Google Summer of Code

Proposal

Multimodal embeddings based on OVMS

Raghav Yadav

OpenVINO™

Table of Contents

Introduction

I am Raghav Yadav, a computer engineering student at the University Of Toronto, currently enrolled in my second year at the university. At present I am residing in the Eastern Time Zone (ET).

I have a keen interest in deep learning, reinforcement learning and distributed systems. Besides that I enjoy competitive programming and doing hackathons.

With a strong background in Python and C++, my development experience spans in deep learning, natural language processing (NLPs) and system optimization. In Python, my focus primarily lies in transformer-based models, and I have extensive development experience with Hugging Face's transformer library, as well vector databases like pinecone and gen ai frameworks like langchain.

In C++, my work includes deep learning inference optimization and low level memory management. In addition to this I have built high-performance socket based servers in C++.

I have built and deployed several Machine Learning (ML) and Deep Learning (DL) systems across a range of domains, with a focus on multimodal AI, model optimization, and inference acceleration. During my internship at Vector Institute, one of Canada's leading AI research institutes, I had the opportunity to work on state-of-the-art AI models and contribute to research in multimodal learning and model efficiency. This experience provided me with valuable insights into the theoretical foundations of AI, as well as hands-on experience applying research to solve real-world problems

One of my most significant projects involved developing a multimodal retrieval system, where I integrated transformer-based models from Hugging Face to process and retrieve text, image, and audio data. This allowed the system to understand and match queries across multiple modalities, providing a seamless experience for semantic search and cross-modal retrieval.

In the realm of computer vision, I created an image recognition system using YOLOv5 and ResNet, implementing transfer learning to optimize the models for real-world applications. I trained the model on large datasets and deployed it for use in a real-time object detection application. Additionally, I developed an audio classification pipeline utilizing Librosa and torchaudio, which enabled speech-to-text and audio feature extraction from noisy environments, ensuring the robustness of the model in practical settings.

With these projects, I have developed a strong foundation in AI model optimization and multimodal learning. I am excited to bring this experience to the OpenVINO Model Server project, where I can contribute to developing multimodal embedding retrieval endpoints and further optimize AI inference pipelines for both edge devices and cloud infrastructures.

About the Project

I have chosen to work on **Project 18: Multimodal Embeddings based on OpenVino Model Server (OVMS).** The project aims to develop an efficient system for computing multimodal embeddings, enabling semantic comparison across different data types such as text, images, video and audio. These embeddings play a fundamental role in operations such as classification and retrieval.

The reason why I am choosing this project is because it directly aligns with my profession in **machine learning (ML)**, **deep learning (DL)**, and **AI optimization**. I have worked on **multimodal systems** where I've integrated models for tasks such as **image recognition**, **audio classification**, and **text analysis**. These experiences have sparked my interest in creating more efficient systems that can seamlessly process and retrieve information across different modalities, such as text, images, and audio.

I am committed to dedicating **40-45 hours per week** to this project, maintaining structured and meaningful contributions. The primary focus would be on building and optimizing multimodal embedding endpoints while exploring other ways to enhance model efficiency, retrieval accuracy and real-world applicability. Moreover I will maintain a strong feedback loop with mentors to validate progress, and integrate improvements iteratively.

The solution is divided into modular stages to simplify development, improve clarity, and ensure a smooth and maintainable workflow.

1. Preprocessing: Standardizing Multimodal input data

The first stage involves preprocessing diverse data types-text, images, audio, and video. This would convert them into a format that can be processed by AI models.

1. Text Processing: Selecting the Transformer Model

To ensure efficient and accurate text embedding generation, I evaluated multiple transformer models based on key performance metrics:

I. **Inference Speed CPU/GPU**(time per prediction:128 tokens)
II. **Memory usage Inference CPU/GPU** (RAM consumption)

III. **Throughput** (requests per second)

The models in the test included:

```
 I.   bert-base-uncased (baseline)
 II.  roberta-base (robust NLP model)
III.  distilbert-base-uncased (lightweight, faster
      alternative)
 IV.  sentence-transformers/all-mpnet-base-v2 (balanced
      performance)
 V.   sentence-transformers/all-MiniLM-L6-v2 (optimized for
      efficiency)
```

The key performance results for each model:

