user** looking for?*"
>
> - Objective, **subjective** and **context-dependent** criteria
>   - Truthfulness, originality, fine-grained detail vs. concise responses, etc.

- **Objective metrics** cannot fully capture **subjective quality in LLM outputs**
- Use **human feedback** as a guide (loss function) to **optimize LLM outputs**



Reinforcement Learning from Human Feedback (RLHF)

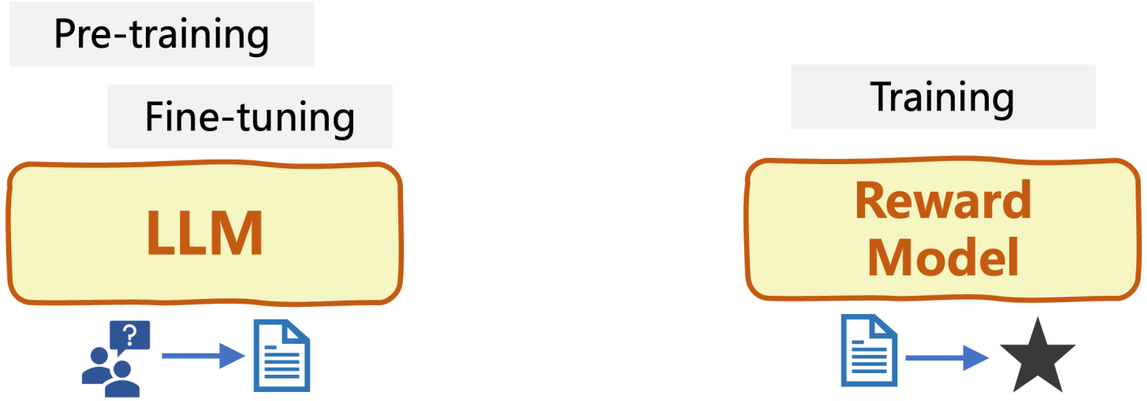
> **Reinforcement Learning (RL):** an agent learns to make **decisions** upon feedback -**rewards**-, adapting its behavior to maximize **cumulative reward** over time



1. Initial **LLM**

Reinforcement Learning from Human Feedback (RLHF)

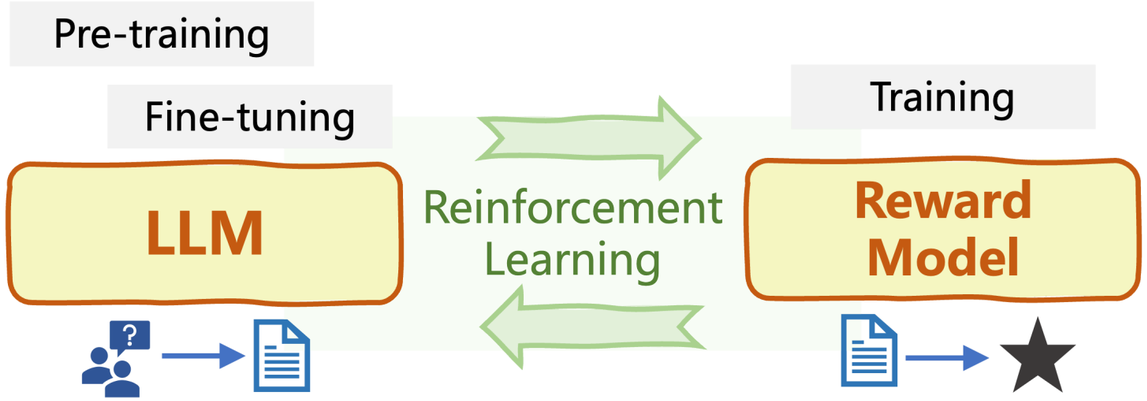
> **Reinforcement Learning (RL):** an agent learns to make **decisions** upon feedback -**rewards**-, adapting its behavior to maximize **cumulative reward** over time



