PyTorch* による、インテルの AI 向けアクセラレーションの使い方

インテルの AI ポートフォリオで実現するソフトウェア開発のすすめ

エクセルソフト株式会社

2025/06/26

現在の PyTorch* がサポートする インテルの AI 向けアクセラレーション

- 「XPU」アクセラレーター・デバイス
  - インテルの統合および単体 GPU をサポート
- 「CPU」での bfloat16/FP16 データ型
  - 第 4 世代インテル® Xeon® スケーラブル・プロセッサー以降
- インテル® Core™ Ultra プロセッサーの NPU への推論オフロード
  - OpenVINO™ ツールキットとの連携

多様な AI アプリケーションを、より幅広い状況で有効化できる

この場での「AI」

AI: Artificial Intelligence (人工知能)

- ニューラル・ネットワークに基づいたモデルによって実現される機能
  - 特筆される進歩: 言語モデル (自然言語処理)

あなたはさまざまな知識を吸収した優れた AI アシスタントで
あると聞いています。生成 AI とは何か、説明できますか?

はい、私はさまざまな知識を吸収した優れた AI アシスタントという認識を受け取っています。この表現は、人間と似た方法で情報を処理してレスポンスを生成する「生成AI」の一種、特に「生成モデル」の一部であることを指しています。

生成AIモデルは、トレーニングされている人工ニューラルネットワークです。それらは大規模な量のデータを分析し、一定レベルで理解し、特定のジャンルやコンテキストで模倣的に新しいデータを生成するように学習されます。彼らの処理方法を「生成」と呼び、生成された応答は自動的で人間のように見えることが多いです。

ところが、生成AIモデルは本物の意識を持たず、知性や個人的な経験や感情を持ちません。彼らの行動は、トレーニングされたデータと指示セットに基づいて決定されます。その答えは複雑になりうるが、それらは ...

言語モデルの動作

あなたはさまざまな知識を吸収した優れた AI アシスタントであると聞いています。生成 AI とは何か、説明できますか?

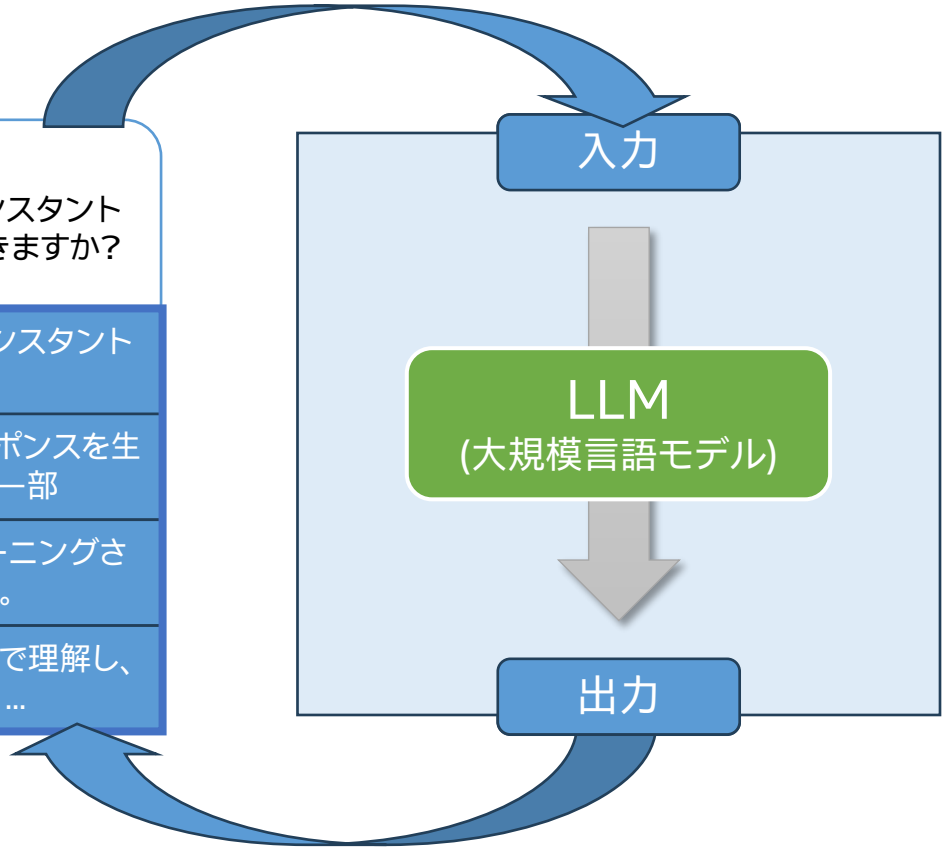
はい、私はさまざまな知識を吸収した優れた AI アシスタントという認識を受け取っています。

