



06/02/2023

# Déployer des modèles IA plus rapide que la lumière

— Présentation d'outils pour optimiser le déploiement de  
modèles IA

# 00 Déployer des modèles IA plus rapide que la lumière

## — Sommaire

01 Qu'est ce qu'un modèle IA ?

02 Présentation des différents outils

03 Démo time

04 Conclusion

# 00 Qui suis-je ? — Rémi Calizzano

- **Nouveau Wewe** 🎉
- **Précédente expérience :**
  - Recherche dans le domaine  
Natural Language  
Processing
  - Gestion de l'infra European  
Language Grid

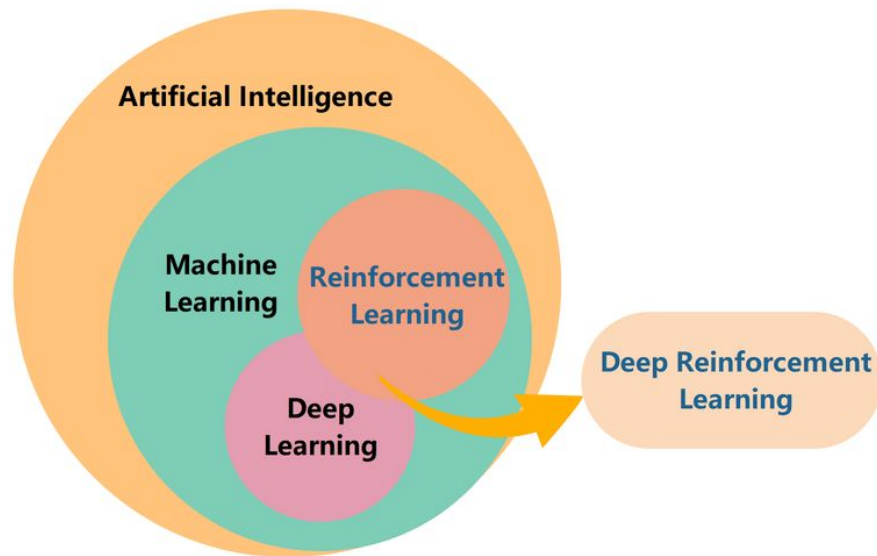


# 01

**Qu'est ce qu'un  
modèle IA ?**

# 01 Qu'est ce qu'un modèle IA ?

## — D'abord, qu'est ce que l'IA ?

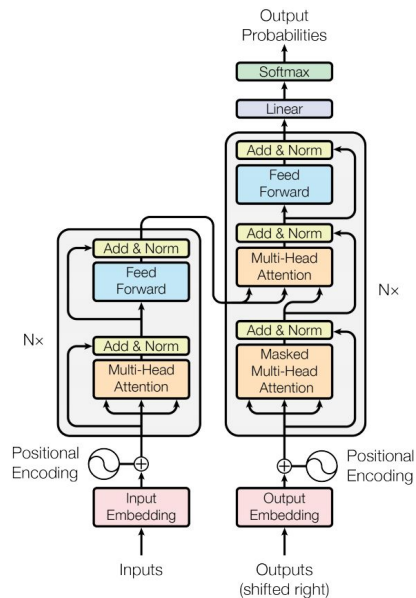


- **AI** = qui simule l'intelligence humaine
- **ML** = qui peut apprendre tout seul
- **DL** = qui utilise des réseaux de neurones et nécessite beaucoup de données

01

Qu'est ce qu'un modèle IA ?

# — Un modèle Deep Learning c'est :



→ Une architecture

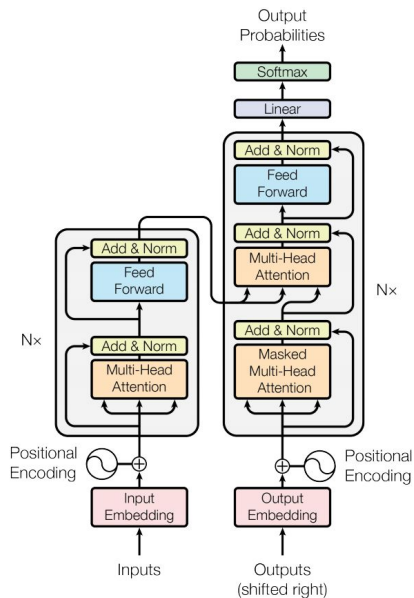
→ Des poids

pytorch_model-00001-of-00005.bin	pickle	9.45 GB	LFS	↓
pytorch_model-00002-of-00005.bin	pickle	9.6 GB	LFS	↓
pytorch_model-00003-of-00005.bin	pickle	9.96 GB	LFS	↓
pytorch_model-00004-of-00005.bin	pickle	10 GB	LFS	↓
pytorch_model-00005-of-00005.bin	pickle	6.06 GB	LFS	↓