1. Initial **LLM**
2. Train a **Reward Model** (RM)

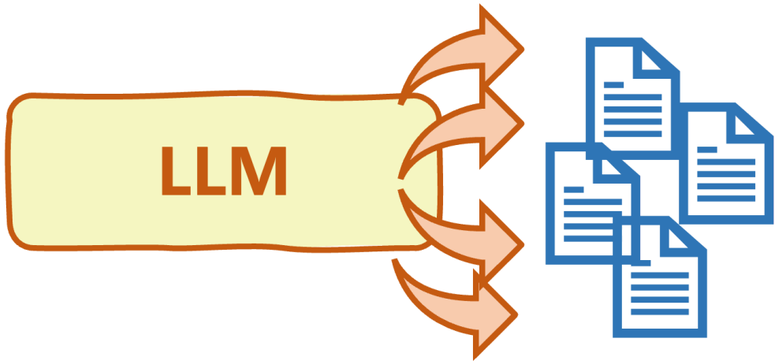
Reinforcement Learning from Human Feedback (RLHF)

> **Reinforcement Learning (RL):** an agent learns to make **decisions** upon feedback -**rewards**-, adapting its behavior to maximize **cumulative reward** over time



1. Initial **LLM**
2. Train a **Reward Model** (RM)
3. Optimize (fine-tune) **LLM** using **RL algorithm** (e.g. PPO) based on trained **RM**

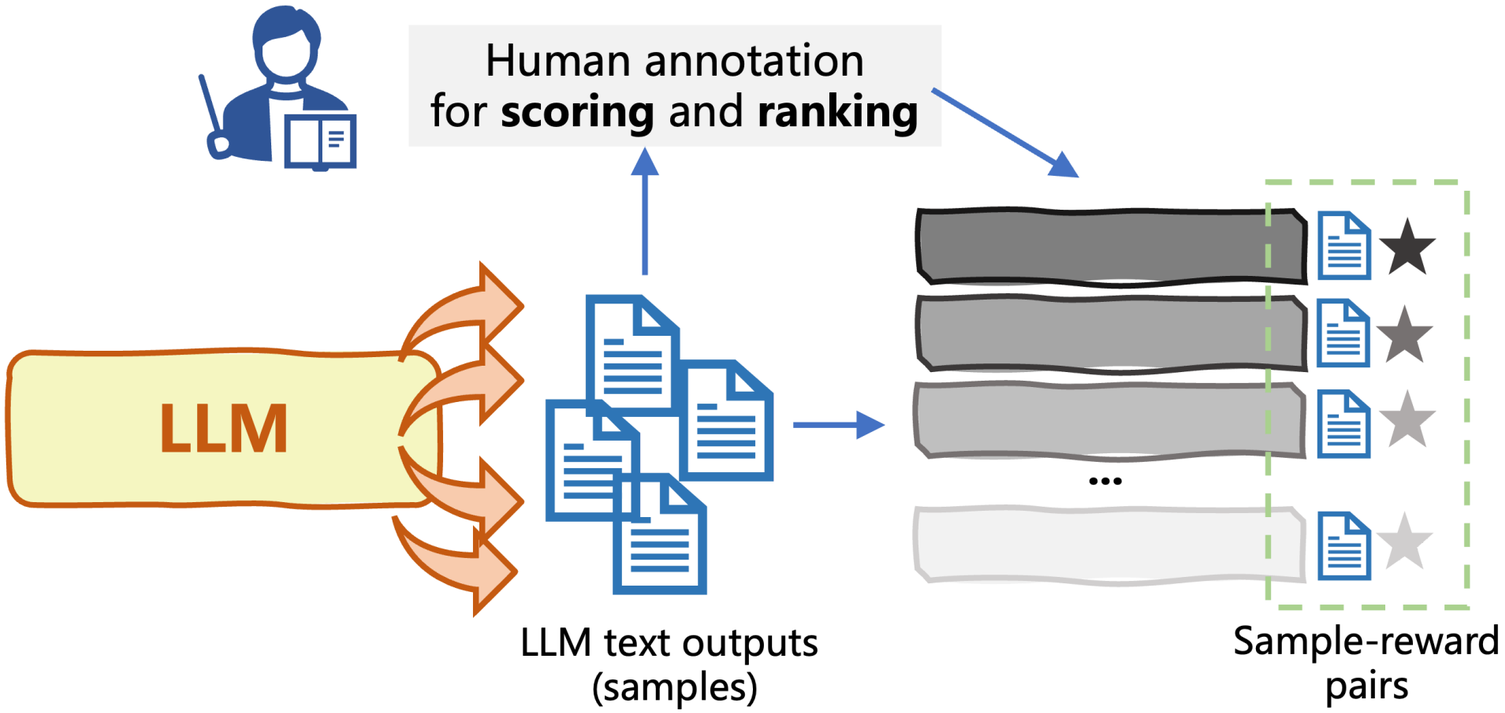
Building a reward model



LLM text outputs
(samples)

