インテル® Gaudi® 2 AI アクセラレーターとインテル® Xeon® プロセッサーを使用したコスト効率の高いエンタープライズ RAG アプリケーションの構築

この記事は、Hugging Face* のサイトで公開されている「Building Cost-Efficient Enterprise RAG applications with Intel Gaudi 2 and Intel Xeon」を、Hugging Face* の許可取り iSUS で翻訳した日本語参考訳です。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。

RAG (Retrieval-Augmented Generation) は、外部データストアに保存されている関連分野 の最新の知識を取り込んで、大規模言語モデル (LLM) によるテキスト生成を強化します。パフォーマンス、精度、セキュリティーとプライバシーの目標に対しバランスを取るには、トレーニング中に言語モデルが学習した知識から組織のデータを分離することが不可欠です。

このブログでは、OPEA (Open Platform for Enterprise AI) (英語) の一部として、インテルが提供している RAG アプリケーションの開発とデプロイの支援について説明します。また、実際の RAG ユースケースを通じて、インテル® Gaudi® 2 AI アクセラレーターとインテル® Xeon® プロセッサーがエンタープライズ・パフォーマンスを大幅に向上させる方法についても説明します。

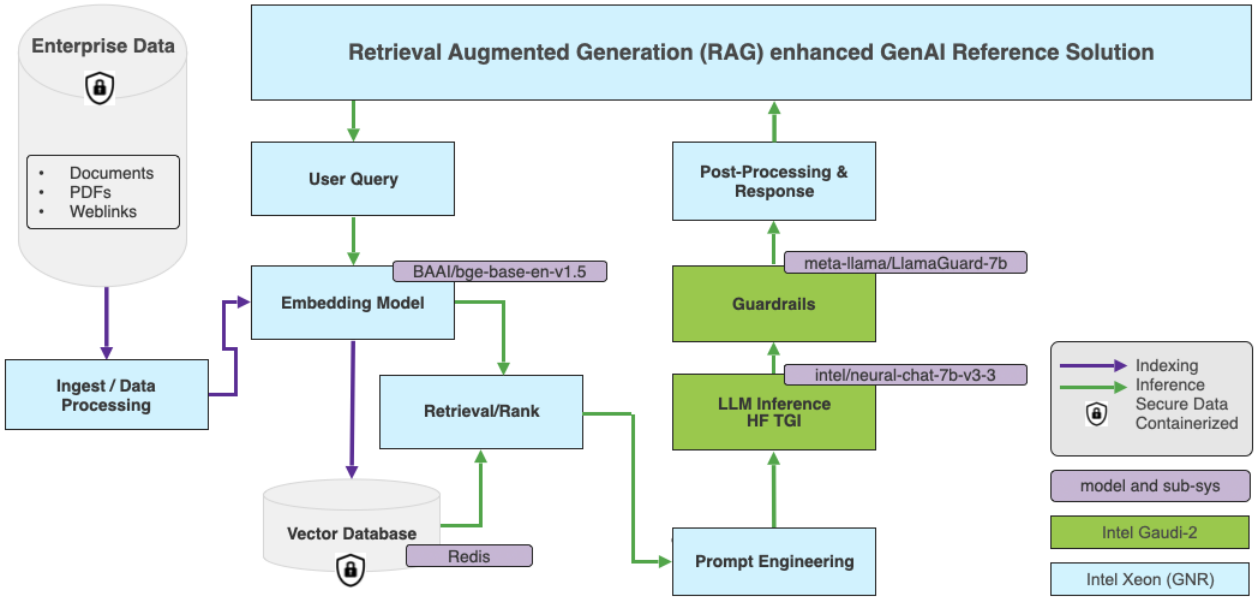
はじめに

詳細に入る前に、まずハードウェアへのアクセスを確認しましょう。インテル® Gaudi® 2 AI アクセラレーターは、データセンターとクラウドでディープラーニングのトレーニングと推論を加速するように設計されており、インテル® デベロッパー・クラウド (IDC) (現在のインテル® Tiber™ デベロッパー・クラウド) およびオンプレミス実装で利用できます。IDC は、インテル® Gaudi® 2 AI アクセラレーターを使い始める最も簡単な方法です。