01

Qu'est ce qu'un modèle IA ?

# — Un modèle Deep Learning C'est :



➔ Une architecture

```
class BartEncoderLayer(nn.Module):
    def __init__(self, config: BartConfig):
        super().__init__()
        self.embed_dim = config.d_model
        self.self_attn = BartAttention(
            embed_dim=self.embed_dim,
            num_heads=config.encoder_attention_heads,
            dropout=config.dropout,
            batch_first=True,
            device=config.device)
        self.self_attn_layer_norm = nn.LayerNorm(self.embed_dim)
        self.dropout = config.dropout
        self.activation_fn = ACT2FN[config.activation_function]
        self.activation_dropout = config.activation_dropout
        self.fc1 = nn.Linear(self.embed_dim, config.encoder_ffn_dim)
        self.fc2 = nn.Linear(config.encoder_ffn_dim, self.embed_dim)
        self.final_layer_norm = nn.LayerNorm(self.embed_dim)
```

```
def forward(
    self,
    hidden_states: torch.FloatTensor,
    attention_mask: torch.FloatTensor,
    layer_head_mask: torch.FloatTensor,
    output_attentions: Optional[bool] = False,
) -> Tuple[torch.FloatTensor, Optional[torch.FloatTensor]]:
    residual = hidden_states
    hidden_states, attn_weights, _ = self.self_attn(
        hidden_states=hidden_states,
        attention_mask=attention_mask,
        layer_head_mask=layer_head_mask,
        output_attentions=output_attentions,
    )
    hidden_states = nn.functional.dropout(hidden_states, p=self.dropout, training=self.training)
    hidden_states = residual + hidden_states
    hidden_states = self.self_attn_layer_norm(hidden_states)

    residual = hidden_states
    hidden_states = self.activation_fn(self.fc1(hidden_states))
    hidden_states = nn.functional.dropout(hidden_states, p=self.activation_dropout, training=self.training)
    hidden_states = nn.functional.dropout(hidden_states, p=self.dropout, training=self.training)
    hidden_states = residual + hidden_states
    hidden_states = self.final_layer_norm(hidden_states)

    if hidden_states.dtype == torch.float16 and (
        torch.isinf(hidden_states).any() or torch.isnan(hidden_states).any()
    ):
        clamp_value = torch.finfo(hidden_states.dtype).max - 1000
        hidden_states = torch.clamp(hidden_states, min=-clamp_value, max=clamp_value)

    outputs = (hidden_states,)

    if output_attentions:
        outputs += (attn_weights,)

    return outputs
```

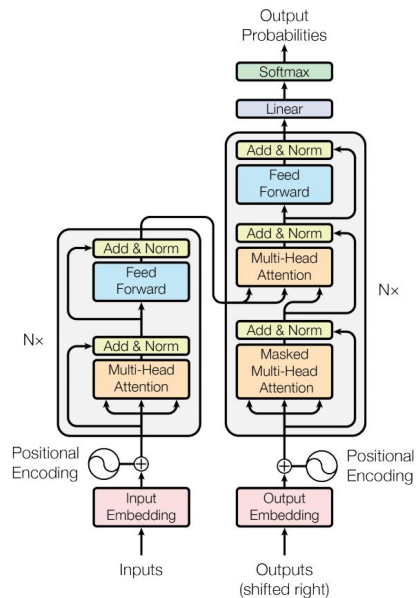




01

Qu'est ce qu'un modèle IA ?