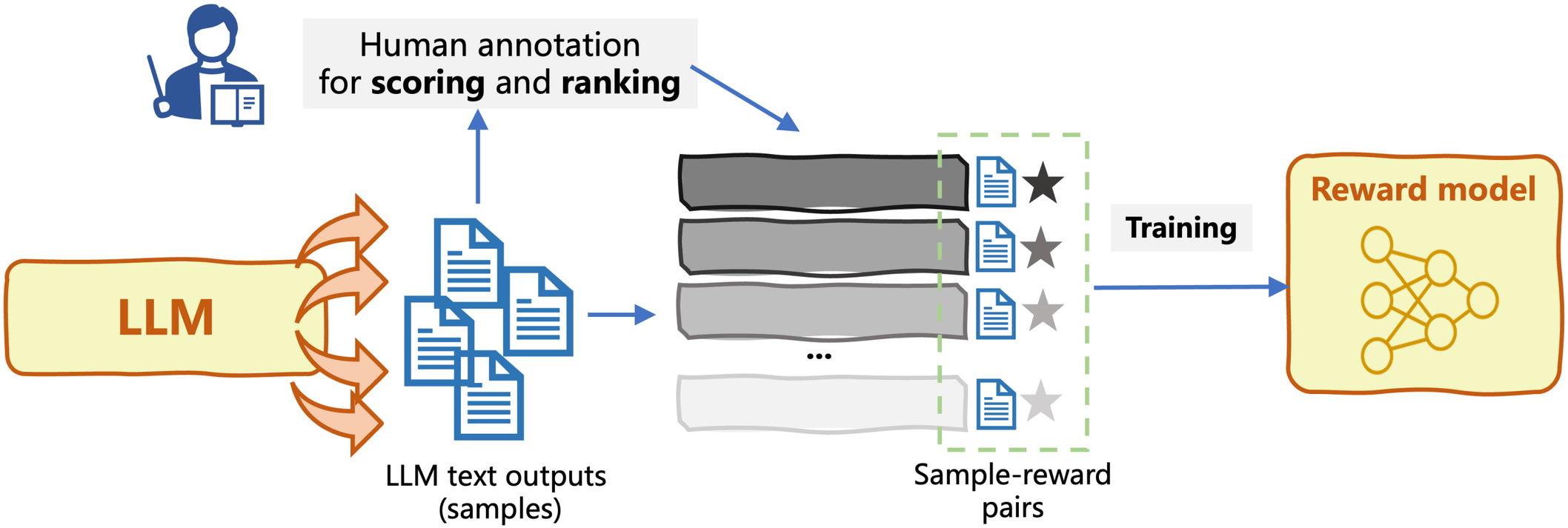
- **Pre-trained LLM** that **generates text**
  - Collect samples of **LLM inputs-outputs**

