

# インテルの GPU を使用して YOLOv8 で 1000fps 越えを達成するには

この記事は、Medium に公開されている「[How to get YOLOv8 Over 1000 fps with Intel GPUs?](#)」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。

著者: Raymond Lo インテル コーポレーション AI ソフトウェア・エバンジェリスト

OpenVINO™ とインテル® Arc™ A770m グラフィックスがあれば YOLOv8 で 1000fps 越えを達成できます!

GPU で AI 推論を実行することは新しいトピックではありません。最近では、AI のトレーニングと推論に GPU を使用するアプリケーションも多くなりました。では、新しいインテル® Arc™ グラフィックスを使用して同じことができるでしょうか? どうすれば良いのでしょうか?

答えは OpenVINO™ を使用することです。私が (インテルに入社する前に) 初めて AI に取り組んだとき、TensorFlow\* を使用して YOLO のような物体検出を実行しようとしてしました。そのときに気になったのは、最適化やハードウェア・アクセラレーション機能が備わっていない CPU のパフォーマンスでした。動作は非常に遅く、エンジンが 1 つのコアで動作していることがよく分かりました (非常に遅いわけです)。その当時、優れたリアルタイム・パフォーマンスを得るには高価なグラフィックス・カードが必要でした。現在では、[YOLOv8 と OpenVINO™](#) (英語) により、状況は大きく変わりました。CPU、iGPU、dGPU を、同じコードベースでシームレスに連携できます。これは素晴らしいことです。

この記事で覚えてほしいことは、[Neural Network Compression Framework \(ニューラル・ネットワーク圧縮フレームワーク、NNCF\)](#) (英語) でモデルを INT8 向けに圧縮および最適化して、[VNNI](#) (英語)、[インテル® AMX](#)、[インテル® XMV](#) (英語) のようなハードウェア・アクセラレーションに対応する方法です。

変換と最適化により、精度にほとんど影響を与えることなく、モデルは「軽量」で高速になります。

## Validate quantized model accuracy

As we can see, there is not significant difference between int8 and float model result in single image test. To understand how quantization influence on model prediction precision, we can compare model accuracy on dataset.

```
[27]: int8_stats = test(quantized_model, core, tqdm(data_loader))
```

100% ██████████ 5000/5000 [03:10<00:00, 30.34it/s]

```
[28]: print("FP32 model accuracy")
print_stats(fp_stats, validator.seen, validator.nt_per_class.sum())

print("INT8 model accuracy")
print_stats(int8_stats, validator.seen, validator.nt_per_class.sum())
```

FP32 model accuracy							
Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5: .95	
all	5000	36335	0.633	0.474	0.521	0.371	
INT8 model accuracy							
Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5: .95	
all	5000	36335	0.633	0.474	0.519	0.369	

Great! Looks like accuracy was changed, but not significantly and fulfil passing criteria.

.. .

## シングルストリームの推論

第12世代インテル® Core™ プロセッサーを搭載したノートブックでは、(1つの1080p webcamで)45fpsでした。驚くべきことに、ノートブックは「フル稼働」状態ではなく、ほかの多くの推論タスクを実行する余裕があります。

どんな魔法を使ったのでしょうか？ その魔法とは、OpenVINO™ の [GPU プラグイン](#) (英語) によるデバイスの切り替え (`device = "GPU"`) でした。

```
run_object_detection(source=0, flip=True, use_popup=False, model=ov_model, device="GPU")
```



iGPUとOpenVINO™でYOLOv8を実行。かなりシームレスです。

インテル® Arc™ A770m グラフィックスに切り替えると、10% の負荷では GPU はほとんど使用されていません。



dGPU は、使用率が比較的低い状態で多くの推論を並列で実行できます。

## 1000fps 越えを達成するには?

ここで、記事のタイトルに戻ります。どうすれば 1000fps 越えを達成できるのでしょうか？ その秘訣は、スループット・モードでマルチストリームと複数の推論リクエストを使用する (つまり、複数を並列で実行する) ことです。例えば、"benchmark\_app" を使用して、モデルのスループットを計算できます。ボトルネックが発生しないように前処理パイプラインと後処理パイプラインを微調整する必要があります。