# — Un modèle Deep Learning c'est :


















→ Une architecture

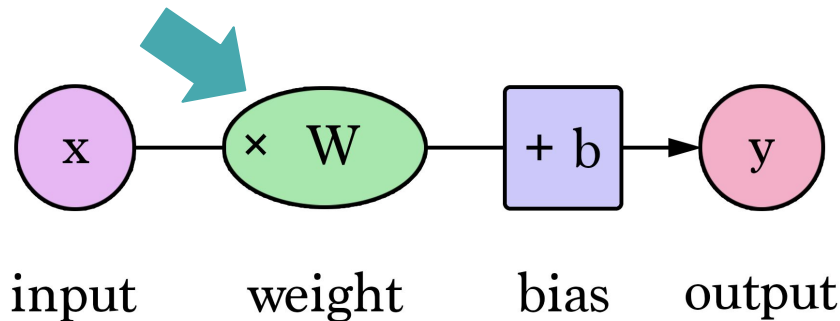
→ Des poids

pytorch_model-00001-of-00005.bin	pickle	9.45 GB	LFS	↓
pytorch_model-00002-of-00005.bin	pickle	9.6 GB	LFS	↓
pytorch_model-00003-of-00005.bin	pickle	9.96 GB	LFS	↓
pytorch_model-00004-of-00005.bin	pickle	10 GB	LFS	↓
pytorch_model-00005-of-00005.bin	pickle	6.06 GB	LFS	↓

01 Qu'est ce qu'un modèle IA ?

# — Un modèle Deep Learning c'est :
















 pytorch_model-00001-of-00005.bin	 pickle	9.45 GB	 LFS	↓
 pytorch_model-00002-of-00005.bin	 pickle	9.6 GB	 LFS	↓
 pytorch_model-00003-of-00005.bin	 pickle	9.96 GB	 LFS	↓
 pytorch_model-00004-of-00005.bin	 pickle	10 GB	 LFS	↓
 pytorch_model-00005-of-00005.bin	 pickle	6.06 GB	 LFS	↓

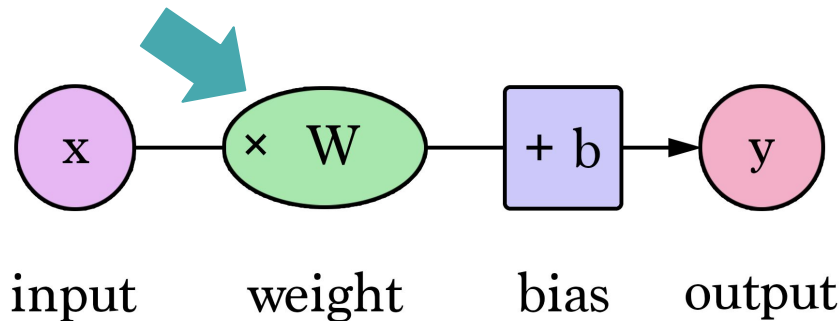


→ Des poids

01 Qu'est ce qu'un modèle IA ?

## — Un modèle Deep Learning c'est :

 pytorch_model-00001-of-00005.bin	 pickle	9.45 GB	 LFS	↓
 pytorch_model-00002-of-00005.bin	 pickle	9.6 GB	 LFS	↓
 pytorch_model-00003-of-00005.bin	 pickle	9.96 GB	 LFS	↓
 pytorch_model-00004-of-00005.bin	 pickle	10 GB	 LFS	↓
 pytorch_model-00005-of-00005.bin	 pickle	6.06 GB	 LFS	↓



→ Des poids

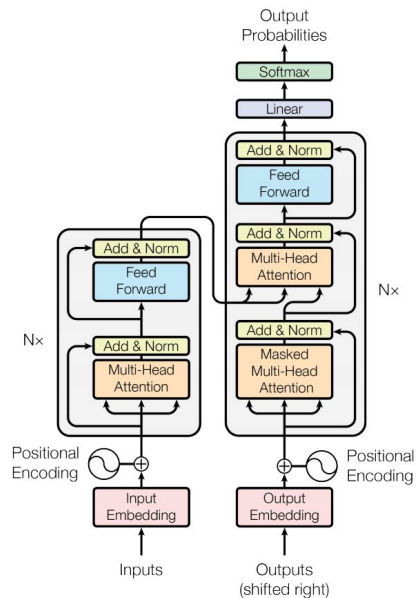
.bin      .msgpack

.safetensors      .h5

01

Qu'est ce qu'un modèle IA ?