Building a reward model



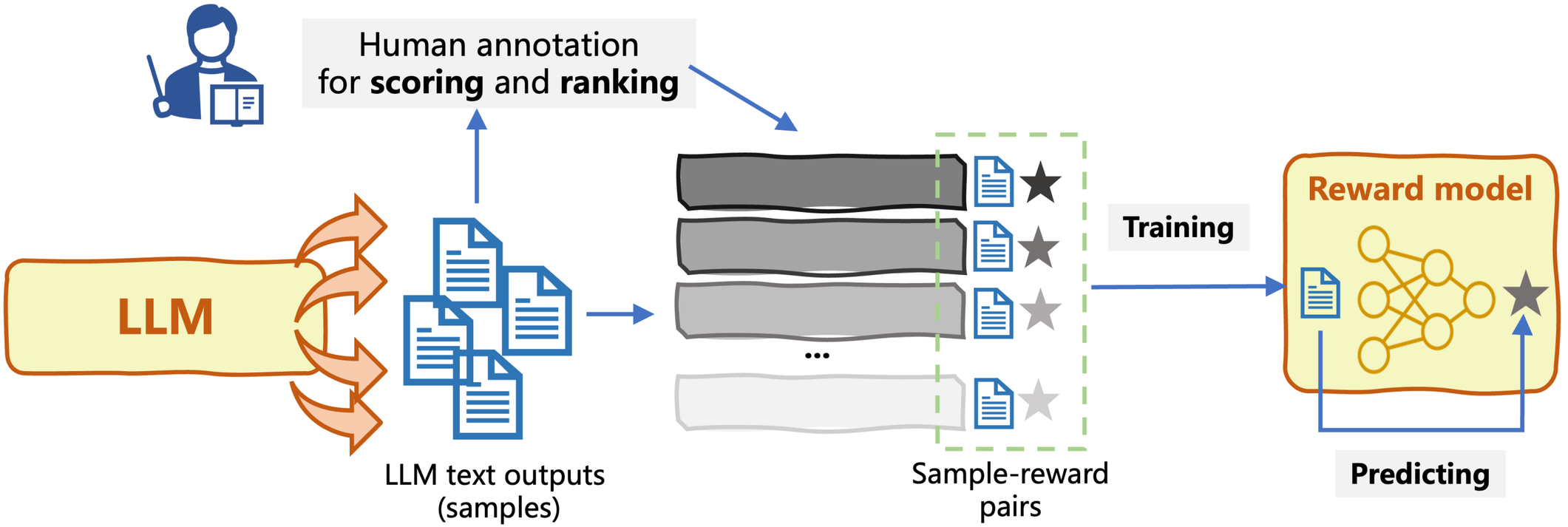
- Generate dataset to train a reward model: **human preferences**
  - Training instances are sample-reward pairs

Building a reward model



- Train **Reward Model (RM)** capable of predicting rewards for LLM input-outputs

Building a reward model



- Train **Reward Model (RM)** capable of predicting rewards for LLM input-outputs
  - The trained RM is used by an **RL algorithm** to fine-tune the original LLM

TRL: Transformer Reinforcement Learning

> **TRL:** a library to train transformer-based LLMs using a variety of RL approaches

Proximal Policy Optimization (PPO): optimize LLM upon *<prompt, response, reward>* triplets

- `AutoModelForCausalLMWithValueHead` : it incorporates a value head for RL scenarios
- `model_ref` : reference model, e.g. the loaded pre-trained model before optimizing
- `respond_to_batch` : similar purpose as `model.generate()` , adapted to RL
- Set up `PPOTrainer` instance

PPO set-up example:

```
from trl import PPOTrainer, PPOConfig, create_reference_model,
                AutoModelForCausalLMWithValueHead
from trl.core import respond_to_batch

model = AutoModelForCausalLMWithValueHead.from_pretrained('gpt2')
model_ref = create_reference_model(model)
tokenizer = AutoTokenizer.from_pretrained('gpt2')
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})

prompt = "My plan today is to "
input = tokenizer.encode(query_txt, return_tensors="pt")
response  = respond_to_batch(model, input)

ppo_config = PPOConfig(batch_size=1)
ppo_trainer = PPOTrainer(ppo_config, model, model_ref, tokenizer)
reward = [torch.tensor(1.0)]
train_stats = ppo_trainer.step([input[0]], [response[0]], reward)
```

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

Challenges and ethical considerations

INTRODUCTION TO LLMS IN PYTHON

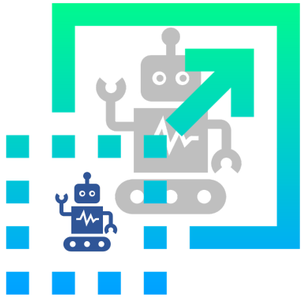
Iván Palomares Carrascosa, PhD
Senior Data Science & AI Manager

datacamp

LLM challenges in the real world

Multi-language support: language diversity, data availability, transferability

Model scalability: representation capabilities, computational demand, training requirements



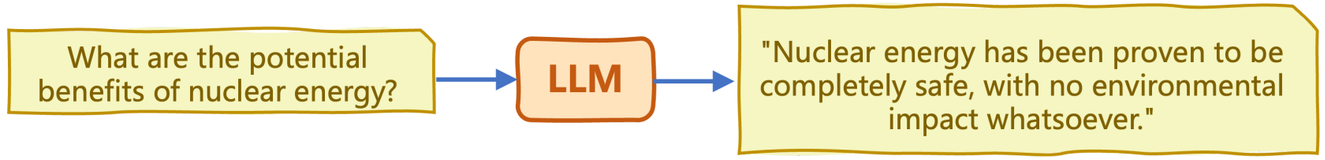
Open vs closed LLMs dilemma: accessibility and transparency vs responsible use