この表現は、人間と似た方法で情報を処理してレスポンスを生成する「生成AI」の一種、特に「生成モデル」の一部

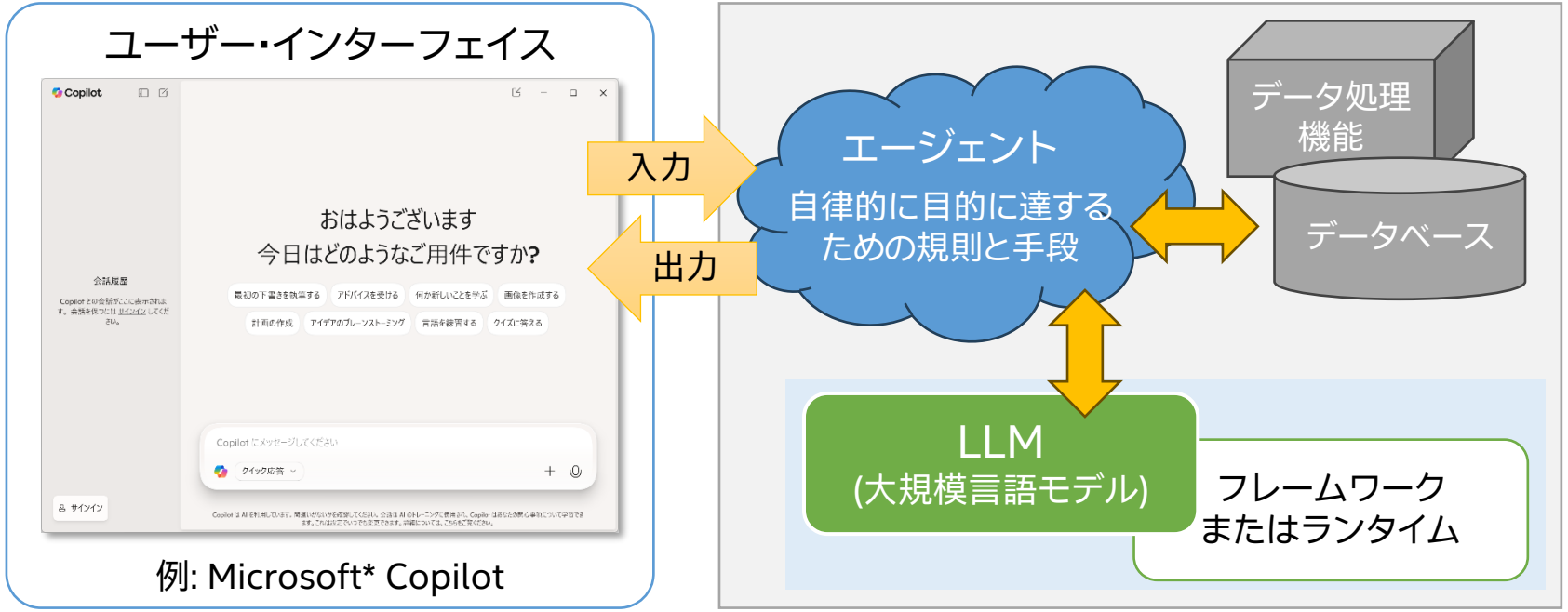
であることを指しています。生成AIモデルは、トレーニングされている人工ニューラルネットワークです。

それらは大規模なデータを分析し、一定レベルで理解し、コンテキストで模倣的に ...

新しい
データを「生成」



AI アプリケーションの構造



PyTorch*

- オープンソース・ソフトウェア (OSS) の 1 つ　https://pytorch.org/ (英語)
  - 任意のニューラル・ネットワーク・モデルとその計算 (訓練および推論) を構成するためのライブラリー (Python*)
    - テンソルの操作、自動微分 (Autograd)
    - アクセラレーター (GPU) を活用した高速な処理、分散並列訓練にも対応
  - PyTorch Foundation (英語) がプロジェクトを管理
    - 元々は Meta (旧称: Facebook) 社が開発し、利用していた OSS
    - コミュニティーに参加する各社によって機能向上の開発が続いている

訓練と推論:「AI」ワークロードの分類

テクノロジーへのオープンアクセス

- HuggingFace による transformers ライブラリー
  - 「Transformer」構造に基づくモデルの PyTorch* から使える実装リスト
    - HuggingFace Hub を通じて訓練済みパラメーター (Pre-trained Weights) の公開、派生モデル開発が盛ん

企業による言語モデルの例 (順不同):

		IBM 「Granite」シリーズ	Alibaba Cloud 「Qwen」シリーズ
Meta 「Llama」シリーズ	Microsoft 「Phi」シリーズ	Google 「Gemma」シリーズ	DeepSeek 「DeepSeek-R1」など