# — Un modèle Deep Learning c'est :



→ Une architecture

→ Des poids

pytorch_model-00001-of-00005.bin	pickle	9.45 GB	LFS	↓
pytorch_model-00002-of-00005.bin	pickle	9.6 GB	LFS	↓
pytorch_model-00003-of-00005.bin	pickle	9.96 GB	LFS	↓
pytorch_model-00004-of-00005.bin	pickle	10 GB	LFS	↓
pytorch_model-00005-of-00005.bin	pickle	6.06 GB	LFS	↓

# 02

## **Présentation des différents outils**

# 02 Présentation des différents outils

## — Hugging Face



**“The AI community building the future.** Build, train and deploy state of the art models powered by the reference open source in machine learning.”

- Licorne franco-américaine (~200 employees)
- Open source
- Des outils pour créer, stocker, partager, utiliser des modèles IA
- Groupe de recherche



50. huggingface

★ 149075

# — Hugging Face

**transformers** Public

Transformers: State-of-the-art Machine Learning for Pytorch, TensorFlow, and JAX.

Python

★ 78.7k

🔗 17.7k

**datasets** Public

The largest hub of ready-to-use datasets for ML models with fast, easy-to-use and efficient data manipulation tools

Python

★ 15.1k

🔗 2k

**diffusers** Public

Diffusers: State-of-the-art diffusion models for image and audio generation in PyTorch

Python

★ 9.7k

🔗 1.8k

**accelerate** Public

A simple way to train and use PyTorch models with multi-GPU, TPU, mixed-precision

Python

★ 3.6k

🔗 308

**evaluate** Public

Evaluate: A library for easily evaluating machine learning models and datasets.

Python

★ 1.1k

🔗 96

**optimum** Public

Accelerate training and inference of 🤖 Transformers with easy to use hardware optimization tools

Python

★ 887

🔗 120

# 02 Présentation des différents outils

## — Hugging Face



**Petite démo du Hub**



# 02 Présentation des différents outils

## — Hugging Face



### Transformers

(Une architecture)

- Librairie Python
- Définition des architectures des modèles
- API communes
- Entraînement, utilisation des modèles

### Hub

(Des poids)

- Server Git
- Stockage des poids

```
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
MODEL = "facebook/bart-large-mnli" # address of the model in the Hub

tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

# 02 — Présentation des différents outils

## — Le scénario

facebook/**bart-large-mnli** like 306

Zero-Shot Classification PyTorch JAX Rust Safetensors Transformers multi\_nli arxiv:1910.13461

arxiv:1909.00161 bart text-classification License: mit

Model card Files Community 9

**bart-large-mnli**

This is the checkpoint for [bart-large](#) after being trained on the [MultiNLI \(MNLI\)](#) dataset.

Additional information about this model:

- The [bart-large](#) model page
- [BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension](#)
- [BART fairseq implementation](#)

**NLI-based Zero Shot Text Classification**

[Yin et al.](#) proposed a method for using pre-trained NLI models as a ready-made zero-shot sequence classifiers. The method works by posing the sequence to be classified as the NLI premise and to construct a hypothesis from each candidate label. For example, if we want to evaluate whether a sequence belongs to the class "politics", we could construct a hypothesis of This text is about

Downloads last month  
1,261,816

Hosted inference API ⓘ

Zero-Shot Classification Example 2 ▾

Last week I upgraded my iOS version and ever since then my phone has been overheating whenever I use your app.

Possible class names (comma-separated)

mobile, website, billing, account access

☐ Allow multiple true classes

Compute

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

mobile	0.960
account access	0.017
billing	0.014
	0.000

- Modèle entraîné
- Disponible sur HF Hub

➔ **Objectif :** Mettre ce modèle en production

# 02 — Présentation des différents outils

## Optimum



- Accélérer l'entraînement et l'inférence des modèles
- Permet de s'adapter au hardware pour augmenter l'efficacité
- Optimisation de la séquence d'exécution
- Se base sur d'autres projets
  - ◆ Habana
  - ◆ Graphcore
  - ◆ ONNX
  - ◆ Intel OpenVINO
  - ◆ ...

### Optimum Graphcore

Train transformers on [Graphcore IPUs](#), a completely new kind of massively parallel processor to accelerate machine intelligence.

### Optimum Habana

Maximize training throughput and efficiency with [Habana's Gaudi processor](#).

### Optimum Intel

Use Intel's [Neural Compressor](#) and [OpenVINO](#) frameworks to accelerate transformer inference.

### ONNX Runtime

Apply quantization and graph optimization to accelerate transformer training and inference with [ONNX Runtime](#)

### Torch FX

Create and compose custom graph transformations to optimize PyTorch transformer models with [Torch FX](#)

### BetterTransformer

A one-liner integration to use [PyTorch's BetterTransformer](#) with Transformers models

# 03

## Démo time

Mise en pratique  
d'Optimum lors de la  
mise en production

# — facebook/bart-large-mnli

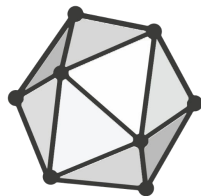


