PyTorch と Open Platform for Enterprise AI (OPEA) を使用して AI アバター・チャットボットを作成する

～インテル® Xeon® スケーラブル・プロセッサーとインテル® Gaudi® AI アクセラレーターで構築および展開する～

この記事は、インテルのウェブサイトで公開されている「Create an AI Avatar Talking Bot with PyTorch* and Open Platform for Enterprise AI (OPEA)」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

概要

今日の AI の世界において、AI アバター・チャットボットはさまざまな業界のビジネスを変革しています。金融機関や教育機関の受付から、空港や病院などの公共施設まで、至る所でデジタルヒューマン AI が顧客の問い合わせに対応し、パーソナライズされたガイダンスを提供しています。

このような企業からのデジタルヒューマン AI に対するニーズに応えるため、AI アバター音声チャットボット (英語)を開発しました。この記事では、最先端の生成 AI (SOTA GenAI) システム向けのオープンでマルチプロバイダー、かつ構成可能なビルディング・ブロックのフレームワークである Open Platform for Enterprise AI (OPEA) (英語) を活用して、インテル® Xeon® スケーラブル・プロセッサーとインテル® Gaudi® AI アクセラレーター上で AI アバター・チャットボットを作成する方法を紹介します。また、インテル® Gaudi® ソフトウェア・スイートや PyTorch (英語) などのインテルの最適化ソフトウェアが、これらの AI ソリューションのトレーニングと推論のパフォーマンス向上にどのように役立つかを説明します。

図 1. Stable Diffusion 3 Medium で生成されたサンプル画像[13]

Open Platform for Enterprise AI (OPEA)

OPEA プラットフォームには、次のものが含まれます。

- LLM、データストア、プロンプトエンジンを含む SOTA GenAI システム用のマイクロサービス・ビルディング・ブロックのフレームワーク
- 検索拡張生成 (RAG) GenAI コンポーネント・スタック構造のアーキテクチャー・ブループリント
- GenAI ソリューションを開発およびデプロイするためのマイクロサービスとメガサービス
- パフォーマンス、機能、信頼性、エンタープライズグレードの準備状況に関する GenAI システムの 4 段階評価

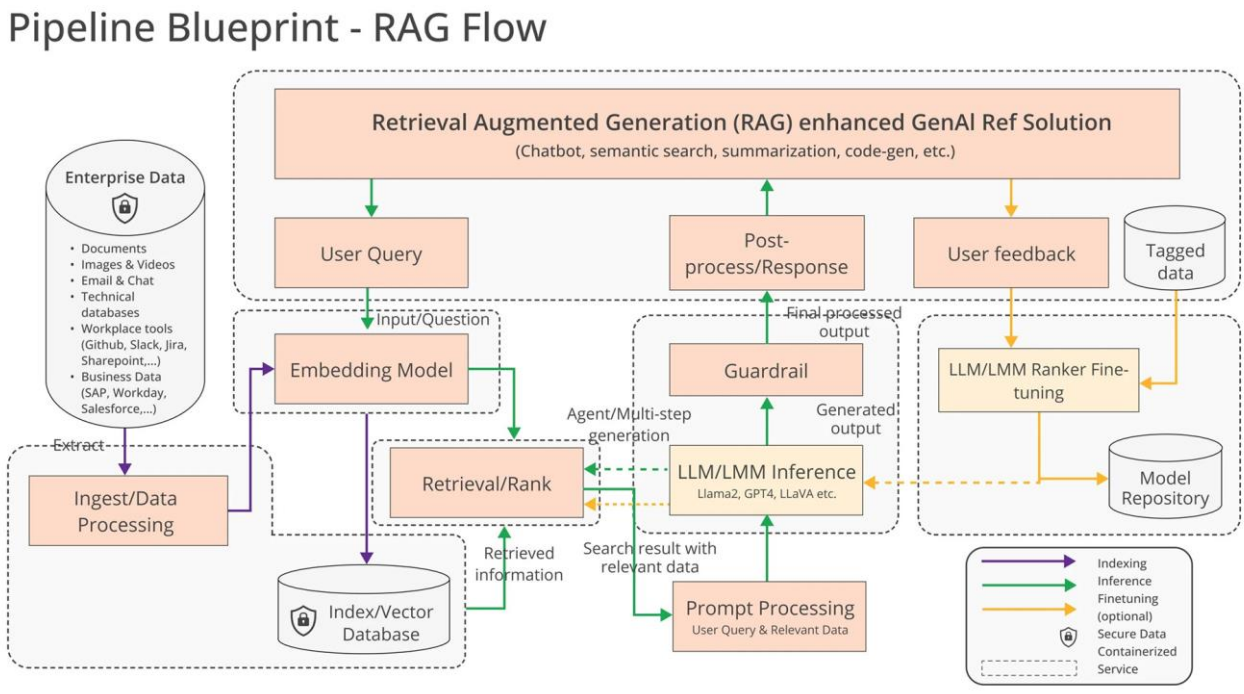


図 2. OPEA 上の検索拡張生成 (RAG) 強化 GenAI Ref ソリューション[14]