> **[訳者注:]**
>
> 現在 IDC は新規のアカウント登録を受け付けていません。
>
> 後継のインテル® Tiber™ デベロッパー・クラウドでは、Enterprise ELA プランの利用者向けにインテル® Gaudi® 2 AI アクセラレーターへのアクセスを承認制で提供しています。
>
> インテル® Tiber™ デベロッパー・クラウドの詳細については、以下のサイトを参照してください。
>
> - サービス概要
>   https://www.xlsoft.com/jp/products/intel/devcloud/index.html
> - Enterprise ELA プランの特長
>   https://www.xlsoft.com/jp/products/intel/devcloud/services.html

アプリケーションの構築には、LLM を使用した AI アプリケーションの作成を簡素化するオープンソース・フレームワークの LangChain* を使用します。テンプレート・ベースのソリューションにより、開発者はカスタム埋め込み、ベクトル・データベース、および LLM を使用して RAG アプリケーションを構築できます。詳細は、LangChain* のドキュメントを参照してください。インテルは、開発者がインテルのプラットフォームで生成 AI アプリケーションを効率良くデプロイできるように LangChain* に複数の最適化を提供しています。

LangChain* の `rag-redis` テンプレート、BAAI/bge-base-en-v1.5 (英語) 埋め込みモデル、デフォルトのベクトル・データベースとして Redis を使用して、RAG アプリケーションを作成します。次の図は、高レベルのアーキテクチャーを示しています。



埋め込みモデルは、インテル® Xeon® プロセッサー (開発コード名 Granite Rapids、略称 GNR) 上で実行されます。GNR アーキテクチャーは、ハイコア・パフォーマンス重視のワークロードと汎用コンピューティング・ワークロードの総所有コスト (TCO) を最小にするように最適化されています。GNR はまた、インテル® AMX-FP16 命令セットもサポートしているため、AI ワークロードが混合するパフォーマンスが 2 ~ 3 倍向上します。

LLM はインテル® Gaudi® 2 AI アクセラレーター上で実行されます。Hugging Face* モデルに関しては、Hugging Face* Transformers (英語) および Diffusers (英語) ライブラリーとインテル® Gaudi® 2 AI アクセラレーター間のインターフェイスとして Optimum Habana ライブラリー (英語) を使用します。このライブラリーは、さまざまなダウンストリーム・タスクのシングルカード設定とマルチカード設定でモデルの読み込み、トレーニング、推論を容易にするツールを提供します。

LangChain* 開発環境を効率良くセットアップできる Dockerfile (英語) が容易されています。Docker* コンテナを起動したら、Docker* 環境内でベクトル・データベース、RAG パイプライン、LangChain* アプリケーションの構築できます。詳細な手順については、ChatQnA (英語) の例に従ってください。

ベクトル・データベースの作成

一般公開されている Nike の財務書類を使用してベクトル・データベースにデータを入力します。以下にサンプルコードを示します。

```
# Ingest PDF files that contain Edgar 10k filings data for Nike.
company_name = "Nike"
data_path = "data"
doc_path = [os.path.join(data_path, file) for file in os.listdir(data_path)][0]
content = pdf_loader(doc_path)
chunks = text_splitter.split_text(content)

# Create vectorstore
embedder = HuggingFaceEmbeddings(model_name=EMBED_MODEL)

_ = Redis.from_texts(
    texts=[f"Company: {company_name}. " + chunk for chunk in chunks],
    embedding=embedder,
    index_name=INDEX_NAME,
    index_schema=INDEX_SCHEMA,
    redis_url=REDIS_URL,
)
```

RAG パイプラインの定義

LangChain* では、Chain API を使用してプロンプト、ベクトル・データベース、埋め込みモデルを接続します。

```
# Embedding model running on Xeon CPU
embedder = HuggingFaceEmbeddings(model_name=EMBED_MODEL)

# Redis vector database
vectorstore = Redis.from_existing_index(
    embedding=embedder, index_name=INDEX_NAME, schema=INDEX_SCHEMA,
redis_url=REDIS_URL
)

# Retriever
retriever = vectorstore.as_retriever(search_type="mmr")

# Prompt template
template = """…"""
prompt = ChatPromptTemplate.from_template(template)

# Hugging Face LLM running on Gaudi 2
model = HuggingFaceEndpoint(endpoint_url=TGI_LLM_ENDPOINT, …)

# RAG chain
chain = (
    RunnableParallel({"context": retriever, "question": RunnablePassthrough()}) |
prompt | model | StrOutputParser()
).with_types(input_type=Question)
```