```
from transformers import pipeline
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
sequence_to_classify = "one day I will see the world"
candidate_labels = ['travel', 'cooking', 'dancing']
classifier(sequence_to_classify, candidate_labels)
#{'labels': ['travel', 'dancing', 'cooking'],
# 'scores': [0.9938651323318481, 0.0032737774308770895, 0.002861034357920289],
# 'sequence': 'one day I will see the world'}
```

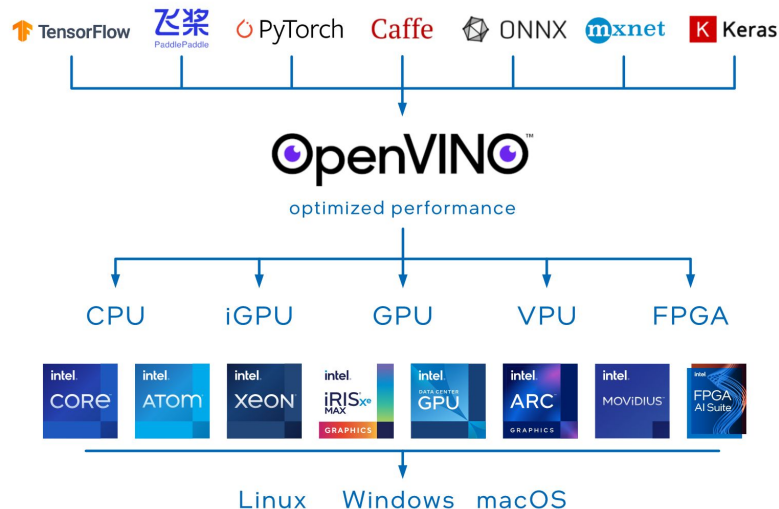
# 03 — Objectif

Démo time

- Optimum facilite l'utilisation d'autres outils
- **Optimiser l'inférence** de notre modèle en utilisant :
  - ◆ **ONNX**
  - ◆ **Intel OpenVINO**
- Comparer les résultats en regardant la **durée d'une requête** sans et avec optimisation
- Comparaison sur **différents hardware**



# ONNX



# 03 — Création de l'API et optimisation

Démo time

## Sans optimisation



```
from fastapi import FastAPI
from transformers import pipeline
from pydantic import BaseModel
from typing import List
```

```
class Request(BaseModel):
    sequence: str
    labels: List[str]
    multi_class: bool = False
```

```
app = FastAPI()
```

```
classifier = pipeline("zero-shot-classification",
model="./bart-large-mnli")
```

```
@app.get("/")
async def health():
    return
```

```
@app.post("/")
async def root(request: Request):
    return classifier(request.sequence, request.labels,
multi_label=request.multi_class)
```

## Optimum ONNX



```
from fastapi import FastAPI
from transformers import pipeline, AutoTokenizer
from optimum.onnxruntime import
ORTModelForSequenceClassification
from pydantic import BaseModel
from typing import List
```

```
class Request(BaseModel):
    sequence: str
    labels: List[str]
    multi_class: bool = False
```

```
app = FastAPI()
```

```
model =
ORTModelForSequenceClassification.from_pretrained("./onnx",
file_name="model.onnx")
tokenizer = AutoTokenizer.from_pretrained("./onnx")
classifier = pipeline("zero-shot-classification",
model=model, tokenizer=tokenizer)
```