OPEA エンタープライズ AI ソリューションには、3 つの主要コンポーネントがあります。

1. **マイクロサービス:** 柔軟でスケーラブルなソリューションを提供します。
   GenAIComps リポジトリー (英語) には、利用可能なすべてのマイクロサービスがあります。各マイクロサービスは、アプリケーション・アーキテクチャー内で特定の機能またはタスクを実行するように設計されています。

2. **メガサービス:** 包括的なソリューションを提供します。
   GenAIExamples リポジトリー (英語) には、ユースケース・ベースのアプリケーションのコレクションがあります。特定のタスク用の個々のマイクロサービスとは異なり、メガサービスは包括的なソリューションを提供するため複数のマイクロサービスをオーケストレーションします。

3. **ゲートウェイ:** 通信を提供します。
   ゲートウェイは、ユーザーがメガサービスとその基盤となるマイクロサービスにアクセスするためのインターフェイスとして機能します。ゲートウェイは、API 定義、バージョン管理、レート制限、リクエスト変換、およびマイクロサービスからのデータ取得をサポートします。

さらに、ほとんどの AI ソリューションは、ユーザーがより直接的、インタラクティブ、かつ視覚化された方法で OPEA メガサービスと対話できるようにする UI を備えています。

OPEA フレームワークで AI アバター・チャットボットを構築するには?

I. フローチャート

アプリケーション全体のフローをグラフに示します。例として、OPEA GenAIExamples リポジトリーの Avatar Chatbot サンプル[15] を使用します。フローチャート図では、アプリケーションの中核となる AvatarChatbot メガサービスに注目しています。このメガサービスは、有向非巡回グラフ (DAG) 形式で接続された 4 つの異なるマイクロサービス「自動音声認識 (ASR)」、「大規模言語モデル (LLM)」、「テキスト読み上げ (TTS)」、「アニメーション」をオーケストレーションします。

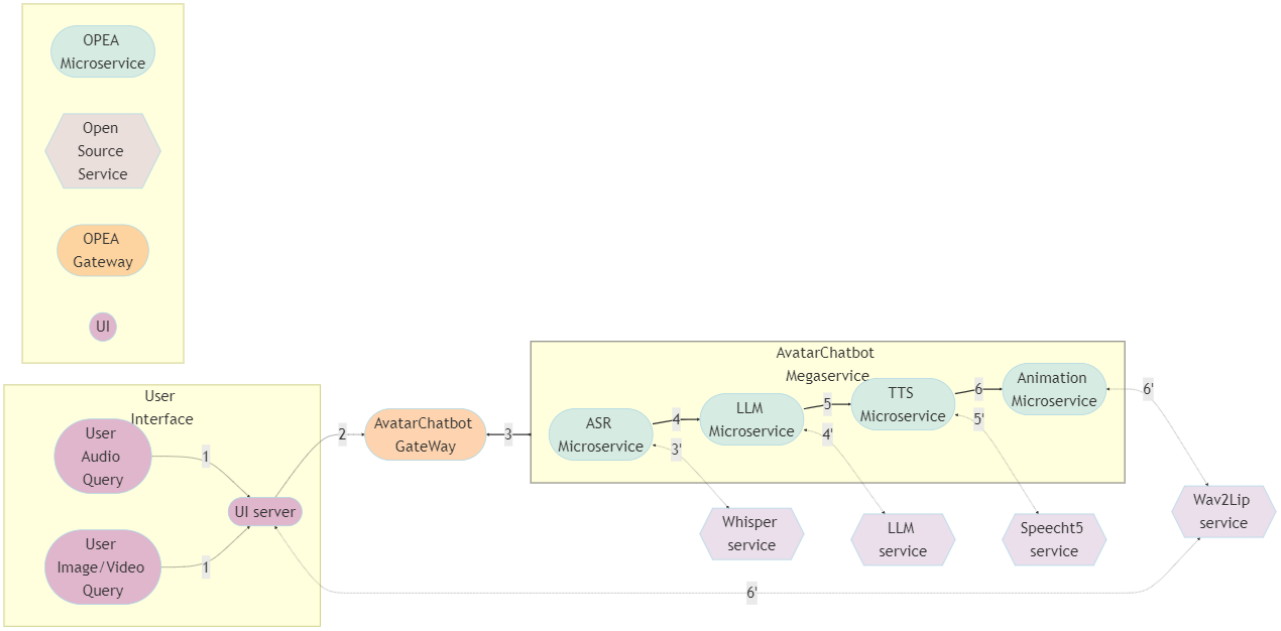


図 3. OPEA の AvatarChatbot サンプルのフローチャート[15]