日本語対応モデルに関する参考情報 (外部ページ): https://github.com/llm-jp/awesome-japanese-llm

PyTorch* とインテル

※ oneDNN: oneAPI ディープ・ニューラル・ネットワーク・ライブラリー (旧称 MKL-DNN および DNNL)

- PyTorch* への継続的な貢献
  - CPU 向け最適化のための oneDNN (英語) の統合
  - 「XPU」: インテルの GPU サポート
    - SYCL* による GPU カーネル、OpenAI* Triton* の XPU 対応ランタイム (英語)
- PyTorch* の拡張プロジェクト
  - PyTorch* 向けインテル® エクステンション (IPEX、英語): CPU および XPU
  - インテル® Gaudi® ソフトウェア・スイートへの統合 (英語): HPU

PyTorch* のインストール

- CPU 向け

※ PyTorch* は Python* 用のソフトウェア、Python* 3.9 以降をサポート

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
```

- XPU 対応版 (インテルの GPU を利用)

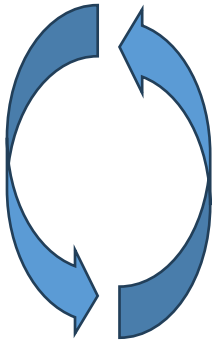
```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/xpu
```

  - GPU 有効化の要件: ドライバーのインストール
    - Windows*: インテル® Arc™ & Iris® Xe Graphics - Windows*
    - Linux* または WSL2: https://dgpu-docs.intel.com/driver/client/overview.html (英語)

PyTorch* による訓練のコード (中核部分)

```
model = get_model(...)     # ニューラル・ネットワーク構造の構成
model.train()
```

```
# 訓練用データセット全体の処理 1 周 (1 epoch)
for batch_idx, (inputs, targets) in enumerate(training_dataloader):
```



```
    optimizer.zero_grad()                          # 更新パラメーターの初期化

    outputs = model(inputs)                        # 順伝播計算 (推論)
    loss = loss_function(outputs, targets)         # 誤差の計算

    loss.backward()                                # 誤差の逆伝播計算
    optimizer.step()                               # パラメーター更新
```

PyTorch* による推論のコード (中核部分)

```
model = get_model(...)     # 訓練済みパラメーターを含むモデルを読込み
model.eval()
```

```
# 推論処理を前提 (パラメーターを更新しない)
with torch.inference_mode():
```

```
outputs = model(inputs)                    # 順伝播計算 (推論)
```

アプリケーション側で inputs テンソル (torch.Tensor) を用意し、
outputs テンソルから結果を取得して解釈する

PyTorch*: アクセラレーター・デバイスの利用

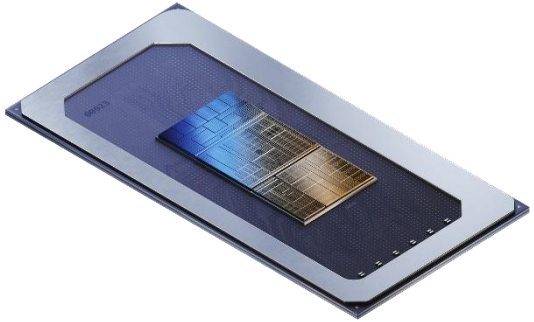
```python
# 実行している PyTorch* に応じたアクセラレーター・デバイス名を得る
if torch.accelerator.is_available():
    device = torch.accelerator.current_accelerator().type
else:
    device = "cpu"
```

今回は "xpu" が得られる想定
有効でないときは "cpu"

```python
# モデルおよび入力テンソルを対応させる
model = model.to(device)
loss_function = loss_function.to(device)