Model	**Inference (sec) CPU:**	**Inference (sec) GPU:**	**Memory usage CPU (MB)**	**Memory usage GPU (MB)**	**GPU Throughput (Inf/sec)**
bert-base-uncased	1.4179	0.0161	321.14	433.05	62.11
roberta-base	0.9684	0.0124	319.46	491.05	80.65
distilbert-base-uncased	0.4849	0.0050	160.77	268.56	200.00
sentence-transformers/all-mpnet-base-v2	0.5070	0.0147	321.68	433.79	68.03
sentence-transformers/all-MiniLM-L6-v2	0.0688	0.0048	41.07	98.10	208.33

*__Link to the colab notebook:__ text.cpu, text.gpu

Based on the results for each models the potential models for the project can be:

- **For speed and efficiency: all-MiniLM-L6-v2** (best for quick inference and low resource usage)

- **For balanced performance: distilbert-base-uncased** (good balance of speed and accuracy)

2. Image Processing:

1. Preprocessing Pipeline

A standardized preprocessing pipeline will be applied to ensure consistency across different models. The preprocessing steps include:

- **Resizing** – Images resized to model-specific dimensions (e.g., 224×224 for ViTs).
- **Normalization** – Pixel values normalized to match the expected input range (e.g., [0,1] or [-1,1]).
- **Format Conversion** – Ensuring correct color format (RGB/BGR) and tensor structure.
- **Augmentation (Optional**) – Testing minor transformations (cropping, flipping) for robustness.

This pipeline will be implemented using OpenCV, PIL, and TorchVision transforms to ensure compatibility with Hugging Face transformers and OpenVINO Model Server (OVMS).

2. Model Selection & Benchmarking

To ensure seamless integration into the project and optimal performance, I tested multiple transformer-base vision models on the same metrics used for text embeddings.

Models in the test included:

```
 I.   google/vit-base-patch16-224 → Standard Vision
      Transformer
II.   facebook/dino-vits16 → Self-supervised learning,
      great for feature extraction
III.  microsoft/resnet-50 → Hybrid CNN-Transformer
      approach, optimized for speed
IV.   nvidia/mit-b0 → Efficient Transformer-based
      model for segmentation tasks
```

***Link to colab notebook:** Images.cpu, Images.gpu

The results were as follows:

Model	**Inference (cpu) sec**	**Inference (gpu) sec**	**Memory usage CPU (MB)**	**Memory usage GPU (MB)**	**Throughput (Images/sec)**
google/vit-base-patch16-224	0.9282	0.0094	332.48	672.25	**Batch 1:** 66.58 **Batch 4:** 94.62 **Batch 8:** 105.33 **Batch 16:** 103.49
facebook/dino-vits16	0.2251	0.0081	51.94	424.03	**Batch 1:** 122.54 **Batch 4:** 316.67 **Batch 8:** 331.55 **Batch 16:** 347.54
microsoft/resnet-50	0.2121	0.0072	121.06	430.16	**Batch 1:** 133.16 **Batch 4:** 301.86 **Batch 8:** 336.59 **Batch 16:** 377.84
nvidia/mit-b0	0.3071	0.0088	38.63	354.67	**Batch 1:** 49.58 **Batch 4:** 265.82 **Batch 8:** 911.87 **Batch 16:** 1057.18

Based on performance evaluation, **MIT-B0** (high **throughput: 1057.18 images/sec**) and **ResNet-50** (fastest **inference: 0.0072 sec GPU**) emerge as **potential transformer models** for this project. They offer the best balance of speed, efficiency, and scalability, making them ideal for multimodal embedding generation.

3. Audio Processing:

1. Preprocessing Pipeline

To generate **audio embeddings**, raw audio data must be transformed into a structured format that deep learning models can process. This involves extracting key features like **Mel spectrograms** and **MFCCs (Mel-Frequency Cepstral Coefficients)**.

For this, we considered two approaches:

- **Librosa:** A lightweight library for audio feature extraction. It is simple to use but **limited to CPU processing**, making it inefficient for large-scale inference.

- **PyTorch-based Pipelines:** Offers **GPU acceleration**, integrates well with Hugging Face models, and is more **scalable** for OpenVINO optimization.

Given these advantages, **PyTorch-based preprocessing** is chosen to ensure efficient and optimized feature extraction.