各マイクロサービスは、アバター・チャットボットの特定の機能を処理します。次に例を示します。

- **自動音声認識 (ASR)** は、人間の音声をテキストに変換する音声認識ソフトウェアです。
- **大規模言語モデル (LLM)** は、ユーザーのクエリーを理解して ASR からの転写されたテキストを処理し、適切なテキスト応答を生成します。
- **テキスト読み上げ (TTS)** サービスは、LLM からの生成されたテキスト応答を音声音声に変換します。
- **アニメーション**・サービスは、TTS からの音声応答とユーザー定義の AI アバター画像/ビデオの両方を組み合わせ、アバターの唇の動きが同期された音声と一致するようにします。次に、アバターがユーザーに話しかけるビデオを生成します。

ユーザー入力には、音声クエリーと画像/ビデオの視覚入力が含まれます。出力は、顔がアニメーション化されたアバタービデオです。ユーザーは、音声による回答を聞いたり、チャットボットが自然に話すのを見たりなど、アバター・チャットボットからほぼリアルタイムのフィードバックを得られます。

1. GenAIComps リポジトリーに新しいマイクロサービス「Animation」を構築する

Animation などの新しいマイクロサービスを追加するには、そのマイクロサービスを comps/animation に登録する必要があります。

```
# Register the microservice
@register_microservice(
    name="opea_service@animation",
    service_type=ServiceType.ANIMATION,
    endpoint="/v1/animation",
    host="0.0.0.0",
    port=9066,
    input_datatype=Base64ByteStrDoc,
    output_datatype=VideoPath,
)
@register_statistics(names=["opea_service@animation"])
```

登録後、このマイクロサービスが実行される際のコールバック関数を定義します。Animation の場合、Base64ByteStrDoc オブジェクトを入力オーディオとして受け取り、生成されたアバタービデオへのパスを含む VideoPath オブジェクトを生成する animate 関数を使用します。animation.py 内で、FastAPI のエンドポイント「wav2lip」に API リクエストを送信し、レスポンスを収集します。リクエストの送信とレスポンスの収集はどちらも json 形式で行われます。

注: `comps/cores/proto/docarray.py` に Base64ByteStrDoc クラスと VideoPath クラスを追加し、`comps/__init__.py` にインポートするのを忘れないでください。

これは FastAPI で、入力されるオーディオ Base64Str とユーザー指定のアバター画像またはビデオを処理し、アニメーション・ビデオを出力してそのパスを返すポスト関数が指定されています。

上記の手順は、マイクロサービスの機能ブロックを作成するのに役立ちます。ユーザーが必要な依存関係を構築して Animation マイクロサービスを実行できるようにするには、wav2lip サーバー API 用の Dockerfile と Animation 用の Dockerfile を作成する必要があります。例えば、Dockerfile.intel_hpu は、インテル® Gaudi® AI アクセラレーター上の PyTorch インストーラー Docker イメージで始まり、「entrypoint」bash スクリプトの呼び出しで終わります。

2. GenAIExamples の新しいメガサービス「AvatarChatbot」をビルドする

最初に、Python ファイル AvatarChatbot/docker/avatarchatbot.py でメガサービス・クラス AvatarChatbotService を定義します。add_remote_service 関数でメガサービス・オーケストレーターの add 関数を使用して asr、llm、tts、および animation マイクロサービスを有向非巡回グラフ (DAG) のノードとして追加し、flow_to 関数を使用してエッジに接続します。

```
class AvatarChatbotService:
    def __init__(self, host="0.0.0.0", port=8000):
        self.host = host
        self.port = port
        self.megaservice = ServiceOrchestrator()

    def add_remote_service(self):
        asr = MicroService(
            name="asr",
            host=ASR_SERVICE_HOST_IP,
            port=ASR_SERVICE_PORT,
            endpoint="/v1/audio/transcriptions",
            use_remote_service=True,
            service_type=ServiceType.ASR,
        )
        llm = MicroService(
            name="llm",
            host=LLM_SERVICE_HOST_IP,
            port=LLM_SERVICE_PORT,
            endpoint="/v1/chat/completions",
            use_remote_service=True,
            service_type=ServiceType.LLM,
        )
        tts = MicroService(
            name="tts",
            host=TTS_SERVICE_HOST_IP,
            port=TTS_SERVICE_PORT,
```

```
            endpoint="/v1/audio/speech",
            use_remote_service=True,
            service_type=ServiceType.TTS,
        )
        animation = MicroService(
            name="animation",
            host=ANIMATION_SERVICE_HOST_IP,
            port=ANIMATION_SERVICE_PORT,
            endpoint="/v1/animation",
            use_remote_service=True,
            service_type=ServiceType.ANIMATION,
        )
        self.megaservice.add(asr).add(llm).add(tts).add(animation)
        self.megaservice.flow_to(asr, llm)
        self.megaservice.flow_to(llm, tts)
        self.megaservice.flow_to(tts, animation)
        self.gateway = AvatarChatbotGateway(megaservice=self.megaservice,
host="0.0.0.0", port=self.port)
```