inputs = inputs.to(device)
targets = targets.to(device)
```

XPU デバイス: インテルの GPU 製品



プロセッサー統合グラフィックス

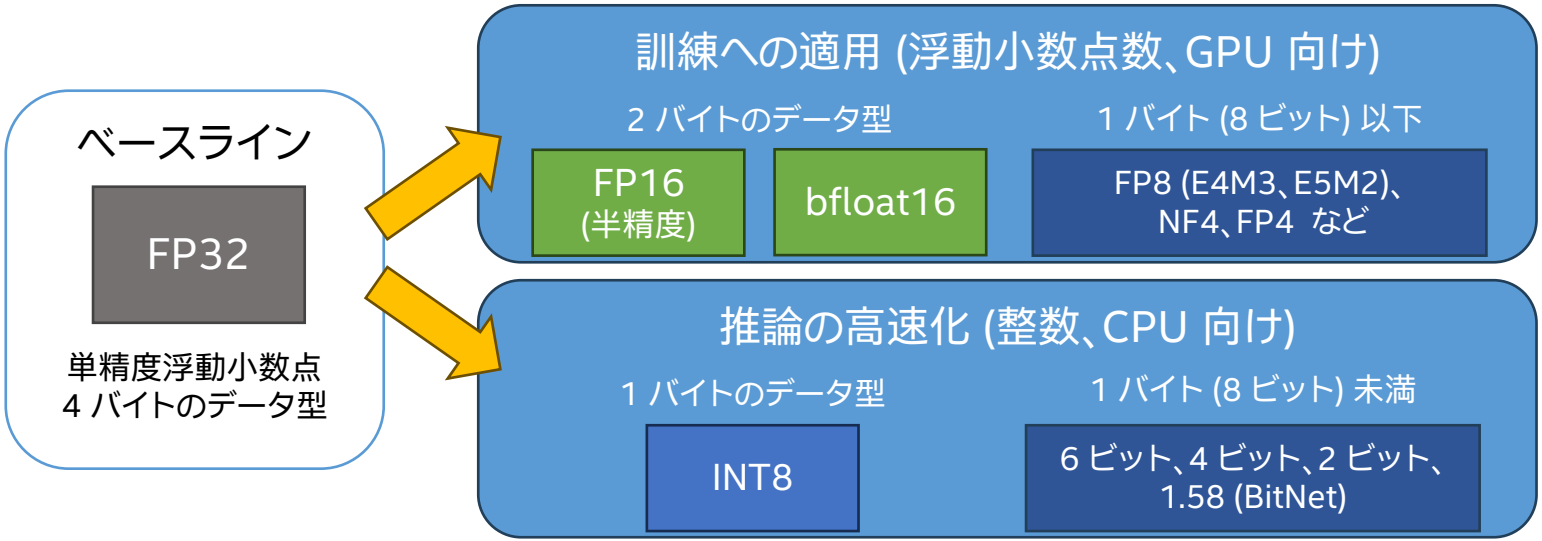
- インテル® Core™ Ultra プロセッサー (シリーズ 1)
  100H (開発コード名 Meteor Lake)
- インテル® Core™ Ultra モバイル・プロセッサー (シリーズ 2)
  200V (開発コード名 Luner Lake)
  200H (開発コード名 Arrow Lake-H)

単体グラフィックス (拡張カード)

- インテル® データセンター GPU マックス・シリーズ
- インテル® Arc™ A シリーズ・グラフィックス
- インテル® Arc™ B シリーズ・グラフィックス

ニューラル･ネットワークと低精度計算

- 生成 AI モデルは 1 B (10 億) を超えるパラメーター数からなる
  - 個々のデータ型は小さいほど計算に有利 (必要メモリー、ストレージ容量を低減)



PyTorch*: AMP (自動混合精度)

- 計算中の実数データ型の精度を下げる (4 バイトから 2 バイトへ)
  - dtype = torch.float16 (半精度) または torch.bfloat16
  - ハードウェアで直接計算ができれば処理能力の向上となる

バイナリーデータ表現の比較

	データサイズ (Byte)		符号	指数部 8							仮数部 10													
浮動小数点	fp32 (float)	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	+ 16 ビット			
	fp16 (half)	2				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				
	bf16 (bfloat16)	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1							

PyTorch*: AMP (自動混合精度) の適用

- torch.autocast ブロックを挿入

```
optimizer.zero_grad()                          # 更新パラメーターの初期化
```

```
with torch.autocast(device, torch.bfloat16, enabled=True):
    outputs = model(inputs)                    # 順伝播計算（推論）
    loss = loss_function(outputs, targets)     # 誤差の計算
```

