**LLM on FPGA (Documentation)**

**Large Language Models (LLMs)**, such as the popular **Transformer-based models**, have revolutionized various domains by achieving impressive performance in natural language understanding and generation tasks. However, deploying these models efficiently is a challenge due to their heavy computation and memory overheads.

Here’s why we use LLMs on **Field-Programmable Gate Arrays (FPGAs)**:

1. **Efficiency Enhancement**:
   * **LLMs** require significant computational resources for inference. FPGAs offer an opportunity to **accelerate** LLM inference by leveraging their parallelism and customizability.
   * [**FlightLLM**, for instance, enables efficient LLM inference on FPGAs by utilizing FPGA-specific resources like **DSP48** blocks and heterogeneous memory hierarchy1](https://arxiv.org/pdf/2401.03868.pdf).
   * The goal is to achieve higher energy efficiency and better cost efficiency compared to commercial GPUs.
2. **Challenges Addressed**:
   * **Low Computation Efficiency**:
     + LLMs exhibit various sparsity patterns (e.g., block sparsity, N:M sparsity), leading to inefficient computations.
     + FPGAs can address this by supporting configurable sparse **DSP chains** for different sparsity patterns.
   * **Underutilized Memory Bandwidth**:
     + LLMs repetitively access fine-grained data during decoding.
     + FPGAs can boost memory bandwidth using an **always-on-chip decode scheme** with mixed-precision support.
   * **Compilation Overheads**:
     + To reduce compilation overhead, **length-adaptive compilation methods** are proposed for real-world LLMs.
3. **Performance Comparison**:
   * **FlightLLM** implemented on the **Xilinx Alveo U280 FPGA** achieves:
     + **6.0× higher energy efficiency** and **1.8× better cost efficiency** compared to commercial GPUs (e.g., NVIDIA V100S) for modern LLMs (e.g., LLaMA2-7B) using techniques like **vLLM** and **SmoothQuant**.
     + [**1.2× higher throughput** than NVIDIA A100 GPU using the latest **Versal VHK158 FPGA**1](https://arxiv.org/pdf/2401.03868.pdf).

* [Using FPGAs for LLM inference allows us to strike a balance between computational efficiency, memory bandwidth utilization, and compilation overheads, resulting in improved performance and cost-effectiveness](https://arxiv.org/pdf/2401.03868.pdf).

**COMPARISION BETWEEN CPU,GPU and FPGA:**

**CPUs**, **GPUs**, and **FPGAs** in the context of working with **Large Language Models (LLMs)**.

| **Aspect** | **CPU** | **GPU** | **FPGA** |
| --- | --- | --- | --- |
| **Parallelism** | Limited parallelism. Suitable for general-purpose tasks. | High parallelism with thousands of cores. Ideal for matrix operations. | Customizable parallelism. Can be tailored to specific LLM workloads. |
| **Memory Hierarchy** | Hierarchical memory (cache, RAM). | High memory bandwidth. GDDR for GPU. | Customizable memory hierarchy. Can optimize for LLM data access patterns. |
| **Latency** | Higher latency due to sequential execution. | Lower latency due to parallelism. | Low latency with custom data paths. |
| **Throughput** | Moderate throughput for serial tasks. | High throughput for parallel tasks. | Configurable throughput based on design. |
| **Energy Efficiency** | Efficient for serial workloads. | Less energy-efficient due to high power consumption. | Energy-efficient when optimized for specific tasks. |
| **Customization** | Not customizable beyond software optimizations. | Limited customization (e.g., CUDA kernels). | Highly customizable using hardware description languages. |
| **Ease of Programming** | High-level languages (C/C++, Python). | CUDA, OpenCL, or specialized libraries. | Requires hardware description languages (VHDL, Verilog). |
| **Deployment Cost** | Affordable, widely available. | Expensive upfront cost. | Moderate upfront cost, but can be cost-effective for specific workloads. |
| **Scalability** | Limited scalability for parallel tasks. | Scalable for parallel tasks across multiple GPUs. | Scalable with custom designs for large-scale deployment. |

**What is hugging face?**

Hugging Face aids in Large Language Model (LLM) development and usage through several means:

Model Repository: Provides a platform to host and discover LLMs, facilitating easy access to pretrained models for tasks like text generation, translation, and summarization.

Dataset Sharing: Offers a wide range of datasets relevant to language tasks, enabling researchers to train and fine-tune LLMs on diverse data sources.

Compute Solutions: Offers compute solutions, including GPU support, for training and inference tasks, reducing the computational burden on researchers and developers.