3. メガサービスのゲートウェイを定義する

ゲートウェイは、ユーザーがメガサービスにアクセスするためのインターフェイスです。Python ファイル GenAIComps/comps/cores/mega/gateway.py で AvatarChatbotGateway クラスを定義します。AvatarChatbotGateway には、メガサービス・オーケストレーター、ホスト、ポート、エンドポイント、入力と出力のデータ型の情報が含まれています。また、最初のマイクロサービスに初期入力とパラメーターを渡すスケジュールを設定し、最後のマイクロサービスから応答を収集する handle_request 関数もあります。最後に、ユーザーが AvatarChatbot バックエンド Docker イメージを容易にビルドし、AvatarChatbot サンプルをデプロイできるように、Dockerfile を作成する必要があります。Dockerfile には、必要な GenAI コンポーネントと依存関係をインストールするスクリプトが含まれます。

II. リップシンク・モデルと顔アニメーション・モデル

1. Wav2Lip + GFPGAN

Wav2Lip[2] は、ディープラーニングを活用してオーディオとビデオを正確に一致させる最先端のリップシンク技術です。Wav2Lip には以下が含まれます。

- 実際のビデオで同期を正確に検出する事前トレーニング済みのエキスパート・リップシンク識別器
- フレーム単位で会話する顔のビデオを生成する改良された LipGAN[6]

事前トレーニング・ステージには、LRS2[4] データセットでのエキスパート・リップシンク識別器のトレーニングが含まれます。リップシンク・エキスパートは、入力されたビデオとオーディオのペアが同期している確率を出力するように事前トレーニングされています。

Wav2Lip のトレーニングにおいては、LipGAN と同様のアーキテクチャーが使用されます。生成器には、音声エンコーダー、ビジュアル・エンコーダー、および顔デコーダーが含まれます。これら 3 つはすべて畳み込み層のスタックです。識別器も畳み込みブロックです。改良された LipGAN は、ほかの GAN と同様にトレーニングされます。生成器は、識別器のスコアに基づいて敵対的損失を最小限に抑えるようにトレーニングされます。識別器は、生成器によって生成されたフレームと実際のフレームを区別するようにトレーニングされます。全体として、生成器は次の損失コンポーネントの重み付き合計を最小化するようにトレーニングされます。

1. 生成されたフレームと実際のフレーム間の L1 再構成損失
2. 生成されたビデオフレームとリップシンク・エキスパートからの入力オーディオ間の同期損失
3. 識別器のスコアに基づく、生成されたフレームと実際のフレーム間の敵対的損失

推論時に、Wav2Lip モデルに、前の TTS ブロックからの音声スピーチと、アバターフィギュアを含むビデオフレームをフィードします。トレーニングされた Wav2Lip モデルは、アバターがスピーチを話すリップシンク・ビデオを出力します。

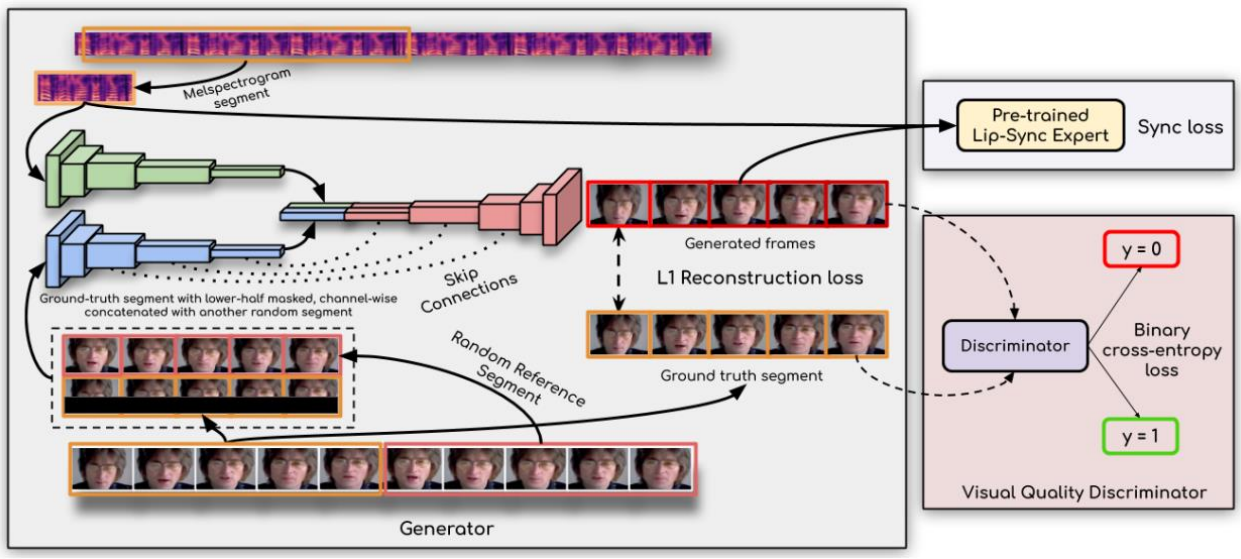


図 4. 正確なリップシンクを生成する Wav2Lip[2] アプローチ

Wav2Lip から生成されたビデオはリップシンクを特徴としていますが、口周りの解像度が低下します。オプションとして、Wav2Lip の後に GFPGAN[7] モデルを追加して、生成されたビデオフレーム内の顔の品質を向上させることができます。GFPGAN モデルは、顔の復元を通じて、未知の劣化を含む入力顔画像から高品質の画像を推定します。これは、事前トレーニング済みの顔 GAN (Style-GAN2[3] など) を事前に使用する U-Net[8] 劣化除去モジュールです。GFPGAN モデルは、出力フレームで高品質の顔の詳細を復元するように事前トレーニングされており、より鮮明でリアルなアバター表現が得られます。