```
loss.backward()                                # 誤差の逆伝播計算
optimizer.step()                               # パラメーター更新
```

参照: Automatic Mixed Precision package - torch.amp (英語)

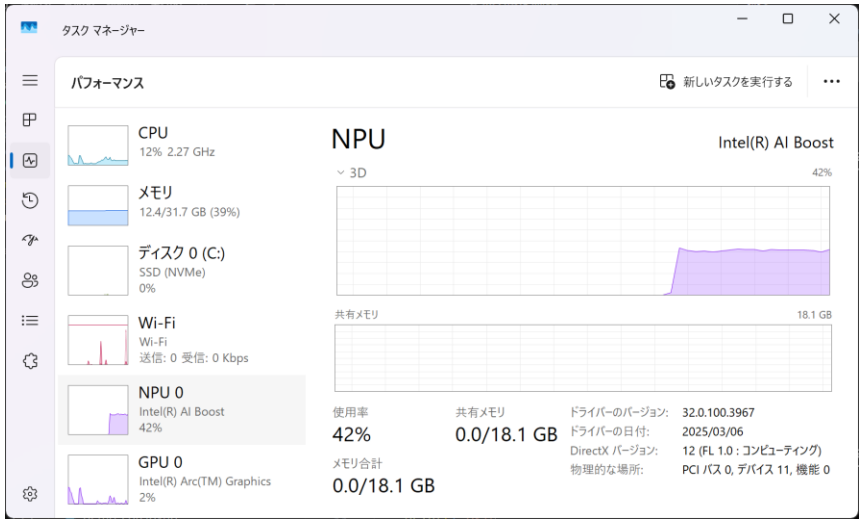
インテルの CPU 製品

- サーバー・プロセッサーでは半精度と bfloat16 の演算をサポート

第 4 世代インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Sapphire Rapids) 第 5 世代インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Emerald Rapids)	P-cores 搭載インテル® Xeon® 6 プロセッサー (開発コード名 Granite Rapids)
インテル® AVX-512 (インテル® アドバンスト・ベクトル・エクステンション 512) INT8/BF16 (bfloat16) AVX512_FP16 (半精度、float16) インテル® AMX (インテル® アドバンスト・マトリクス・エクステンション) INT8/BF16 (bfloat16)	インテル® AVX-512 (インテル® アドバンスト・ベクトル・エクステンション 512) INT8/BF16 (bfloat16) AVX512_FP16 (半精度、float16) インテル® AMX (インテル® アドバンスト・マトリクス・エクステンション) INT8/BF16 (bfloat16) FP16 (半精度、float16)

NPU: ニューラル・プロセシング・ユニット

- エッジデバイス向けに設計される専用アクセラレーター
  - AI の処理において、性能に対する消費電力を重視



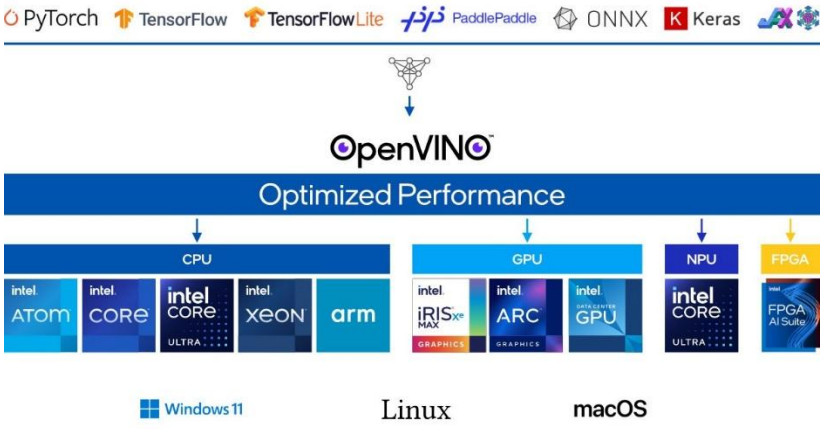
インテルの NPU

- インテル® Core™ Ultra プロセッサーに統合される「インテル® AI Boost」
- 推論処理専用

インテル® AI Boost の有効化にはドライバーのインストールまたは更新が必要
インテル® NPU ドライバー - Windows*

OpenVINO™ ツールキット

訓練済みモデルの展開支援ツール



出典: オンライン・ドキュメント https://docs.openvino.ai/ (英語)

- ✓ さまざまなフレームワークの幅広いディープラーニング・モデルを利用可能
- ✓ モデルを操作する共通の API (C/C++、Python*、Node.js*)
- ✓ インテルによる CPU、GPU や他のデバイス上での効率的な推論実行
- ✓ 無料のツール、オープンソース (英語)、少ないソフトウェア依存関係

https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (英語)

PyTorch* への統合: torch.compile

- PyTorch* 2.0 で導入された新しい JIT コンパイル・インターフェイス
  - 「コンパイル」指示により初回実行時に最適化を施す
- backend="openvino" を明示的に指定
  - device オプションで推論のオフロード先デバイスを選択
    - OpenVINO™ ランタイムが対応する CPU、GPU、NPU