前置きはこれくらいにして、NUC の CPU -> iGPU -> dGPU のベンチマーク結果を示します。



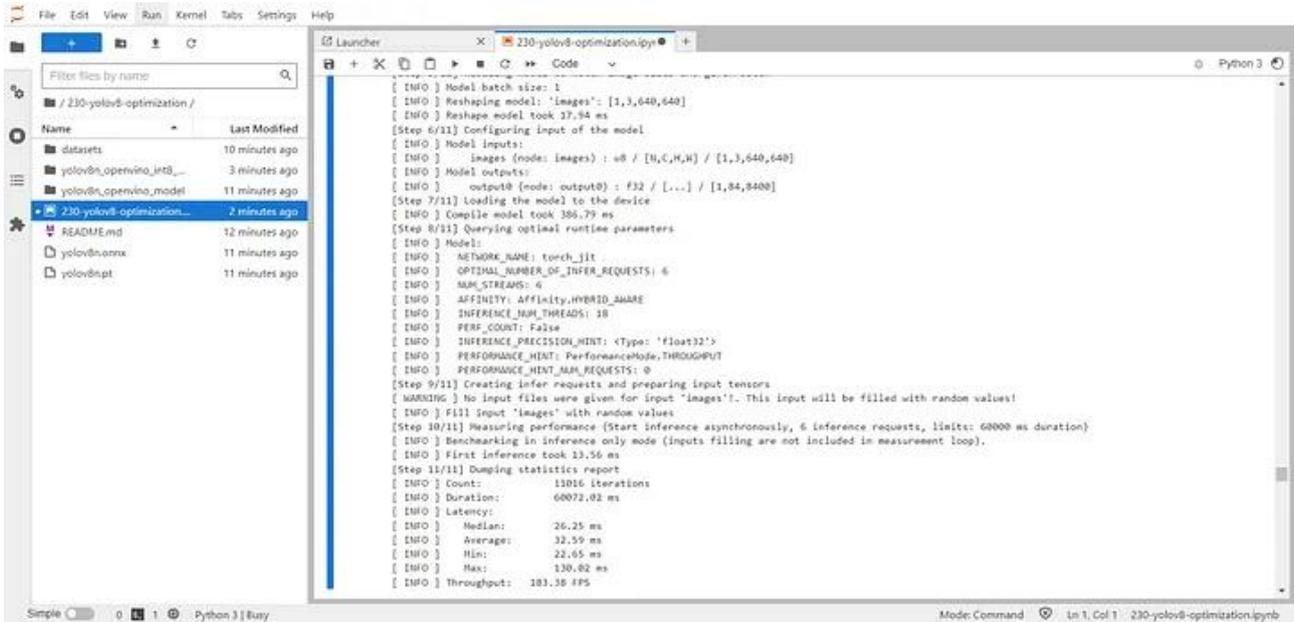
NUCのセットアップ

```
# Inference FP32 model (OpenVINO IR)  
!benchmark_app -m $model_path -d CPU -api async -shape "[1,3,640,640]"
```

```
[ INFO ] Model batch size: 1  
[ INFO ] Reshaping model: 'images': [1,3,640,640]  
[ INFO ] Reshape model took 12.96 ms  
[Step 6/11] Configuring input of the model  
[ INFO ] Model inputs:  
[ INFO ] images (node: images) : u8 / [N,C,H,W] / [1,3,640,640]  
[ INFO ] Model outputs:  
[ INFO ] output0 (node: output0) : F32 / [...] / [1,84,8400]  
[Step 7/11] Loading the model to the device  
[ INFO ] Compile model took 159.70 ms  
[Step 8/11] Querying optimal runtime parameters  
[ INFO ] Model:  
[ INFO ] NETWORK_NAME: torch jit  
[ INFO ] OPTIMAL_NUMBER_OF_INFER_REQUESTS: 6  
[ INFO ] NUM_STREAMS: 6  
[ INFO ] AFFINITY: Affinity.HYBRID_AWARE  
[ INFO ] INFERENCE_NUM_THREADS: 10  
[ INFO ] PERF_COUNT: False  
[ INFO ] INFERENCE_PRECISION_HINT: <Type: 'float32'>  
[ INFO ] PERFORMANCE_HINT: PerformanceMode.THROUGHPUT  
[ INFO ] PERFORMANCE_HINT_NUM_REQUESTS: 0  
[Step 9/11] Creating infer requests and preparing input tensors  
[ WARNING ] No input files were given for input "images"! This input will be filled with random values!  
[ INFO ] fill input "images" with random values  
[Step 10/11] Measuring performance (Start inference asynchronously, 6 Inference requests, limits: 60000 ms duration)  
[ INFO ] Benchmarking in inference only mode (inputs filling are not included in measurement loop).  
[ INFO ] First inference took 31.80 ms  
[Step 11/11] Dumping statistics report  
[ INFO ] Counts: 4554 iterations  
[ INFO ] Duration: 60098.94 ms  
[ INFO ] Latency:  
[ INFO ] Median: 78.36 ms  
[ INFO ] Average: 78.96 ms  
[ INFO ] Min: 55.58 ms  
[ INFO ] Max: 169.93 ms  
[ INFO ] Throughput: 75.78 FPS
```