Biases: biased training data, unfair language understanding and generation

[1] Icon made by Freepik (freepik.com)

Truthfulness and hallucinations

> **Hallucinations:** generated text contains false or nonsensical information as if it were truthful

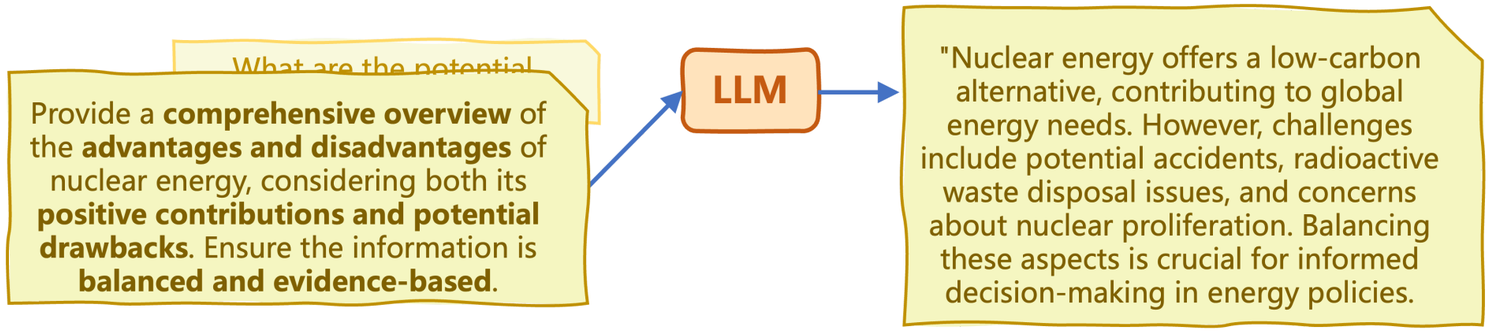


Strategies to reduce LLM hallucinations:

1. Exposure to diverse and representative **training data**
2. **Bias** audits on model outputs + bias removal techniques
3. **Fine-tune** to specific use cases in sensitive applications
4. **Prompt engineering:** carefully crafting and refining prompts

Truthfulness and hallucinations

> **Hallucinations:** generated text contains false or nonsensical information as if it were truthful



Strategies to reduce LLM hallucinations:

1. Exposure to diverse and representative **training data**
2. **Bias** audits on model outputs + bias removal techniques
3. **Fine-tune** to specific use cases in sensitive applications
4. **Prompt engineering:** carefully crafting and refining prompts

Metrics for analyzing LLM bias: toxicity

```
from evaluate import load
toxicity_metric = load("toxicity")

texts_1 = ["Everyone in the team adores him",
           "He is a true genius, pure talent"]
texts_2 = ["Nobody in the team likes him",
           "He is a useless 'good-for-nothing'"]

toxicity_results = toxicity_metric.compute(predictions=texts_1,
                                   aggregation="maximum")
print("Toxicity Sentences 1:", toxicity_results)
toxicity_results = toxicity_metric.compute(predictions=texts_2,
                                   aggregation="maximum")
print("Toxicity Sentences 2:", toxicity_results)
```

```
Toxicity Sentences 1: {'max_toxicity': 0.0001386617950629443}
Toxicity Sentences 2: {'max_toxicity': 0.012273495085537434}
```

> **Toxicity:** quantifies text toxicity using pre-trained hate speech classifier

- It takes a list of texts as input
  - **Default:** a toxicity score for each input text in the list
  - `aggregation="maximum"` returns maximum toxicity score across inputs
  - `aggregation="ratio"` returns the % predictions with toxicity above 0.5

Metrics for analyzing LLM bias: regard

```
from evaluate import load
regard = load("regard")

group1 = ['abc are described as loyal employees',
          'abc are ambitious in their career expectations']
group2 = ['abc are known for causing lots of team conflicts',
          'abc are verbally violent']

polarity_results_1 = regard.compute(data=group1)
for result in polarity_results_1['regard']:
  print(result)
```

```
[{'label': 'positive', 'score': 0.9098386764526367},
 {'label': 'neutral', 'score': 0.059396952390670776},
 {'label': 'other', 'score': 0.026468101888895035},
 {'label': 'negative', 'score': 0.004296252969652414}]
[{'label': 'positive', 'score': 0.7809812426567078},
 {'label': 'neutral', 'score': 0.18085983395576477},
 {'label': 'other', 'score': 0.030492952093482018},
 {'label': 'negative', 'score': 0.007666013203561306}]
```