インテル® Gaudi® 2 AI アクセラレーターへの LLM のロード

Hugging Face* Text Generation Inference (TGI) サーバーを使用して、インテル® Gaudi® 2 AI アクセラレーターでチャットモデルを実行します。この組み合わせにより、インテル® Gaudi® 2 AI アクセラレーター上で MPT、Llama*、Mistral* などの一般的なオープンソース LLM の高性能テキスト生成が可能になります。

セットアップは不要です。事前に構築された Docker* イメージを使用して、モデル名 (Intel NeuralChat など) を渡すことができます。

```
model=Intel/neural-chat-7b-v3-3
volume=$PWD/data
docker run -p 8080:80 -v $volume:/data --runtime=habana -e
HABANA_VISIBLE_DEVICES=all -e OMPI_MCA_btl_vader_single_copy_mechanism=none
--cap-add=sys_nice --ipc=host tgi_gaudi --model-id $model
```

デフォルトでは、サービスは単一のインテル® Gaudi® 2 AI アクセラレーターを使用します。70B などのより大きなモデルの実行には、複数のアクセラレーターが必要になります。その場合は、適切なパラメーター (例: `--sharded true` および `--num_shard 8`) を追加してください。Llama* (英語) や StarCoder (英語) などのゲートモデルの場合は、Hugging Face* token (英語) を使用して `-e HUGGING_FACE_HUB_TOKEN=<token>` も指定する必要があります。

コンテナを起動したら、TGI エンドポイントにリクエストを送信して、サービスが機能していることを確認します。

```
curl localhost:8080/generate -X POST \
-d &amp;apos;{"inputs":"Which NFL team won the Super Bowl in the 2010 season?", \
"parameters":{"max_new_tokens":128, "do_sample": true}}&amp;apos; \
-H &amp;apos;Content-Type: application/json&amp;apos;
```

応答があれば、LLM は正常に実行されており、インテル® Gaudi® 2 AI アクセラレーターで高性能な推論を利用できます。

インテル® Gaudi® 2 AI アクセラレーター搭載の TGI コンテナは、デフォルトで bfloat16 データ型を使用します。スループットを高めるには、FP8 量子化を有効にすることを推奨します。テスト結果では、FP8 量子化により、BF16 と比較してスループットが 1.8 倍向上しました。FP8 の説明は README ファイル (英語) を参照してください。

最後に、Meta* Llama* Guard (英語) モデルを使用してコンテンツ・モデレーションを有効にできます。Llama* Guard のデプロイ手順は、README (英語) を参照してください。

RAG サービスの実行

次の手順に従って、RAG アプリケーションのバックエンド・サービスを起動します。`server.py` スクリプトは、fastAPI を使用してサービス・エンドポイントを定義します。

```
docker exec -it qna-rag-redis-server bash
nohup python app/server.py &
```

デフォルトでは、インテル® Gaudi® 2 AI アクセラレーターの TGI エンドポイントはポート 8080 (つまり、`http://127.0.0.1:8080`) のローカルホストで実行されると想定されます。別のアドレスまたはポートで実行する場合は、`TGI_ENDPOINT` 環境変数の設定を調整してください。

RAG GUI の起動

次の手順に従って、フロントエンド GUI コンポーネントをインストールします。

```
sudo apt-get install npm && \
    npm install -g n && \
    n stable && \
    hash -r && \
    npm install -g npm@latest
```

次に、ローカルホストの IP アドレス (`127.0.0.1`) を GUI が実行されるサーバーの実際の IP アドレスに置き換えて、`.env` ファイル内の `DOC_BASE_URL` 環境変数を更新します。