CPU上のFP32モデルでは~75fps

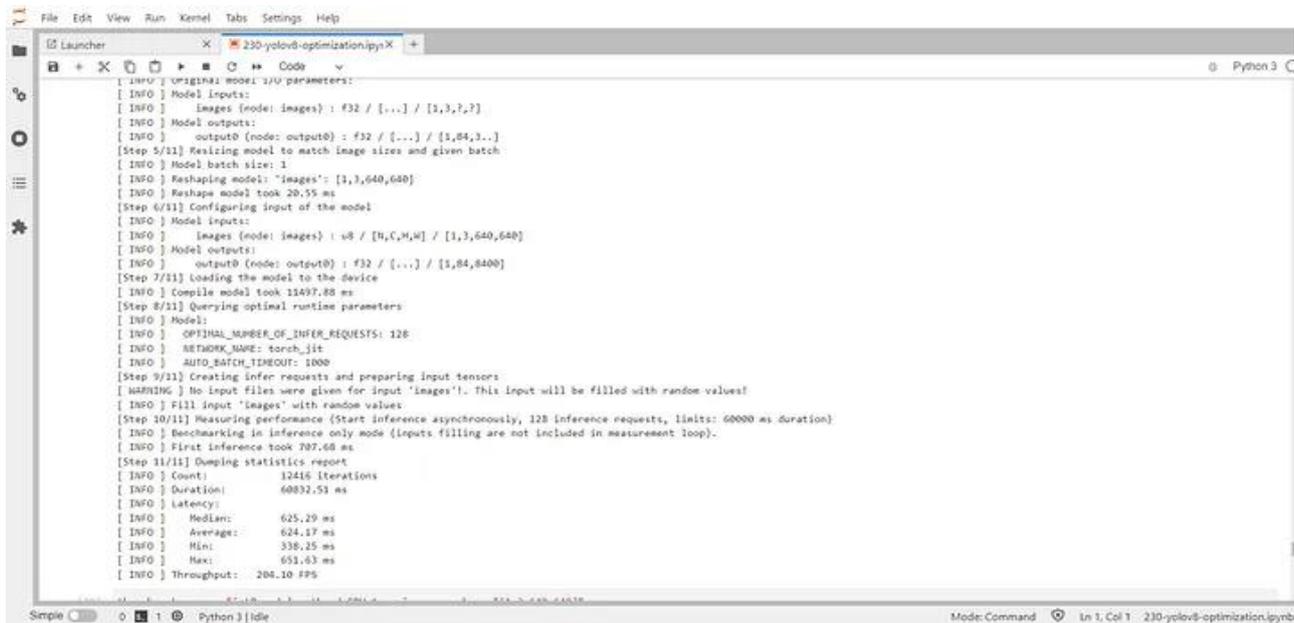
```
# Inference INT8 model (OpenVINO IR)
!benchmark_app -m $int8_model_path -d CPU -api async -shape "[1,3,640,640]"
```



```
[ INFO ] Model batch size: 1
[ INFO ] Reshaping model: 'images': [1,3,640,640]
[ INFO ] Reshape model took 37.94 ms
[Step 6/11] Configuring input of the model
[ INFO ] Model inputs:
[ INFO ]   images (node: images) : u8 / [N,C,H,W] / [1,3,640,640]
[ INFO ] Model outputs:
[ INFO ]   output0 (node: output0) : f32 / [...] / [1,84,840]
[Step 7/11] Loading the model to the device
[ INFO ] Compile model took 396.79 ms
[Step 8/11] Querying optimal runtime parameters
[ INFO ] Model:
[ INFO ]   NETWORK_NAME: torch_jit
[ INFO ]   OPTIMAL_NUMBER_OF_INFER_REQUESTS: 6
[ INFO ]   NUM_STREAMS: 6
[ INFO ]   AFFINITY: Affinity:HYBRID_AWARE
[ INFO ]   INFERENCE_NUM_THREADS: 18
[ INFO ]   PEP3COUNT: False
[ INFO ]   INFERENCE_PRECISION_HINT: <Type: 'float32'>
[ INFO ]   PERFORMANCE_HINT: PerformanceMode:THROUGHPUT
[ INFO ]   PERFORMANCE_HINT_NUM_REQUESTS: 0
[Step 9/11] Creating infer requests and preparing input tensors
[ WARNING ] No input files were given for input 'images'. This input will be filled with random values!
[ INFO ] Fill input 'images' with random values
[Step 10/11] Measuring performance (Start inference asynchronously, 6 inference requests, limit: 60000 ms duration)
[ INFO ] Benchmarking in inference only mode (inputs filling are not included in measurement loop).
[ INFO ] First inference took 13.56 ms
[Step 11/11] Dumping statistics report
[ INFO ] Count: 11016 iterations
[ INFO ] Duration: 60072.02 ms
[ INFO ] Latency:
[ INFO ]   Median: 26.25 ms
[ INFO ]   Average: 32.59 ms
[ INFO ]   Min: 22.65 ms
[ INFO ]   Max: 130.62 ms
[ INFO ] Throughput: 183.38 FPS
```

