Phi-3.5-Mini-Instruct ONNX models

This repository hosts the optimized versions of Phi-3.5-mini-instruct to accelerate inference with ONNX Runtime. Optimized Phi-3.5 Mini models are published here in ONNX format to run with ONNX Runtime on CPU and GPU across devices, including server platforms, Windows, Linux and Mac desktops, and mobile CPUs, with the precision best suited to each of these targets.

To easily get started with Phi-3.5, you can use our newly introduced ONNX Runtime Generate() API. See here for instructions on how to run it.

ONNX Models

Here are some of the optimized configurations we have added:

1. ONNX model for fp16 CUDA: ONNX model you can use to run for your NVIDIA GPUs.
2. ONNX model for int4 CUDA: ONNX model for NVIDIA GPUs using int4 quantization via AWQ.
3. ONNX model for int4 CPU and Mobile: ONNX model for CPU and mobile using int4 quantization via AWQ.

Model Summary

Phi-3.5-mini is a lightweight, state-of-the-art open model built upon datasets used for Phi-3 - synthetic data and filtered publicly available websites - with a focus on very high-quality, reasoning dense data. The model belongs to the Phi-3 model family and supports 128K token context length. The model underwent a rigorous enhancement process, incorporating both supervised fine-tuning, proximal policy optimization, and direct preference optimization to ensure precise instruction adherence and robust safety measures.

Intended Uses

The Phi 3.5 model is intended for commercial and research use in multiple languages. The model provides uses for general purpose AI systems and applications which require:

1. Memory/compute constrained environments
2. Latency bound scenarios
3. Strong reasoning (especially code, math and logic)

Use Case Considerations

Phi 3.5 models are not specifically designed or evaluated for all downstream purposes. Developers should consider common limitations of language models as they select use cases, and evaluate and mitigate for accuracy, safety, and fariness before using within a specific downstream use case, particularly for high risk scenarios. Developers should be aware of and adhere to applicable laws or regulations (including privacy, trade compliance laws, etc.) that are relevant to their use case. Nothing contained in this Model Card should be interpreted as or deemed a restriction or modification to the license the model is released under.

Release Notes

This is an update over the instruction-tuned Phi-3 Mini ONNX model release. We believe most use cases will benefit from this release, but we encourage users to test their particular AI applications. We appreciate the enthusiastic adoption of the Phi-3 model family and continue to welcome all feedback from the community.

Hardware Supported

The ONNX models are tested on:

- GPU SKU: RTX 4090 (DirectML)
- GPU SKU: 1 A100 80GB GPU, SKU: Standard_ND96amsr_A100_v4 (CUDA)
- CPU SKU: Standard D16s v6 (16 vcpus, 64 GiB memory)
- AMD CPU: Internal_D64as_v5

Minimum Configuration Required:

- Windows: DirectX 12-capable GPU and a minimum of 4GB of combined RAM
- CUDA: NVIDIA GPU with Compute Capability >= 7.0

Model Description

- **Developed by:** Microsoft
- **Model type:** ONNX
- **Language(s) (NLP):** Python, C, C++
- **License:** MIT
- **Model Description:** This is a conversion of the Phi-3.5 Mini-Instruct model for ONNX Runtime inference.

How to Get Started with the Model

To make running of the Phi-3 models across a range of devices and platforms across various execution provider backends possible, we introduce a new API to wrap several aspects of generative AI inferencing. This API make it easy to drag and drop LLMs straight into your app. For running the early version of these models with ONNX Runtime, follow the steps here.

For example:

```
python model-qa.py -m /*{YourModelPath}*/Phi-3.5-mini-instruct-
onnx/cpu_and_mobile/cpu-int4-awq-block-128-acc-level-4 -k 40 -p 0.95 -t 0.8 -r
1.0
```

```
*Input:*  <|user|>Tell me a joke<|end|><|assistant|>

*Output:*  Why don't scientists trust atoms?
           Because they make up everything!

This joke plays on the double meaning of "make up." In science, atoms are the
fundamental building blocks of matter, literally making up everything. However,
in a colloquial sense, "to make up" can mean to fabricate or lie, hence the
humor.
```

Performance Metrics

Phi-3.5 Mini-Instruct performs better in ONNX Runtime than PyTorch for all batch size, prompt length combinations.

The table below shows the average throughput of the first 256 tokens generated (tps) for FP16 and INT4 precisions on CUDA as measured on 1 A100 80GB GPU, SKU: Standard_ND96amsr_A100_v4. ONNX Runtime

models for GPU are 21X faster than PyTorch Compile and up to 8X faster than llama.cpp on A100 GPU.