Transformers Library: The Transformers library, a key open-source project by Hugging Face, provides state-of-the-art LLM architectures and pre-trained models for PyTorch, TensorFlow, and JAX, streamlining LLM development.

Tokenizers: Fast tokenizers provided by Hugging Face enhance the efficiency of LLM training and inference, especially for large-scale language datasets.

Open Source Contributions: Continuously contributes to the advancement of LLM research through open-source projects like Transformers, offering cutting-edge architectures and methodologies for natural language processing tasks.

**What is ONNX Model?**

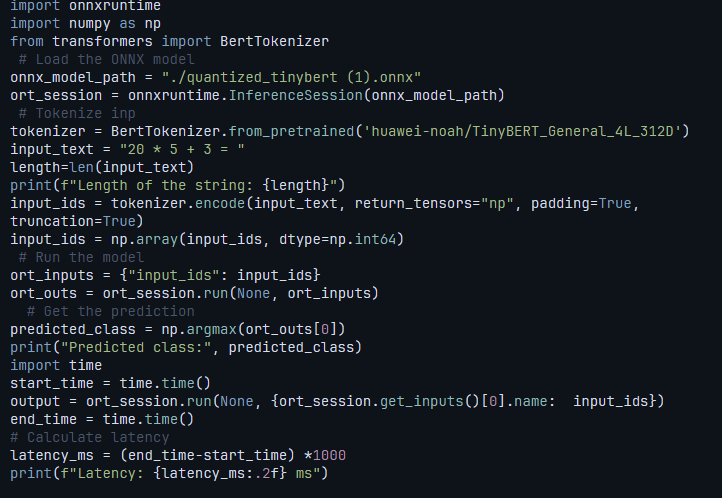
ONNX (Open Neural Network Exchange) serves as a vital bridge between machine learning frameworks, converting models seamlessly and providing a universal format for representation. Users can train models in frameworks like PyTorch or TensorFlow and convert them to ONNX for versatile deployment. Benefits include framework agnosticism, facilitating compatibility, transferability without retraining, and standardization of computational graphs. Overall, ONNX streamlines model development and deployment, enhancing collaboration and interoperability in the machine learning community.

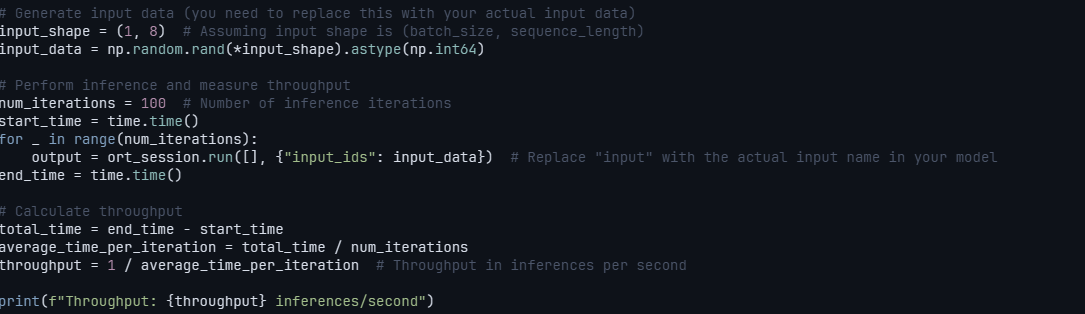
Using ONNX in Large Language Models (LLMs):

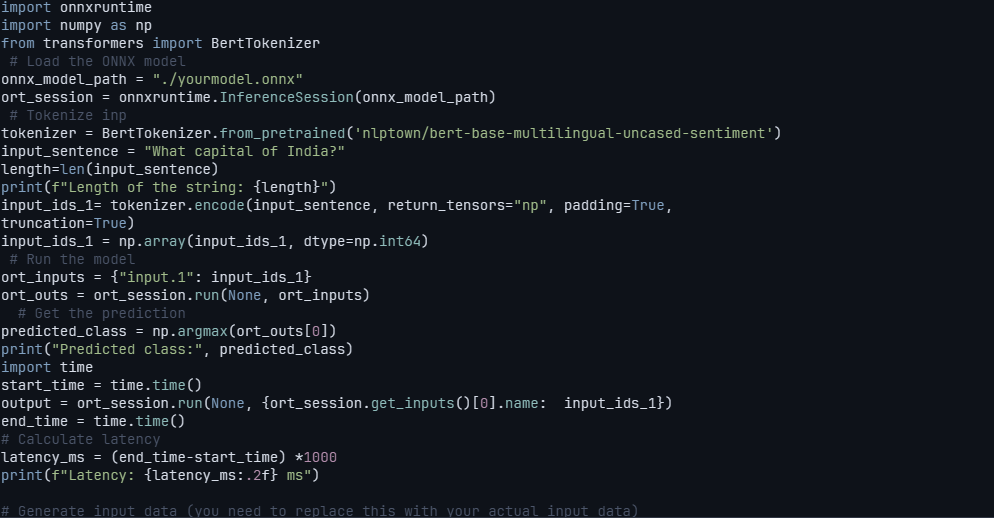
* Train your LLM in a framework like PyTorch or TensorFlow.
* Convert the trained LLM model into the ONNX format.
* Deploy the ONNX LLM model in different frameworks for inference, such as serving it with TensorRT or TFLite for optimized performance.