```
@app.get("/")
async def health():
    return
```

```
@app.post("/")
async def root(request: Request):
    return classifier(request.sequence, request.labels,
multi_label=request.multi_class)
```

## Optimum Intel OpenVINO



```
from fastapi import FastAPI
from transformers import AutoTokenizer, pipeline
from pydantic import BaseModel
from typing import List
from optimum.intel.openvino import
OVModelForSequenceClassification
```

```
class Request(BaseModel):
    sequence: str
    labels: List[str]
    multi_class: bool = False
```

```
app = FastAPI()
```

```
model =
OVModelForSequenceClassification.from_pretrained("./bart-large-mnli",
from_transformers=True)
tokenizer = AutoTokenizer.from_pretrained("./bart-large-mnli")
classifier = pipeline("zero-shot-classification",
model=model, tokenizer=tokenizer)
```

```
@app.get("/")
async def health():
    return
```

```
@app.post("/")
async def root(request: Request):
    return classifier(request.sequence, request.labels,
multi_label=request.multi_class)
```

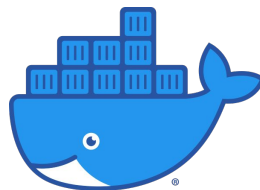
# 03 Démo time — API en local

**Petite démo de l'API en local**






# 03 — Démo time

## — Containerisation

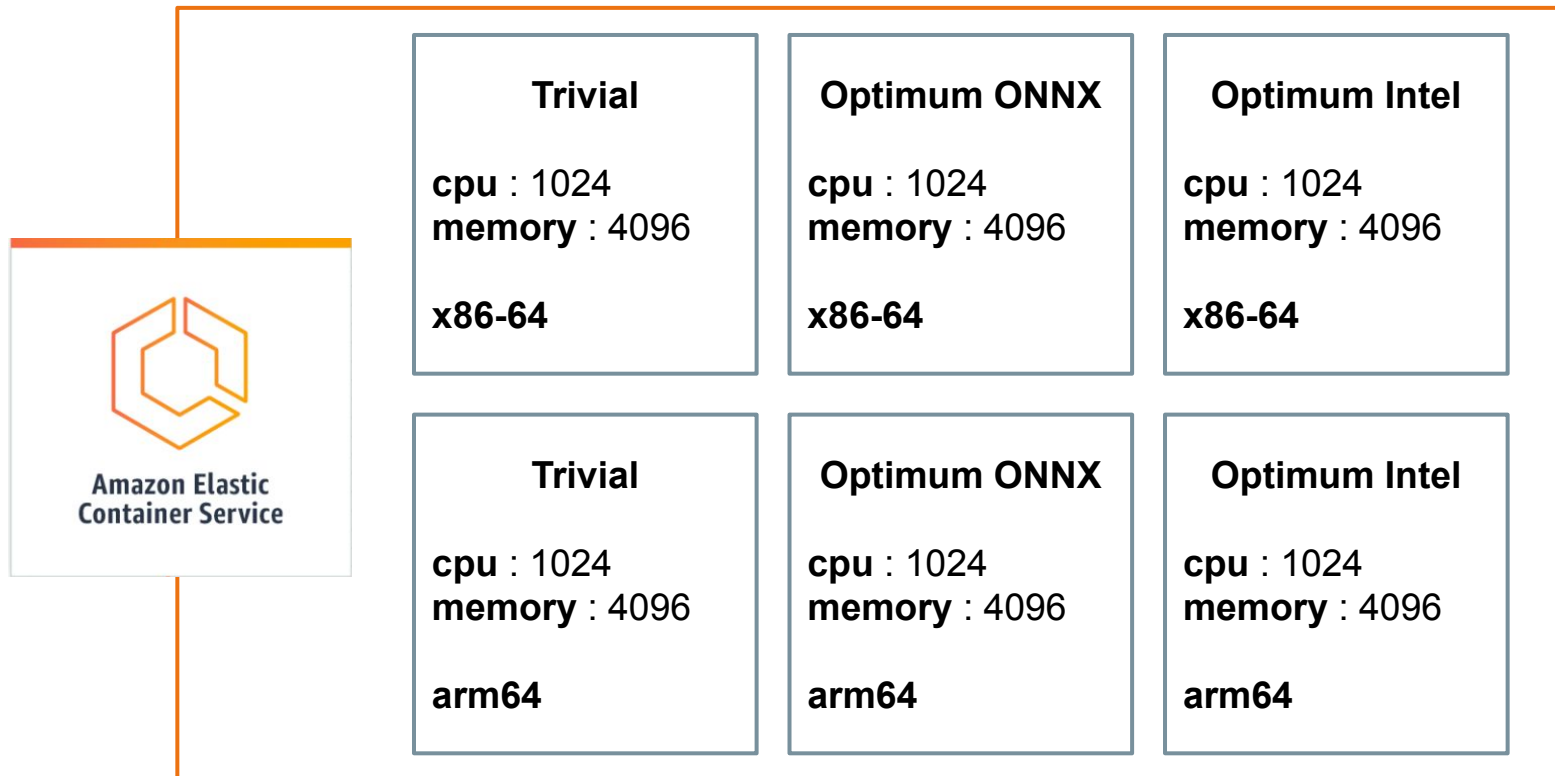


- Containerisation des 3 serveurs http
- Cross platform images (x86-64, arm64)

	<b>airklizz/fastapi-optimum-onnx</b> By <a href="#">airklizz</a> · Updated 3 days ago	<b>169</b> Downloads	<b>0</b> Stars
	Linux x86-64 arm64		
	<b>airklizz/fastapi-trivial</b> By <a href="#">airklizz</a> · Updated 3 days ago	<b>49</b> Downloads	<b>0</b> Stars
	Linux x86-64 arm64		
	<b>airklizz/fastapi-optimum-intel</b> By <a href="#">airklizz</a> · Updated 3 days ago	<b>21</b> Downloads	<b>0</b> Stars
	Linux x86-64 arm64		

# 03 — Déploiement

Démo time



# 03 — Testing with K6

Démo time



	x86-64	arm64