2. SadTalker

Wav2Lip に加えて、顔アニメーションを実行する別の最先端モデルの選択肢を提供します。スタイライズされた音声駆動型トーキング・ヘッド・ビデオ生成の SadTalker[5] は、オーディオから 3D モーフィング・モデル (3DMM)[1] の 3D モーション係数 (頭、姿勢、表情) を生成します。これらの係数は 3D キーポイントにマッピングされ、さらに 3D 対応の顔レンダラーを介して入力画像を駆動するために使用されます。出力はリアルなトーキング・ヘッド・ビデオです。

Wav2Lip モデルと SadTalker モデルの両方をインテル® Xeon® スケーラブル・プロセッサーにデプロイし、Wav2Lip モデルをインテル® Gaudi® AI アクセラレーターにデプロイできるようにしました。

インテル® Xeon® スケーラブル・プロセッサーとインテル® Gaudi® AI アクセラレーターにデプロイする方法

デプロイ環境に応じて、同じ FastAPI サービス/マイクロサービス/メガサービスであっても必要な Docker イメージが異なる場合があります。例えば、wav2lip サービスコンテナは、インテル® Xeon® スケーラブル・プロセッサーにデプロイする場合は opea/wav2lip:latest という名前のイメージを使用し、インテル® Gaudi® AI アクセラレーターにデプロイする場合は opea/wav2lip-gaudi:latest という名前の別のイメージを使用します。これらのイメージは、それぞれの依存関係を含めるため、別々のバージョンの Dockerfile を使用します。

```
services:
  whisper-service:
    image: ${REGISTRY:-opea}/whisper:${TAG:-latest}
    container_name: whisper-service
    ports:
      - "7066:7066"
    ipc: host
    environment:
      no_proxy: ${no_proxy}
      http_proxy: ${http_proxy}
      https_proxy: ${https_proxy}
    restart: unless-stopped
```

```
services:
  whisper-service:
    image: ${REGISTRY:-opea}/whisper-gaudi:${TAG:-latest}
    container_name: whisper-service
    ports:
      - "7066:7066"
    ipc: host
    environment:
      no_proxy: ${no_proxy}
      http_proxy: ${http_proxy}
      https_proxy: ${https_proxy}
      HABANA_VISIBLE_MODULES: all
      OMPI_MCA_btl_vader_single_copy_mechanism: none
    runtime: habana
    cap_add:
      - SYS_NICE
    restart: unless-stopped
```

図 5. インテル® Xeon スケーラブル・プロセッサーと
インテル® Gaudi® AI アクセラレーターの YAML ファイルの比較

上記の違いにより、サービスコンテナごとに YAML ファイルを明示的に設定する必要があります。compose.yaml ファイルを使用すると、OPEA ユーザーは次の主要な要素をカスタマイズできます。

- サービスイメージ: 各サービスで使用する特定の Docker イメージ
- ポート: 外部ポートをコンテナの内部ポートにマッピングする
- 環境変数: 各サービスに必要なコンテキストと構成 (例: LLM モデル名、デバイス、推論モード、追加入力など) があることを確認する
- ボリューム: ホストとコンテナ間で共有されるデータ
- IPC、ネットワーク、ランタイム、cap_add などのその他の要素

デフォルトでは、インテル® Gaudi® AI アクセラレーターでは、OPEA ベースの AI アバター音声チャットボットのサンプルは、単一のインテル® Gaudi® 2 AI アクセラレーター・ノード上の 4 枚のインテル® Gaudi® カードにワークロードを分散します。各マイクロサービスは 1 枚のカードに割り当てられます。これは compose.yaml 構成ファイルで確認できます。asr、llm、tts、および animation マイクロサービスの Docker コンテナはそれぞれ、**HABANA_VISIBLE_MODULES** 環境変数を値 0、1、2、3 に設定することで、専用のインテル® Gaudi® カードに関連付けられます。これは「それぞれ単一のワークロードを持つ複数の Docker」(英語) と呼ばれます[10]。

インテル® Gaudi® 2 AI アクセラレーター・ノードを詳しく調べると、インテル® Gaudi® カードのレイアウトがより明確になります。まず、hl-smi システム管理インターフェイス・ツール[9] とマッピングのサンプル出力 (英語) を使用して、インテル® Gaudi® プロセッサーのインデックスとモジュール ID 間のマッピングを見つけます。