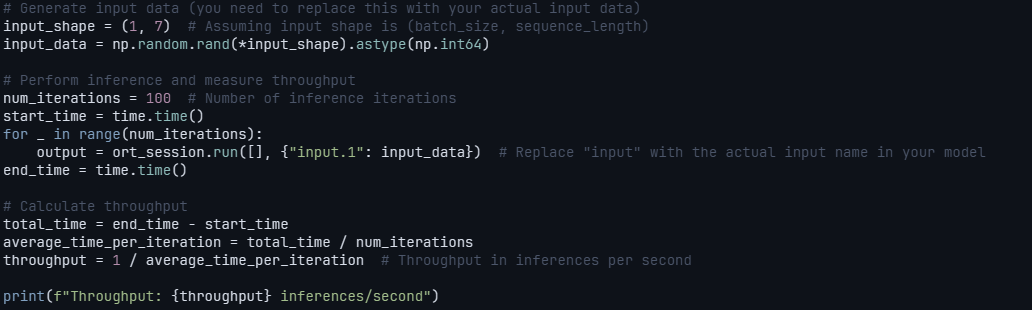
**Example code of the model used for analysis:**

**Model1:TinyBert model:(using ONNX)**



****

**Model2:Sentimental analysis model(using ONNX):**

****

**Result analysis:** both models run on 2 different input

**CPU performance:**

Input 1: "What is capital of India?"

Length of the string:25

|  |  |  |
| --- | --- | --- |
| Model | latency | throughput |
| Tinybert(hugging face) | 177.28829383850098 ms | 69.49934109656841 inferences/second |
| Tinybert(ONNX) | 2.58 m  s | 405.21309340479917 inferences/second |
| Sentimental analysis model(hugging face) | 281.29 ms | 4.53 examples/second |
| Sentimental analysis model(ONNX)  (Input: "What capital of India?") | 22.03 ms | 26.907246461045794 inferences/second |

Input 2: "20 \* 5 + 3 = "

Length of the string:13

|  |  |  |
| --- | --- | --- |
| Model | latency | throughput |
| Tinybert(hugging face) | 232.57684707641602 ms | 4.947033484118066 inferences/second |
| Tinybert(ONNX) | 2.44ms | 535.9578113958811 inferences/second |
| Sentimental analysis model(hugging face) | 232.16ms | 4.25 examples/second |
| Sentimental analysis model(ONNX)  (input: "20 \* 5 = ?") | 32.34 ms | 19.217632388847985 inferences/second |

**GPU performance:**

Input 1: "What is capital of India?"

Length of the string:25

|  |  |  |
| --- | --- | --- |
| Model | latency | throughput |
| Tinybert(hugging face) | 17.5778865814209 ms | 47.60965005130096 inferences/second |
| Tinybert(ONNX) | 1.96 ms | 635.5063833913642 inferences/second |
| Sentimental analysis model(hugging face) | 117.47 ms | 8.29 examples/second |
| Sentimental analysis model(ONNX)  (Input: "What capital of India?") | 13.83 ms | 35.132764414781164 inferences/second |

Input 5: "20 \* 5 + 3 = "

Length of the string:13

|  |  |  |
| --- | --- | --- |
| Model | latency | throughput |
| Tinybert(hugging face) | 5.8956146240234375 ms | 100.99905726597501 inferences/second |
| Tinybert(ONNX) | 1.69 ms | 629.9304933947699 inferences/second |
| Sentimental analysis model(hugging face) | 123.08 ms | 8.48 examples/second |
| Sentimental analysis model(ONNX)  (input: "20 \* 5 = ?") | 10.99 ms | 60.32213192935143 inferences/second |