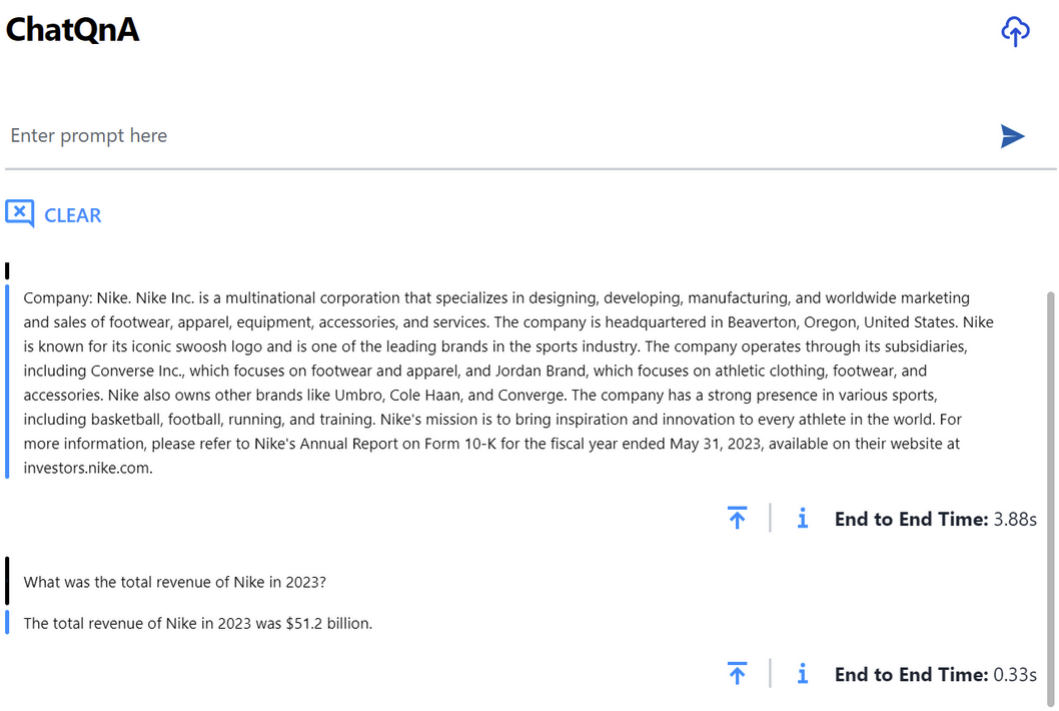
次のコマンドを実行して、必要な依存関係をインストールします。

```
npm install
```