同様に、別の hl-smi コマンドを使用して、NUMA アフィニティー (英語) を調べることができます[9]。デバイスの NUMA (Non-Uniform Memory Access) アフィニティーとは、パフォーマンスを最適化するためデバイスを特定のメモリー領域にアラインメントすることを指します。インテル® Gaudi® AI アクセラレーター・ノードのようなマルチカード・システムでは、NUMA アフィニティーにより、インテル® Gaudi® カードは、最も近い CPU に

よって制御されるメモリーに高速にアクセスできます。この例 (英語) では、モジュール ID が 0～3 のインテル® Gaudi® カードは、CPU 0 と 1 によって制御されるメモリーに対応しています。

ディープラーニングの最適化

動作モード: Eager モードと Lazy モード

動作モードは、値 0 と 1 (Eager モードの場合は 0、Lazy モードの場合は 1) に設定可能な環境変数 PT_HPU_LAZY_MODE によって制御されます。

GenAIComps/comps/animation/entrypoint.sh では、Eager モードを使用するためこの変数を 0 に設定します。同時に、顔検出器、Wav2Lip、および GFPGAN モデルを対応するグラフにラップして、torch.compile で Eager モードを拡張します。Lazy モードとは異なり、torch.compile を使用した Eager モードでは、各反復でグラフを構築する必要がないため、ホストの計算オーバーヘッドが軽減されます。さらに、torch.compile のバックエンド・パラメーターは、トレーニングと推論の両方で hpu_backend に設定する必要があります。Eager モード + torch.compile をサポートする一般的なモデルのリストは、[11] にあります。

```
# Load Wav2Lip, BG sampler, GFPGAN models
model = load_model(args)
model = torch.compile(model, backend="hpu_backend")
print("Wav2Lip Model loaded")
```

一方、Lazy モードは、ユーザーが PyTorch の Define-by-Run アプローチの柔軟性と利点を維持するのに役立ちます。累積グラフ内のオペレーションの実行は、テンソル値が必要な場合にのみトリガーされます。これにより、インテル® Gaudi® カードのグラフを複数のオペレーションで構築できるようになり、グラフ・コンパイラーがこれらのオペレーションのデバイス実行を最適化する機会が提供されます。詳細については、「PyTorch Gaudi Theory of Operations」[16] を参照してください。

PyTorch autocast とインテル® ニューラル・コンプレッサーで BF16 と FP8 を使用した推論を実行する

混合精度の量子化により、ディープラーニング・ニューラル・ネットワークのオペレーションをより高速に実行しながら、メモリー上のモデルの重みのサイズを軽減できます。この例では、2 つの方法で顔検出器と Wav2Lip モデルの量子化を行いました。

1 つ目の方法では、ネイティブの PyTorch autocast を使用して、登録済みのオペレーター (オペレーション) のデフォルトリストを低精度の bfloat16 データ型で自動的に実行します。オペレーションのリストは [12] にあります。これにより、BF16 の顔アニメーション・モジュールの推論が可能になります。

```
with torch.no_grad():
    with torch.autocast(device_type="hpu", dtype=torch.bfloat16):
        pred = model(mel_batch, img_batch)
```

2 つ目の方法では、インテル® ニューラル・コンプレッサー (英語) パッケージを利用して、インテル® Gaudi® AI アクセラレーターで FP8 推論を有効にできます[18]。推論に FP8 データ型を使用すると、モデルに必要なメモリー帯域幅が半分になります。さらに、FP8 の計算速度は BF16 の 2 倍です。最初に JSON 構成ファイルを作成し、次に convert API を使用してモデルを量子化する必要があります。

機能拡張

テキスト読み上げ (TTS) サービス

OPEA で使用するデフォルトの TTS モデルとして、SOTA モデル「microsoft/SpeechT5」[17] を使用しました。テキストトークンの全長を音声に変換する代わりに、speecht5_model.py ファイルのコードを使用して、長いテキストのバッチ分割中に最後のトークンチャンクの最後の句読点を自動的に検出します。この変更により、TTS サービスは突然停止することなく、完全かつ連続したテキストを出力できます。

アニメーション・サービス

Wav2Lip[2] アニメーションでは、ビデオフレーム生成の 1 秒当りのフレーム数 (fps) を調整できます。コードは、こちらにあります。これは、ユーザー指定の fps パラメーターによって制御されます。Wav2Lip への視覚入力がアバターの顔を含む画像である場合、最終的なビデオのフレームレートを自由に定義できます。フレームレートに基づいて、可変数のオーディオ・メル・スペクトログラム・チャンクが 1 つのフレームに融合されます。fps=30 ではビデオ・レンダリングの滑らかさがわずかに向上しますが、fps=10 に設定すると、生成されるビデオフレームが 1/3 になり、ニューラル・ネットワークの反復と計算が 1/3 になります。フレームレートを調整可能にすることは、低レイテンシー、高スループットのアニメーション・シナリオに最適です。