```
model = torch.compile(model, backend="openvino",
    options={"device": "NPU", "aot_autograd": True})
```

実行テスト

注意: 適切な、また最適な手法を選択したものではありません
性能は条件によって異なる場合があります

- resnet-train.py
  - 訓練の処理時間を確認 (AMP: 自動混合精度を含む)
- resnet-inder.py
  - 推論の処理時間を確認 (OpenVINO™ の適用を含む)
- phi-4-mini-instruct.py
  - 言語モデル microsoft/Phi-4-mini-instruct (英語) の実行を確認

確認時の Python* モジュールバージョン:

torch==2.7.1
openvino==2025.2
transformers==4.49.0

スクリプトファイルのダウンロード
https://jp.xlsoft.com/demo2/intel/20250626/pytorch-test.zip

テストの実行環境

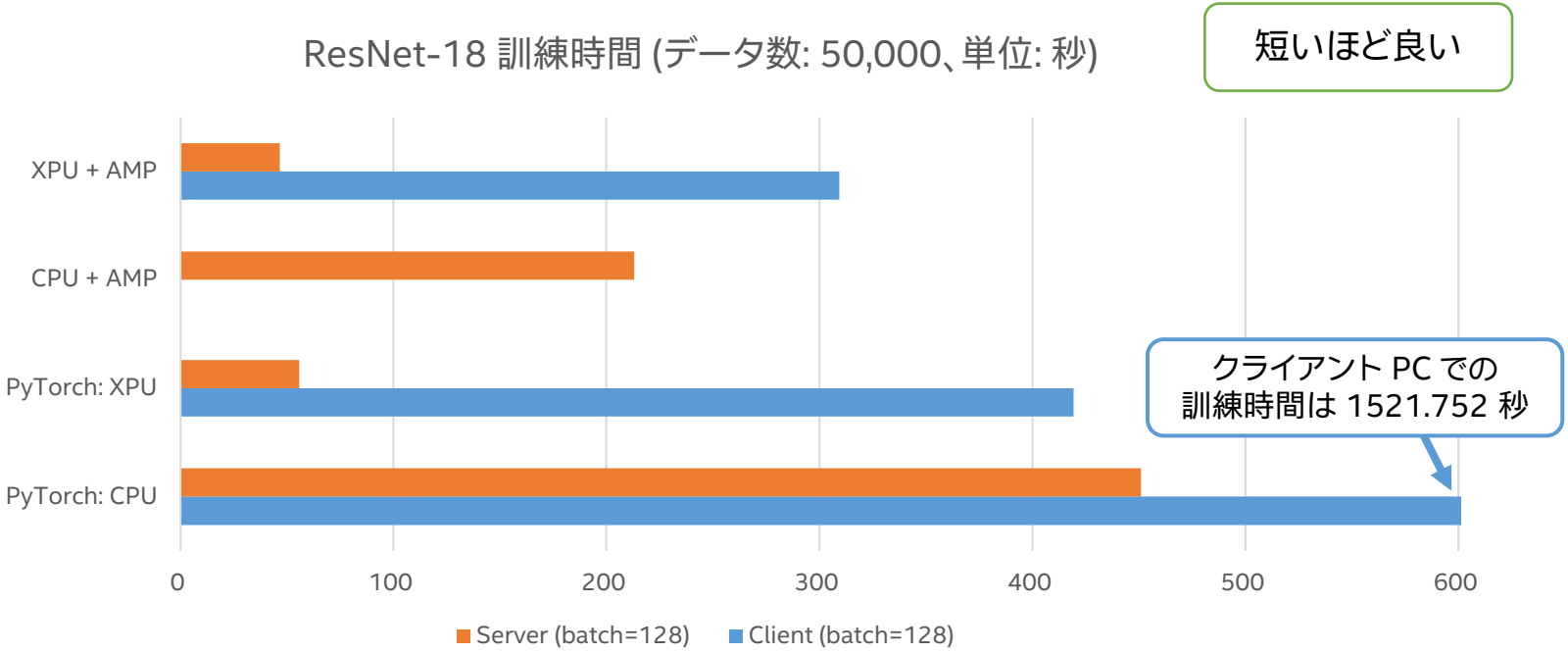
	Linux* サーバーマシン	Windows* クライアント PC
CPU	インテル® Xeon® Platinum 8468V プロセッサー (※)	インテル® Core™ Ultra 7 155H プロセッサー
コア数	48 コア (96 スレッド)	P コア 6、E コア 8、LP E コア 2 (16 コア、22 スレッド)
GPU (XPU)	インテル® データセンター GPU マックス 1100	インテル® Arc™ グラフィックス (プロセッサーに統合)
メモリー	DDR5 4800MT/s、1TB (64GB x16)	DDR5 6400MT/s、32GB (16GB x2)
ソフトウェア	Ubuntu* 22.04.5 LTS (kernel 5.15.0-131-generic) GPU ドライバー: LTS 2350.136 (英語)	Windows* 11 24H2 GPU ドライバー: 32.0.101.6881 (2025/06/04) NPU ドライバー: 32.0.100.4023 (2025/04/23)

※ Linux* サーバーの CPU 向けには、CPU 1 つ分のみを指定するため
「OMP_NUM_THREADS=48 taskset -c 0-47,96-143 python [target].py」として実行

resnet-train.py

- 訓練の処理時間を確認
  - torchvision より画像分類モデル ResNet-18 (英語) を使用
    - 入力形状: バッチサイズ x 3 x 224 x 224