<b>Trivial</b>	1.1 s	1.55 s
<b>Optimum ONNX</b>	0.46 s	0.74 s
<b>Optimum Intel</b>	1.07 s	1.61 s

**Durée d'une requête suivant l'optimisation et l'architecture du CPU (1 VU pendant 1 min)**

# 03 — Testing with K6

Démo time



	x86-64	arm64
<b>Trivial</b>	2.95 s	4.4 s
<b>Optimum ONNX</b>	1.03 s	2.01 s
<b>Optimum Intel</b>	2.73 s	4.55 s

**Durée d'une requête suivant l'optimisation et l'architecture du CPU (3 VU pendant 1 min)**

# 03 Démo time — Résultats

## Optimum ONNX

**-62%** x86-64

**-53%** arm64

→ Réduction du temps  
d'inférence d'un facteur 2

## Optimum Intel

**-4.4%** x86-64

**+3.6%** arm64

→ Pas de réduction du temps  
d'inférence significative

# 03 Démo time — Résultats

## Disclaimers

- Résultats à prendre avec des pincettes, pas de réelle rigueur scientifique mais donne des ordres de grandeurs
- Intel OpenVINO semble très spécifique au processeur, et je n'ai pas eu le temps de tester sur différents CPU Intel

# 04

## Conclusion

# 04 Conclusion

## — Take aways

- **Optimiser** un modèle IA pour l'inférence c'est facile et augmente significativement les performances
- Il existe **beaucoup de méthodes** pour faire cette optimisation (optimum ou autre)
- Ces méthodes d'optimisation sont **bas niveaux** et dépendent donc **du hardware**. I.e., le choix de la méthode utilisée ne peut pas se faire sans connaître où et comment le modèle va être déployé
- **Optimiser un modèle avant sa mise en production doit être fait via une collaboration entre les data scientifiques et les ops**



# 04 Conclusion

## — Pour aller plus loin

- Le **temps de chargement des poids** peut être un facteur important lors qu'il y a de l'**auto scaling**
  - ◆ Safetensor
- **Optimisation avec perte** avec différentes techniques
  - ◆ Optimum avec Quantization (float32 → float16)

**Merci de votre  
attention.**