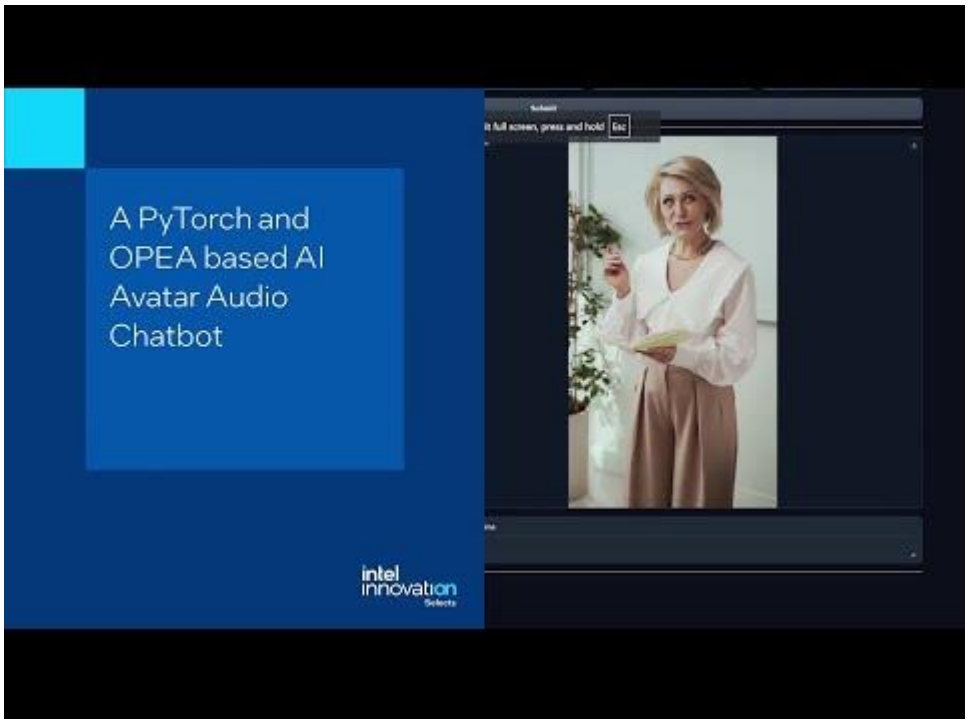
```
    if not os.path.isfile(args.face):
        raise ValueError("--face argument must be a valid path to video/image
file")
    elif args.face.split(".")[-1] in ["jpg", "jpeg", "png"]:
        full_frames = [cv2.imread(args.face)]
        fps = args.fps
    else:
        video_stream = cv2.VideoCapture(args.face)
        fps = video_stream.get(cv2.CAP_PROP_FPS)
.............
.............

 # one single video frame corresponds to 80/25*0.01 = 0.032 seconds (or 32
milliseconds) of audio
    mel_chunks = []
    mel_idx_multiplier = 80.0 / fps
```

AI アバター・チャットボットの実行手順 – 実際に試してみよう!

AvatarChatbot マイクロサービス (英語) とアバター・アニメーション・マイクロサービスの手順に従って、次の操作を行います。

1. すべてのマイクロサービス (asr、llm、tts、animation) と AvatarChatbot メガサービスに必要な Docker イメージをビルドします。
2. エクスポート・コマンドを使用して必要な環境変数を設定します。
3. Docker Compose を使用して、インテル® Xeon® スケーラブル・プロセッサー/インテル® Gaudi® AI アクセラレーターで Docker サービスを開始します。
4. curl コマンドまたは Python ファイルを使用して Docker サービスを検証します。
5. ローカルブラウザーの Gradio UI で AI アバター・アプリケーションを操作します

この AI アバター・アプリケーションを実行する方法については、短いデモ (英語) をご覧ください。



今後の取り組み

LLM および TTS 出力とアニメーション・フレームのリアルタイム・ストリーミング

Hugging Face LLM モデル API[19] を使用すると、LLM のストリーミング・モードを設定して、トークンのストリームを非同期に生成できます。コードは Python ファイルで使用できます。LLM パラメーターで streaming=True を設定し、TTS でのストリーミングを有効にすると、テキストトークンと対応するオーディオ波形がより速いペースで生成されることが期待できます。次に、フレームバッファーなどの手法を適用して、Animation マイクロサービスによって処理されたフレームを蓄積します。バッファーフレームの数がチェックポイント (例えば、予想されるフレームの総数の 1/3) を超えたら、出力ビデオのストリーミングを開始します。そして、ユーザーはすぐにアバター・ビデオ・アニメーションを体験できます。

関連資料

- 概要 - インテル® Gaudi® AI アクセラレーター
- 概要 - インテル® Xeon® スケーラブル・プロセッサー
- Open Platform for Enterprise AI の概要 (英語)
- OPEA プロジェクトの GitHub リポジトリー (英語)
- インテル® Tiber™ AI クラウド
- 公式サイト - インテルの PyTorch 最適化 (英語)
- PyTorch 向けインテル® エクステンション - ドキュメント (英語)

参考資料

[1] B. Egger et al., "3D Morphable Face Models -- Past, Present and Future," Apr. 16, 2020, arXiv: arXiv:1909.01815. Accessed: Sep. 12, 2024. [Online]. Available: http://arxiv.org/abs/1909.01815 (英語).