CPU 上の INT8 モデルでは ~183fps

```
!benchmark_app -m $int8_model_path -d GPU.0 -api async -shape "[1,3,640,640]"
```

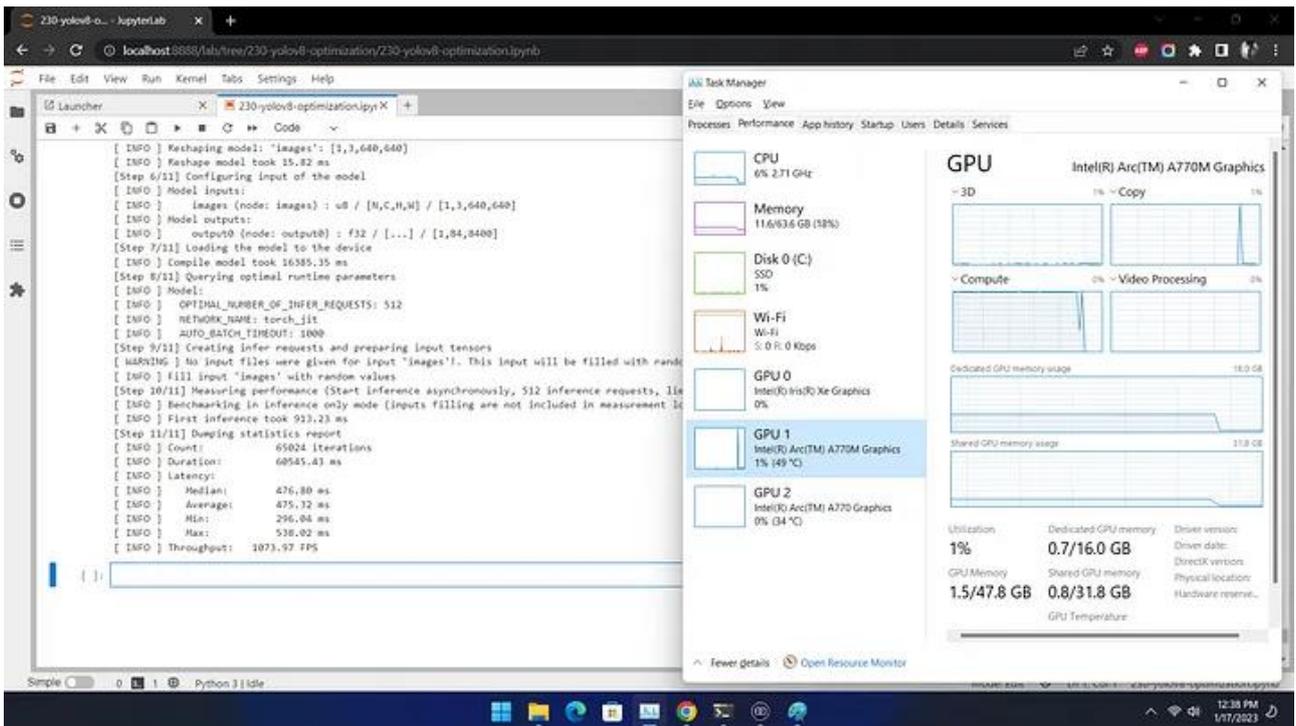


```
[ INFO ] Unrecognized model I/O parameters:
[ INFO ] Model inputs:
[ INFO ]   images (node: images) : f32 / [...] / [1,3,7,7]
[ INFO ] Model outputs:
[ INFO ]   output0 (node: output0) : f32 / [...] / [1,84,1...]
[Step 5/11] Resizing model to match image sizes and given batch
[ INFO ] Model batch size: 1
[ INFO ] Reshaping model: 'images': [1,3,640,640]
[ INFO ] Reshape model took 20.55 ms
[Step 6/11] Configuring input of the model
[ INFO ] Model inputs:
[ INFO ]   images (node: images) : u8 / [N,C,H,W] / [1,3,640,640]
[ INFO ] Model outputs:
[ INFO ]   output0 (node: output0) : f32 / [...] / [1,84,840]
[Step 7/11] Loading the model to the device
[ INFO ] Compile model took 11497.88 ms
[Step 8/11] Querying optimal runtime parameters
[ INFO ] Model:
[ INFO ]   OPTIMAL_NUMBER_OF_INFER_REQUESTS: 128
[ INFO ]   NETWORK_NAME: torch_jit
[ INFO ]   AUTO_BATCH_TIMEOUT: 1000
[Step 9/11] Creating infer requests and preparing input tensors
[ WARNING ] No input files were given for input 'images'. This input will be filled with random values!
[ INFO ] Fill input 'images' with random values
[Step 10/11] Measuring performance (Start inference asynchronously, 128 inference requests, limit: 60000 ms duration)
[ INFO ] Benchmarking in inference only mode (inputs filling are not included in measurement loop).
[ INFO ] First inference took 787.68 ms
[Step 11/11] Dumping statistics report
[ INFO ] Count: 12416 iterations
[ INFO ] Duration: 60032.53 ms
[ INFO ] Latency:
[ INFO ]   Median: 625.29 ms
[ INFO ]   Average: 624.17 ms
[ INFO ]   Min: 338.25 ms
[ INFO ]   Max: 651.63 ms
[ INFO ] Throughput: 204.10 FPS
```

iGPU 上では ~204fps

```
!benchmark_app -m $int8_model_path -d GPU.1 -api async -shape "[1,3,640,640]"
```