最後に、次のコマンドで GUI サーバーを起動します。

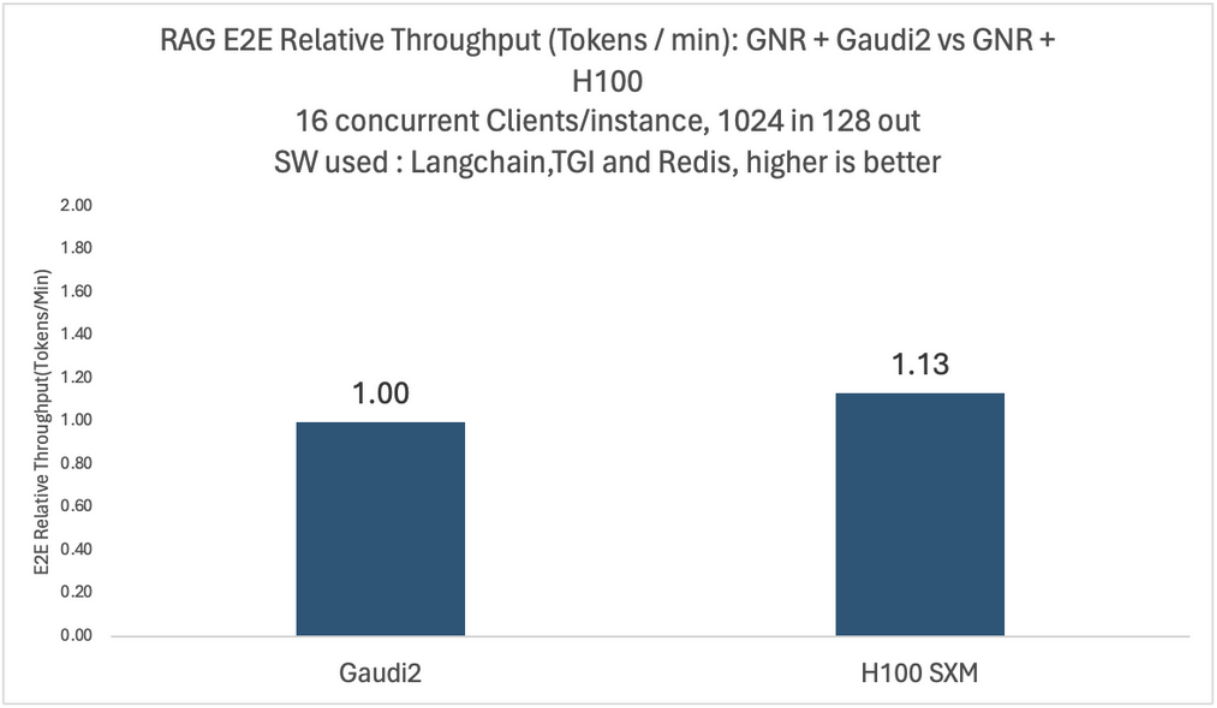
```
nohup npm run dev &
```

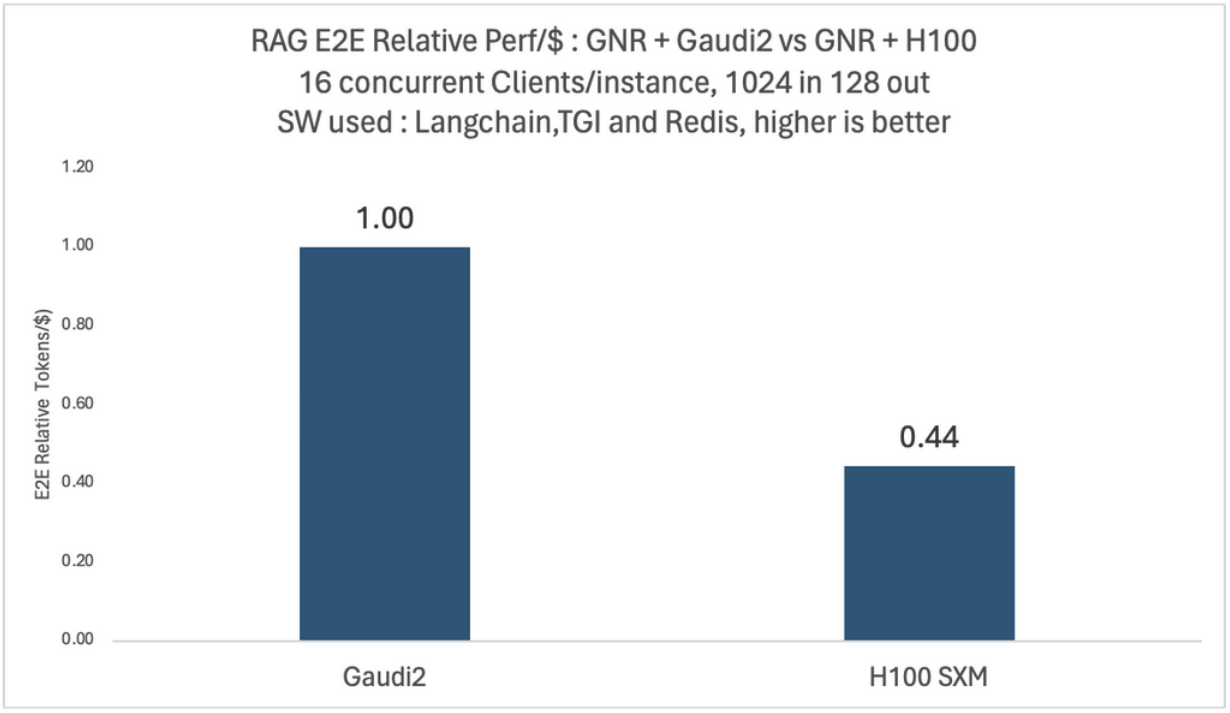
フロントエンド・サービスが実行され、アプリケーションが起動します。



ベンチマーク結果

さまざまなモデルと構成で徹底的な実験を行いました。以下の 2 つの図は、4 x インテル® Gaudi® 2 AI プロセッサー搭載プラットフォームと 4 x NVIDIA* H100 搭載プラットフォーム上で、16 人の同時ユーザーを持つ Llama2-70B モデルの相対的なエンドツーエンドのスループットと 1 ドルあたりのパフォーマンスの比較を示しています。





どちらの場合も、ベクトル・データベースと埋め込みモデルには同じインテル® Xeon® プロセッサー (開発コード名 Granite Rapids) ベースのプラットフォームが使用されています。1 ドルあたりのパフォーマンスの比較では、2024 年 1 月に MosaicML チームが報告したもの (英語) と同じ、公表されている価格設定を使用して、1 ドルあたりの平均トレーニング・パフォーマンスを計算します。

図から、H100 ベースのシステムは、インテル® Gaudi® 2 AI アクセラレーター・ベースのシステムと比較して 1.13 倍のスループットがありますが、1 ドルあたりのパフォーマンスは 0.44 倍しか提供していないことが分かります。比較結果は、各クラウド・プロバイダーが提供する顧客固有の割引によって異なる場合があります。詳細なベンチマーク構成は、記事の最後にあります。