Batch Size, Sequence Length	ONNX RT INT4	PyTorch Eager INT4	PyTorch Compile INT4	Llama.cpp INT4	INT4 SpeedUp ORT/PyTorch Eager	INT4 SpeedUp ORT/PyTorch Compile	INT4 SpeedUp ORT/Llama.cpp
1, 16	238.97	17.75	11.36	183.17	13.46	21.04	1.30
1, 64	233.74	17.74	11.32	182.77	13.17	20.65	1.28
1, 256	208.52	17.82	11.34	182.15	11.70	18.38	1.14
1, 1024	174.19	17.85	11.36	166.39	9.76	15.34	1.05
1, 2048	146.10	17.96	11.35	153.50	8.14	12.87	0.95
1, 3840	112.68	17.91	11.34	141.53	6.29	9.94	0.80
4, 16	286.73	60.90	40.89	180.82	4.71	7.01	1.59
4, 64	282.87	60.88	41.03	177.69	4.65	6.89	1.59
4, 256	268.30	60.85	40.90	166.34	4.41	6.56	1.61
4, 1024	223.30	60.86	40.90	133.39	3.67	5.46	1.67
4, 2048	187.62	60.80	40.93	106.03	3.09	4.58	1.77
4, 3840	145.59	55.96	40.88	78.12	2.60	3.56	1.86
8, 16	541.04	121.92	81.96	171.90	4.44	6.60	3.15
8, 64	532.68	121.87	81.98	166.33	4.37	6.50	3.20
8, 256	480.00	122.06	81.80	148.07	3.93	5.87	3.24
8, 1024	360.60	122.48	81.59	103.58	2.94	4.42	3.48
8, 2048	274.16	105.92	81.71	74.01	2.59	3.36	3.70
8, 3840	192.50	79.74	81.50	49.23	2.41	2.36	3.91
16, 16	1007.69	244.16	163.09	156.99	4.13	6.18	6.42
16, 64	966.42	244.89	163.26	148.23	3.95	5.92	6.52
16, 256	827.37	244.84	163.23	121.85	3.38	5.07	6.79
16, 1024	536.73	209.13	169.30	71.57	2.57	3.17	7.50
16, 2048	375.31	153.95	158.77	45.97	2.44	2.36	8.16
16, 3840	243.66	OOM	OOM	28.33			8.60

Batch Size, Sequence Length	ONNX RT FP16	PyTorch Eager FP16	PyTorch Compile FP16	Llama.cpp	FP16 SpeedUp ORT/PyTorch Eager	FP16 SpeedUp ORT/PyTorch Compile	FP16 SpeedUp ORT/Llama.cpp
1, 16	137.30	26.02	26.83	125.86	5.28	5.12	1.09

Batch Size, Sequence Length	ONNX RT FP16	PyTorch Eager FP16	PyTorch Compile FP16	Llama.cpp	FP16 SpeedUp ORT/PyTorch Eager	FP16 SpeedUp ORT/PyTorch Compile	FP16 SpeedUp ORT/Llama.cpp
1, 64	135.79	26.01	26.48	125.75	5.22	5.13	1.08
1, 256	127.92	26.17	26.61	125.24	4.89	4.81	1.02
1, 1024	114.08	26.11	26.63	117.97	4.37	4.28	0.97
1, 2048	101.68	17.77	21.05	111.08	5.72	4.83	0.92
1, 3840	84.94	25.17	26.77	104.88	3.37	3.17	0.81
4, 16	529.07	99.47	100.22	124.63	5.32	5.28	4.25
4, 64	513.85	99.47	100.54	123.20	5.17	5.11	4.17
4, 256	466.56	99.21	100.22	117.61	4.70	4.66	3.97
4, 1024	352.06	99.56	100.50	100.42	3.54	3.50	3.51
4, 2048	271.02	70.12	73.66	83.95	3.86	3.68	3.23
4, 3840	191.36	74.35	79.68	65.51	2.57	2.40	2.92
8, 16	936.46	198.99	212.40	120.24	4.71	4.41	7.79
8, 64	926.83	200.28	213.97	117.77	4.63	4.33	7.87
8, 256	783.95	200.66	214.88	108.33	3.91	3.65	7.24
8, 1024	511.96	183.10	201.01	82.52	2.80	2.55	6.20
8, 2048	352.86	96.99	122.10	62.41	3.64	2.89	5.65
8, 3840	228.97	96.81	101.60	43.89	2.37	2.25	5.22
16, 16	1675.72	396.52	422.13	112.78	4.23	3.97	14.86
16, 64	1591.61	395.66	422.47	108.36	4.02	3.77	14.69
16, 256	1249.94	399.30	429.10	93.68	3.13	2.91	13.34
16, 1024	685.63	270.99	292.24	60.66	2.53	2.35	11.30
16, 2048	441.15	121.17	162.93	41.30	3.64	2.71	10.68
16, 3840	270.38	OOM	OOM	26.50	0.00	0.00	10.20

The table below shows the average throughput of the first 256 tokens generated (tps) for INT4 precision on CPU as measured on a Standard D16s v6 (16 vcpus, 64 GiB memory)

Batch Size, Sequence Length	ORT INT4 AWQ	Llama.cpp INT4	INT4 AWQ SpeedUp Llama.cpp
1, 16	41.99	26.72	1.57
1, 64	41.81	26.67	1.57
1, 256	41.26	26.30	1.57

Batch Size, Sequence Length	ORT INT4 AWQ	Llama.cpp INT4	INT4 AWQ SpeedUp Llama.cpp
1, 1024	37.15	24.02	1.55
1, 2048	32.68	21.82	1.50

Package Versions

Pip package name	Version
torch	2.4.1
triton	3.0.0
onnxruntime-gpu	1.19.2
onnxruntime-genai	0.4.0
onnxruntime-genai-cuda	0.4.0
transformers	4.44.2
llama.cpp	bdf314f38a2c90e18285f7d7067e8d736a14000a

Appendix

Activation Aware Quantization (AWQ) works by identifying the top 1% most salient weights that are most important for maintaining accuracy and quantizing the remaining 99% of weights. This leads to less accuracy loss from quantization compared to many other quantization techniques. For more on AWQ, see here.

Model Card Contact

parinitarahi

Contributors

Sunghoon Choi, Yufeng Li, Kunal Vaishnavi, Akshay Sonawane, Rui Ren, Parinita Rahi

License

The model is licensed under the MIT license.

Trademarks

This project may contain trademarks or logos for projects, products, or services. Authorized use of Microsoft trademarks or logos is subject to and must follow Microsoft's Trademark & Brand Guidelines. Use of Microsoft trademarks or logos in modified versions of this project must not cause confusion or imply Microsoft sponsorship. Any use of third-party trademarks or logos are subject to those third-party's policies.