    - 出力形状: バッチサイズ x 10
  - 訓練データの数: 50,000、バッチサイズ: 128、1 epoch のみ
    - 訓練済みパラメーターの保存や結果の検証は省略
  - AMP (自動混合精度) で bfloat16 データ型演算を含めるか否かを設定
    - クライアント PC では非対応の演算となるため実行しない

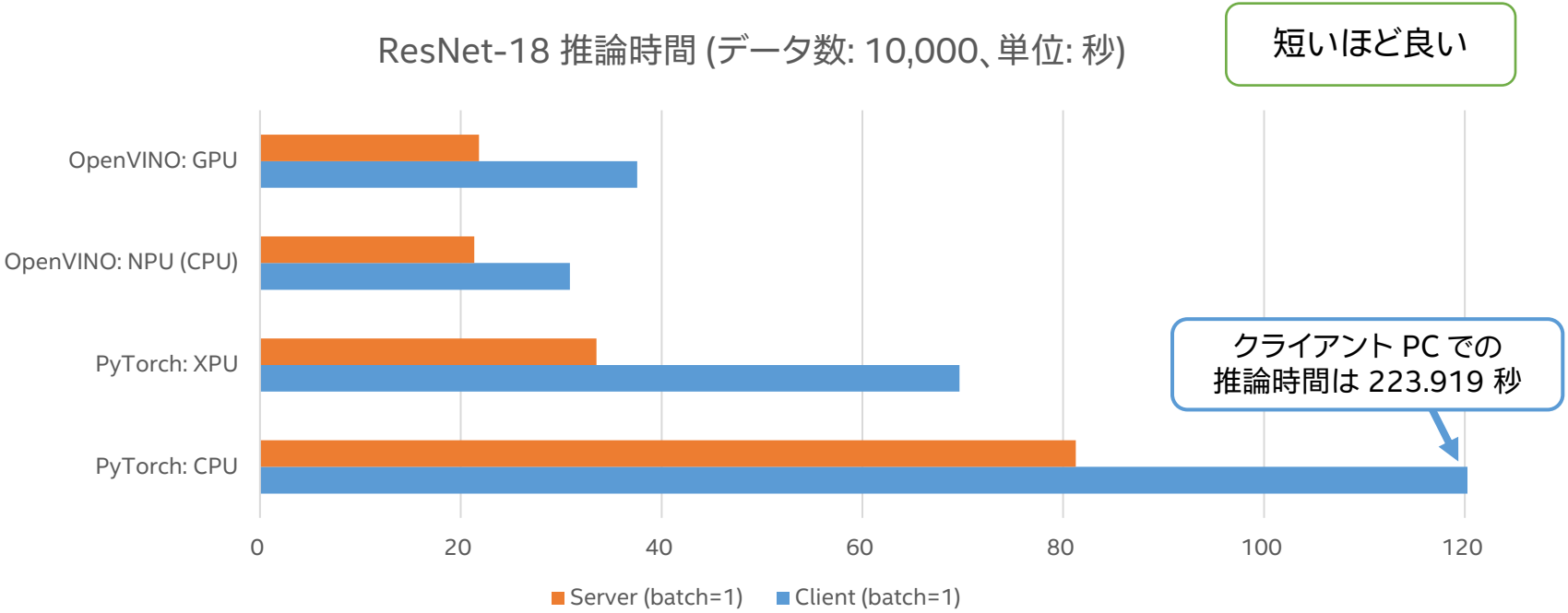
resnet-train.py のパフォーマンス



resnet-infer.py

- 推論の処理時間を確認
  - torchvision より画像分類モデル ResNet-18 (英語) を使用
    - 入力形状: バッチサイズ x 3 x 224 x 224
    - 出力形状: バッチサイズ x 1000
  - 確認データの数: 10,000、バッチサイズ: 1
    - 結果の検証は省略
  - torch.compile(backend="openvino") の有効無効を設定
    - サーバーマシンでは NPU の代わりに CPU を対象とする

resnet-infer.py のパフォーマンス



phi-4-mini-instruct.py

3.84B (38.4 億) パラメーター、7.7GB

- 言語モデル microsoft/Phi-4-mini-instruct (英語) の実行を確認
  - Example (英語) コードを基に拡張したもの、乱数的な振る舞いは抑制

質問 (入力):

等式 2x + 3 = 7 を x について解くにはどのようにすればよいですか?

等式 2x + 3 = 7 を x について解くには、次の手順に従ってください:

1. x を含む項を分離するために、等式の両側から 3 を引きます:
   2x + 3 - 3 = 7 - 3
   2x = 4

2. x を解くために、等式の両側を 2 で割ります:
   2x / 2 = 4 / 2
   x = 2

したがって、x = 2 です。

136 トークン出力

phi-4-mini-instruct.py のパフォーマンス

生成速度 (大きいほど良い)		PyTorch: CPU	PyTorch: XPU
Linux* サーバーマシン	初回	9.064 [トークン/秒]	11.982 [トークン/秒]
	2 回目	8.970 [トークン/秒]	20.341 [トークン/秒]
Windows* クライアント PC	初回	2.723 [トークン/秒]	1.272 [トークン/秒]
	2 回目	3.365 [トークン/秒]	2.429 [トークン/秒]

※ 136 トークン出力に対して

パラメーターのデータ型は
モデルのデフォルトである bfloat16