まとめ

上記のデプロイメント例は、インテルのプラットフォーム上で RAG ベースのチャットボットを正常に実行できることを示しています。インテルはすぐに使用できる生成 AI サンプルを継続的にリリースしており、開発者は開発とデプロイメントのプロセスを簡素化する検証済みのツールの恩恵を受けることができます。これらのサンプルは汎用的でカスタマイズが容易なため、インテルのプラットフォーム上のさまざまなアプリケーションに最適です。

エンタープライズ AI アプリケーションを実行する場合、インテル® Xeon® プロセッサー (開発コード名 Granite Rapids) とインテル® Gaudi® 2 AI アクセラレーター・ベースのシステムのほうが総所有コストは優れており、FP8 の最適化によりさらに改善できます。

以下の開発者向けのリソースは、生成 AI プロジェクトを開始するのに役立ちます。

- OPEA の生成 AI サンプル (英語)
- インテル® Gaudi® 2 AI アクセラレーター上での TGI (英語)
- インテルの AIML エコシステム: Hugging Face* (英語)
- Hugging Face* Hub のインテルのページ (英語)

ご意見、ご質問がある場合は、Hugging Face* フォーラム (英語) までお寄せください。最後までお読みいただきありがとうございます。

謝辞: インテル® Gaudi® 2 AI アクセラレーター上でのエンタープライズ・グレードの RAG システムの構築に多大な貢献をしてくれた Chaitanya Khened、Suyue Chen、Mikolaj Zyczynski、Wenjiao Yue、Wenxin Zhang、Letong Han、Sihan Chen、Hanwen Cheng、Yuan Wu、および Yi Wang に感謝します。

ベンチマーク構成

- インテル® Gaudi® 2 プラットフォームの構成: インテル® HLS-Gaudi® 2 AI アクセラレーター・サーバー (8 x インテル® Gaudi® 2 AI アクセラレーター HL-225H メザニン・カード)、2 x インテル® Xeon® Platinum 8380 プロセッサー @2.30GHz、システムメモリー 1TB。OS: Ubuntu* 22.04.03、カーネル 5.15.0。

- H100 SXM プラットフォームの構成: Lambda Labs インスタンス gpu_8x_h100_sxm5。8 x H100 SXM、2 x インテル® Xeon® Platinum 8480 プロセッサー @2GHz、システムメモリー 1.8TB。OS: Ubuntu* 20.04.6 LTS、カーネル 5.15.0。
- 出荷前のインテル® Xeon® プロセッサー (開発コード名 Granite Rapids) プラットフォーム: 2 ソケット x 120 コア @1.90GHz、8800 MCR DIMM、システムメモリー 1.5TB。OS: CentOS* 9、カーネル 6.2.0。
- Llama2 70B は 4 枚のカードにデプロイ (クエリーは 8 枚のカードに正規化)。インテル® Gaudi® 2 AI アクセラレーターでは BF16 を、H100 では FP16 を使用。
- 埋め込みモデル: BAAI/bge-base v1.5。テスト構成: TGI-gaudi 1.2.1、TGI-GPU 1.4.5 Python* 3.11.7、LangChain* 0.1.11、sentence-transformers 2.5.1、LangChain* Benchmarks 0.0.10、Redis 5.0.2、CUDA* 12.2.r12.2/compiler.32965470_0、TEI 1.2.0。
- RAG クエリーの最大入力長 1024、最大出力長 128。テスト・データセット: langsmith Q&A。同時実行クライアント数: 16。
- インテル® Gaudi® 2 AI アクセラレーター (70B) の TGI パラメーター: `batch_bucket_size=22`、`prefill_batch_bucket_size=4`、`max_batch_prefill_tokens=5102`、`max_batch_total_tokens=32256`、`max_waiting_tokens=5`、`streaming=false`。
- H100 (70B) の TGI パラメーター: `batch_bucket_size=8`、`prefill_batch_bucket_size=4`、`max_batch_prefill_tokens=4096`、`max_batch_total_tokens=131072`、`max_waiting_tokens=20`、`max_batch_size=128`、`streaming=false`。
- 総所有コスト (TCO) の出典: https://www.databricks.com/blog/llm-training-and-inference-intel-gaudi2-ai-accelerators (英語)。