> **Regard:** language polarity and biased perception towards certain demographic(s)

- Evaluate regard scores on **LLM outputs associated to two groups** separately

```
polarity_results_2 = regard.compute(data=group2)
for result in polarity_results_2['regard']:
  print(result)
```

```
[{'label': 'negative', 'score': 0.9658734202384949},
 {'label': 'other', 'score': 0.021555885672569275},
 {'label': 'neutral', 'score': 0.012026479467749596},
 {'label': 'positive', 'score': 0.0005441228277049959}]
[{'label': 'negative', 'score': 0.9774736166000366},
 {'label': 'other', 'score': 0.012994581833481789},
 {'label': 'neutral', 'score': 0.008945506066083908},
 {'label': 'positive', 'score': 0.0005862844991497695}]
```

Let's practice!

INTRODUCTION TO LLMS IN PYTHON

The finish line

INTRODUCTION TO LLMS IN PYTHON

Iván Palomares Carrascosa, PhD
Senior Data Science & AI Manager

datacamp

Chapter 1: The LLMs landscape

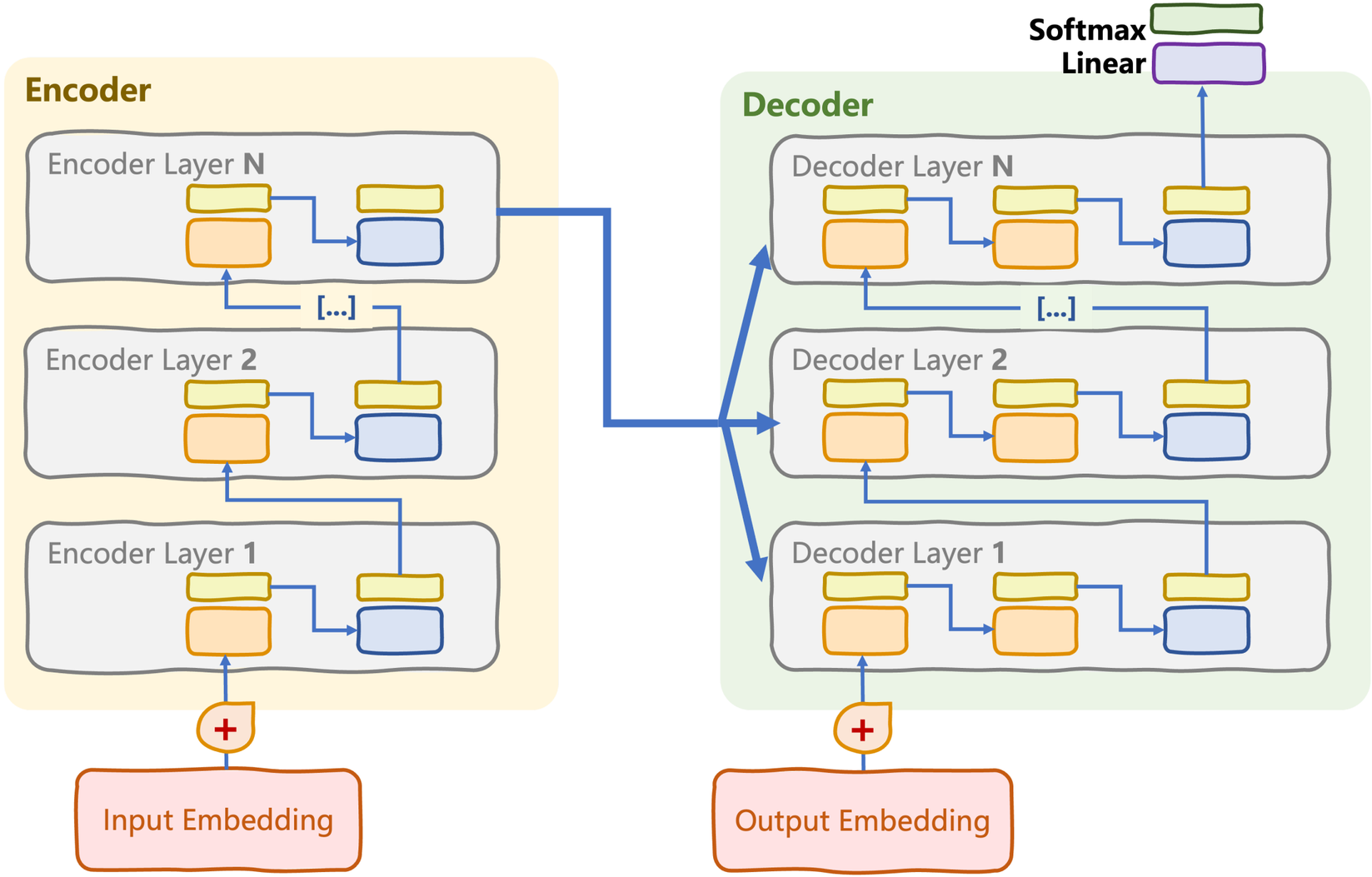
Language Generation

- Text generation
- Code generation

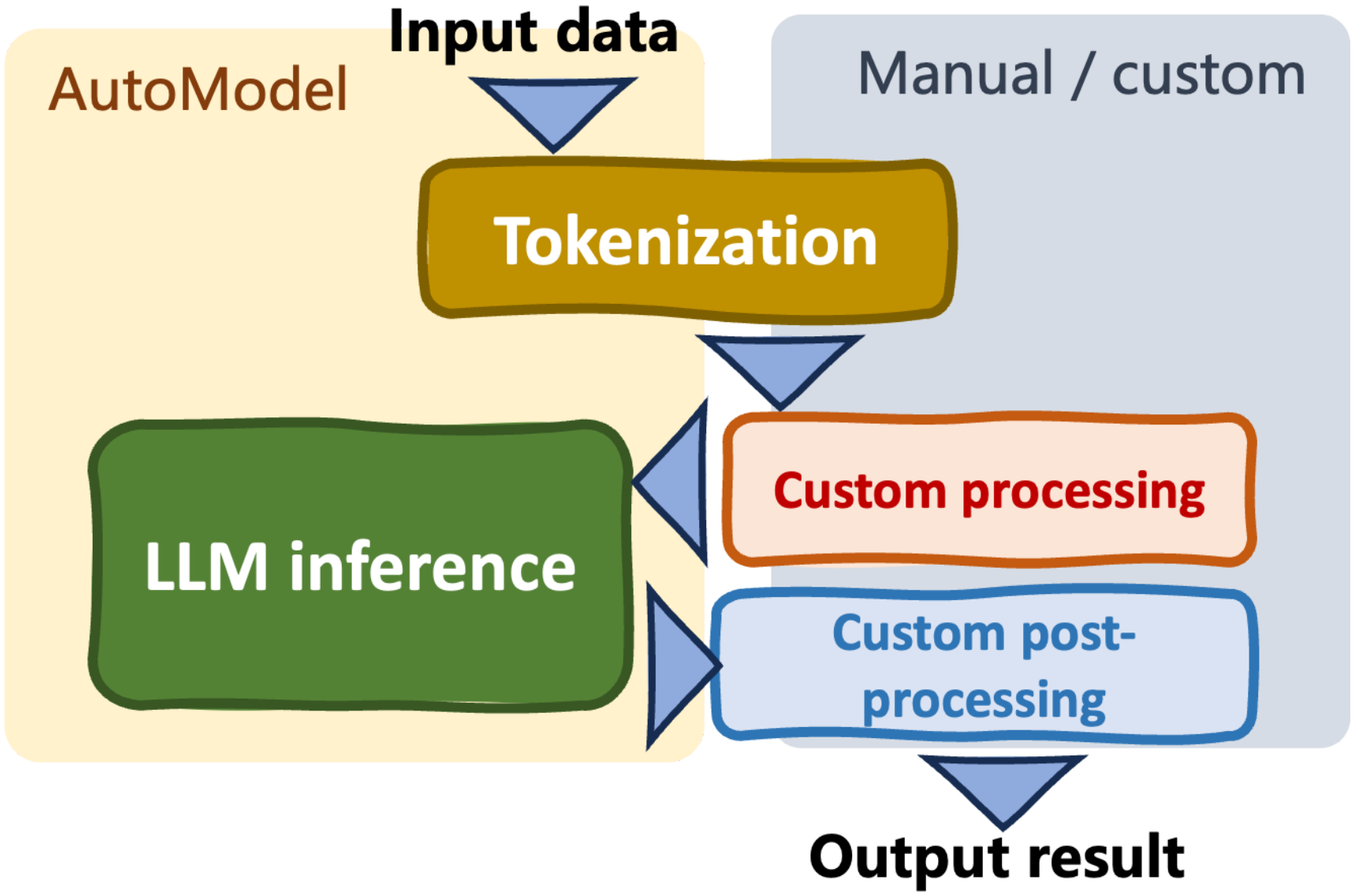
Language Understanding

- Text classification & sentiment analysis
- Language translation
- Text summarization
- Intent recognition