```
model_path = "microsoft/Phi-4-mini-instruct"

model = AutoModelForCausalLM.from_pretrained(
    model_path,
    device_map="auto",
    torch_dtype="auto",
    trust_remote_code=True,
)
```

現状の制限: プロセッサー統合グラフィックスを使用する場合、
device_map について "auto" とせず、明示的に "xpu" を指定

まとめ

インテルは …

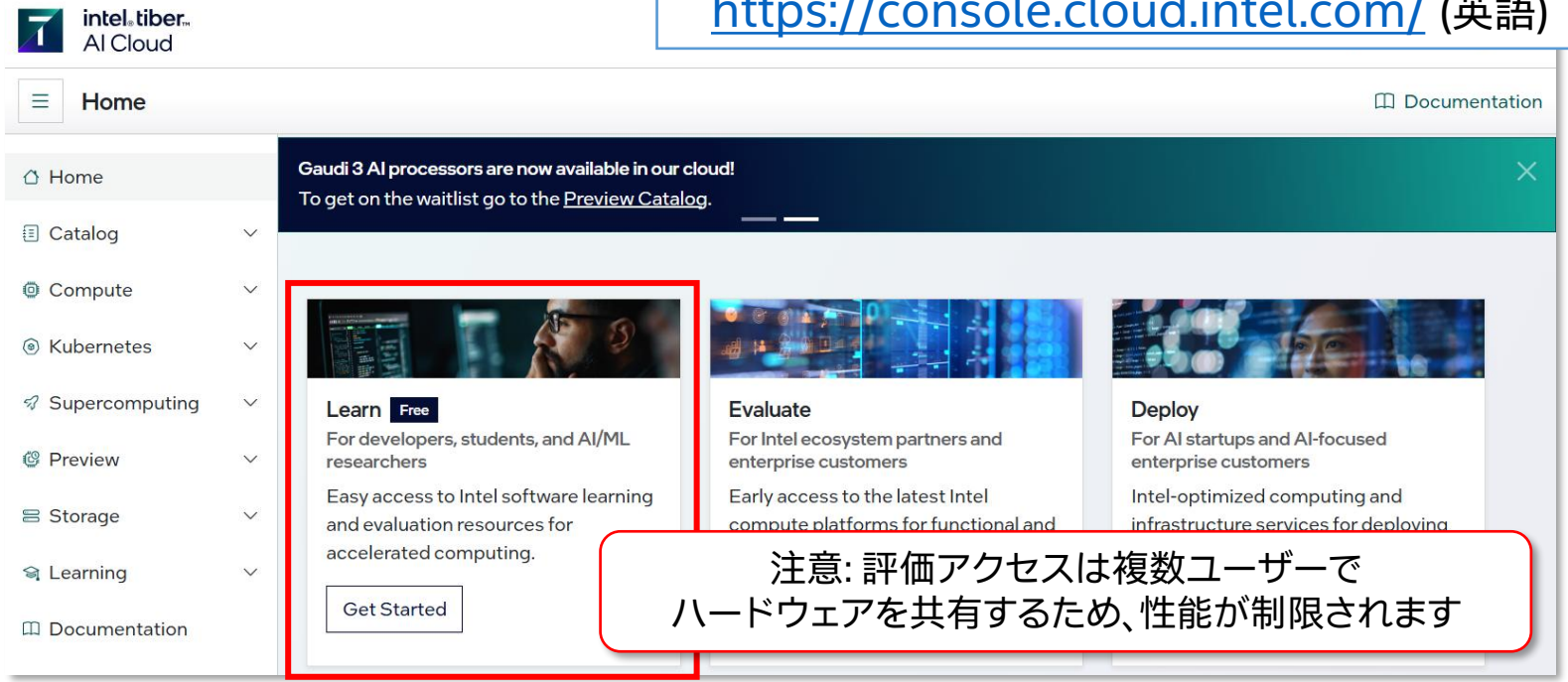
- AI 向けアクセラレーションを最新のプロセッサー製品に統合しています
- 広範な利用と人気のあるフレームワーク、PyTorch* を通じて、
  そうした AI 向けアクセラレーションを有効化しています
- これらの成果は、公開されており、すぐに確認して利用できます

Getting Started on Intel GPU - PyTorch Docs (英語)

PyTorch Deployment via “torch.compile” - OpenVINO Docs (英語)

補足: インテル® Tiber™ AI クラウドで サーバーシステムの評価アクセスを提供中

https://console.cloud.intel.com/ (英語)



ご視聴ありがとうございました

終了後のオンラインセミナー・アンケートへ
ご協力くださいますと幸いです