[2] K. R. Prajwal, R. Mukhopadhyay, V. Namboodiri, and C. V. Jawahar, "A Lip Sync Expert Is All You Need for Speech to Lip Generation In The Wild," in Proceedings of the 28th ACM International Conference on Multimedia, Oct. 2020, pp. 484–492. doi: 10.1145/3394171.3413532 (英語).

[3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," Mar. 23, 2020, arXiv: arXiv:1912.04958. doi: 10.48550/arXiv.1912.04958 (英語).

[4] "Lip Reading Sentences 2 (LRS2) dataset." Accessed: Sep. 12, 2024. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrs2.html (英語).

[5] W. Zhang et al., "SadTalker: Learning Realistic 3D Motion Coefficients for Stylized Audio-Driven Single Image Talking Face Animation," arXiv.org. Accessed: Sep. 12, 2024. [Online]. Available: https://arxiv.org/abs/2211.12194v2 (英語).

[6] P. K. R, R. Mukhopadhyay, J. Philip, A. Jha, V. Namboodiri, and C. V. Jawahar, "Towards Automatic Face-to-Face Translation," in Proceedings of the 27th ACM International Conference on Multimedia, Oct. 2019, pp. 1428–1436. doi: 10.1145/3343031.3351066 (英語).

[7] X. Wang, Y. Li, H. Zhang, and Y. Shan, "Towards Real-World Blind Face Restoration with Generative Facial Prior," Jun. 10, 2021, arXiv: arXiv:2101.04061. doi: 10.48550/arXiv.2101.04061 (英語).

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," May 18, 2015, arXiv: arXiv:1505.04597. doi: 10.48550/arXiv.1505.04597 (英語).

[9] "System Management Interface Tool (hl-smi) — Gaudi Documentation 1.17.1 documentation." Accessed: Oct. 03, 2024. [Online]. Available: https://docs.habana.ai/en/latest/Management_and_Monitoring/Embedded_System_Tools_Guide/System_Management_Interface_Tool.html?highlight=numa#hl-smi-utility-options (英語).

[10] "Multiple Dockers Each with a Single Workload — Gaudi Documentation 1.17.1 documentation." Accessed: Oct. 03, 2024. [Online]. Available: https://docs.habana.ai/en/latest/Orchestration/Multiple_Tenants_on_HPU/Multiple_Dockers_each_with_Single_Workload.html (英語).

[11] HabanaAI/Model-References. (Oct. 02, 2024). Jupyter Notebook. Intel® Gaudi® AI Accelerator. Accessed: Oct. 04, 2024. [Online]. Available: https://github.com/HabanaAI/Model-References (英語).

[12] "Mixed Precision Training with PyTorch Autocast — Gaudi Documentation 1.17.1 documentation." Accessed: Oct. 04, 2024. [Online]. Available: https://docs.habana.ai/en/latest/PyTorch/PyTorch_Mixed_Precision/index.html (英語).

[13] "Stable Diffusion 3 Medium - a Hugging Face Space by stabilityai." Accessed: Oct. 20, 2024. [Online]. Available: https://huggingface.co/spaces/stabilityai/stable-diffusion-3-medium (英語).

[14] "Open Platform For Enterprise AI," Open Platform for Enterprise AI (OPEA). Accessed: Oct. 20, 2024. [Online]. Available: https://opea.dev/ (英語).

[15] "AvatarChatbot · opea-project/GenAIExamples," GitHub. Accessed: Oct. 20, 2024. [Online]. Available: https://github.com/opea-project/GenAIExamples/tree/main/AvatarChatbot (英語).

[16] "PyTorch Gaudi Theory of Operations — Gaudi Documentation 1.18.0 documentation." Accessed: Oct. 20, 2024. [Online]. Available: https://docs.habana.ai/en/latest/PyTorch/Reference/PyTorch_Gaudi_Theory_of_Operations.html (英語).

[17] J. Ao et al., "SpeechT5: Unified-Modal Encoder-Decoder Pre-Training for Spoken Language Processing," May 24, 2022, arXiv: arXiv:2110.07205. Accessed: Oct. 20, 2024. [Online]. Available: http://arxiv.org/abs/2110.07205 (英語).

[18] "Run Inference Using FP8 — Gaudi Documentation 1.18.0 documentation." Accessed: Oct. 20, 2024. [Online]. Available: https://docs.habana.ai/en/latest/PyTorch/Inference_on_PyTorch/Quantization/Inference_Using_FP8.html (英語).

[19] "langchain_community.llms.huggingface_endpoint.HuggingFaceEndpoint — 🦜🔗 LangChain 0.2.16." Accessed: Oct. 20, 2024. [Online]. Available: https://api.python.langchain.com/en/latest/llms/langchain_community.llms.huggingface_endpoint.HuggingFaceEndpoint.html (英語).

製品および性能に関する情報

[1] 性能は、使用状況、構成、その他の要因によって異なります。詳細については、http://www.intel.com/PerformanceIndex/ (英語) を参照してください。