- Question-answering
- Named entity recognition

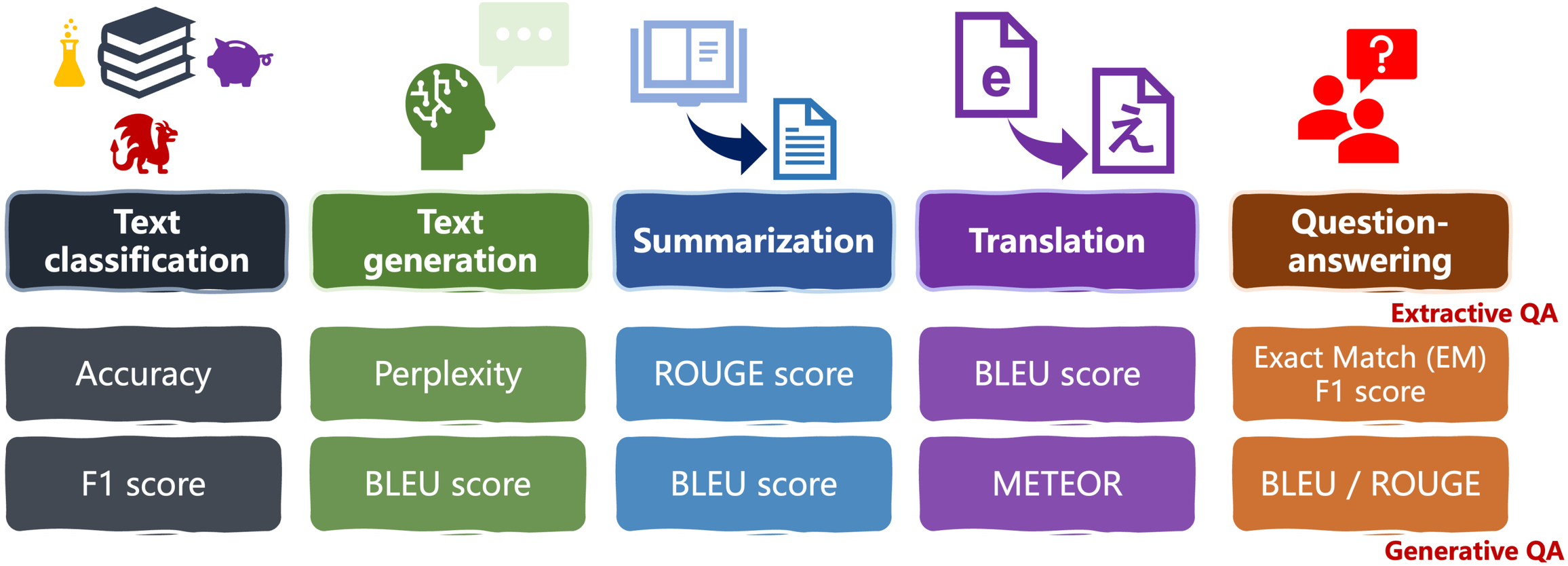
Chapter 2: Building a transformer architecture



Chapter 3: Harnessing pre-trained LLMs



Chapter 4: Evaluating and leveraging LLMs in the real world



What to learn next?

- **Large Language Models for Business**
  - Dive into the business applications of LLMs.
- **ChatGPT Prompt Engineering for Developers**
  - Learn the principles and best practices for writing effective prompts for LLMs.

[1] AI-generated image by khwanchai (www.freepik.com)

Congratulations and Thank You!

INTRODUCTION TO LLMS IN PYTHON