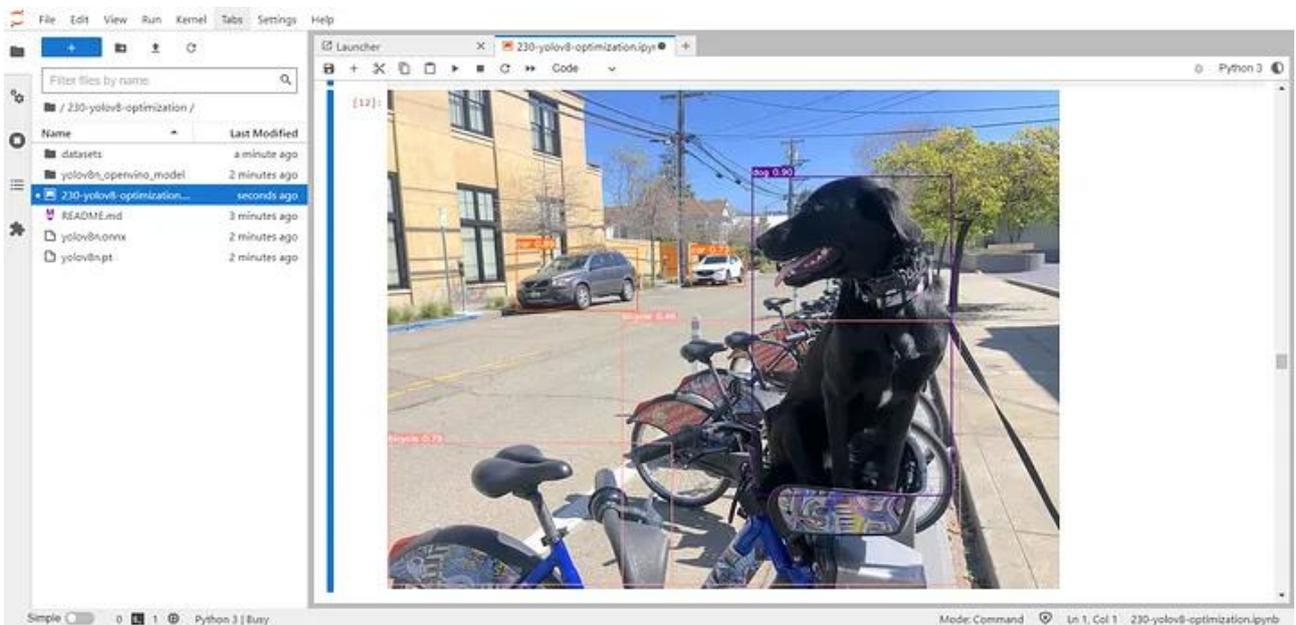
準備はいいですか?



なんと、インテル® Arc™ A770m グラフィックス上の INT8 モデルでは 1073.97fps になりました!

1000fps 越え達成です!

途中の過程を詳しく知りたい場合は、[OpenVINO™ ノートブック \(英語\)](#) のコードを確認してください。皆さんのコメントを歓迎します。



最適化ツールを実行し、結果をプレビューしてコーディング方法を学ぶこともできます。

試してみてください、結果をお知らせください。コーディングを楽しみましょう!

#iamintel

## OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピュータービジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン/ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

<https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html>

### 法務上の注意書き

インテルのテクノロジーを使用するには、対応したハードウェア、ソフトウェア、またはサービスの有効化が必要となる場合があります。

性能は、使用状況、構成、その他の要因によって異なります。

性能の測定結果はシステム構成の日付時点のテストに基づいています。また、現在公開中のすべてのセキュリティ・アップデートが適用されているとは限りません。詳細は、システム構成を参照してください。絶対的なセキュリティを提供できる製品またはコンポーネントはありません。

インテルの最適化機能は、インテルのコンパイラまたはその他のインテル製品を対象としたものであり、他社製品に同等の最適化を行えないことがあります。

インテルは、サードパーティーのデータについて管理や監査を行っていません。ほかの情報も参考にして、正確かどうかを評価してください。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。

\* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

本資料に記載されているインテル製品に関する侵害行為または法的調査に関連して、本資料を使用または使用を促すことはできません。本資料を使用することにより、お客様は、インテルに対し、本資料で開示された内容を含む特許クレームで、その後に作成したものについて、非独占的かつロイヤルティ無料の実施権を許諾することに同意することになります。

本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。唯一の例外として、本資料に含まれるコードは、<http://opensource.org/licenses/0BSD> (英語) のとおり、ゼロ条項 BSD オープンソース・ライセンス (0BSD) の対象になるものとします。