2. Model selection & Benchmarking

For this project, we have shortlisted Wav2Vec2 and HuBERT as potential transformer models for extracting audio embeddings.Unlike text and vision models, where benchmarking was crucial for selecting the best-performing transformer, we have pre-selected these two models based on their proven efficiency and practical considerations. This allows integration and optimization rather than extensive benchmarking tests.

4. Video Processing:

1. Preprocessing Pipeline

A standardized preprocessing pipeline will be applied to ensure consistency across different models. The preprocessing steps include:

- **Frame Extraction** – Videos will be sampled at a fixed frame rate to ensure uniformity in temporal representation.
- **Preprocessing Frames** – Extracted frames will be resized and normalized to match model input requirements.
- **Temporal Structuring** – To preserve motion dynamics, frames will be grouped into sequences before feeding into the model.

2. Model Selection & Benchmarking

For this project, we have shortlisted **ViViT** and **TimeSformer** as potential transformer models for video embeddings:

- **ViViT**: A pure transformer-based approach that models spatiotemporal relationships across frames, making it effective for video understanding tasks.

- **TimeSformer**: A self-attention-based model that efficiently captures temporal dependencies, providing high-quality video embeddings.

Unlike text and vision models, where benchmarking was essential, video transformers are **already well-researched and computationally expensive to benchmark extensively**. By selecting two widely used models, we can **prioritize integration and inference optimization for OpenVINO rather than running comparative benchmarks**.

2. Inference Execution with OpenVINO Runtime: Once the data is preprocessed, it will be passed to OpenVINO for optimized inference execution. This section covers model selection, optimized execution, and integration with OpenVINO Model Server.

1. **Model Selection**
    - We have shortlisted models for text, image, audio, and video embeddings based on benchmarking or prior research.
    - These pre-trained models will be **exported and optimized for OpenVINO** using `ov.convert_model()`,ensuring compatibility with OpenVINO Runtime.
    - The models will undergo validation to ensure they retain accuracy after conversion.

2. **Optimized execution with OpenVino**

    Inference execution is optimized using OpenVINO Runtime to ensure faster, memory-efficient processing across different hardware backends. The optimization techniques include:

    I. **Model Quantization:**
        - Models are converted to FP16 precision, reducing memory footprint while maintaining high accuracy.
        - **INT8 quantization** is applied selectively using OpenVINO's Post-training Optimization Toolkit (POT). This further accelerates inference by leveraging low-bit computations, particularly for deployment on resource-constrained devices such as VPUs.
        - A comparison of **FP16 vs. INT8** performance is conducted to determine the optimal trade-off between speed and accuracy.

    Ii. Adaptive Hardware Deployment:

        - OpenVINO's automatic device discovery selects the best hardware backend for inference execution.
        - The models are tested across:
            I. **CPU:** Optimized using OpenVINO's thread parallelism and vectorized execution.
            II. **GPU:** Leverages OpenVINO's heterogeneous execution, distributing workloads across CPU and GPU for **efficient processing.**

III. **VPU/NPU:** If applicable, low-power AI accelerators are tested to further optimize inference.

III. **MediaPipe Graph Integration: To modularize and scale embeddings generation:**

- **MediaPipe Graph Calculators are created for each data modality:**
  I. **Text:** Already implemented in **OpenVINO Demo Repo**
  II. **Image/Audio/Video:** Custom calculators will be developed by adapting the text embedding graph as a base.
- **Each graph:**
  I. Accepts preprocessed input.
  II. Calls the optimized OpenVINO model.
  III. Outputs the final embedding tensor
- **Benefits:**
  I. Modular design aligns with **OpenVINO Model Server** architecture.
  II. Makes embedding APIs consistent, **reusable**, and **scalable** for any modality.

3. Serving Embeddings via REST API: To provide seamless access to multimodal embeddings, we will create REST API endpoints that process diverse input data types and return embeddings in an efficient and scalable manner.

1. **API Functionality:**

- **Input Acceptance**: The API will accept text, image, audio, or video files as inputs, making it flexible across different data types.
- **Preprocessing & Inference**: Each input type will undergo preprocessing (e.g., text tokenization, image resizing, audio feature extraction, etc.) before being sent to the OpenVINO-powered model for inference.
- **Embedding Output**: After processing, the API will return the embeddings for each input, which can be further used for tasks like search, comparison, or similarity calculations.

2. **No Streaming for Embeddings:** Unlike some use cases that benefit from partial responses (streaming), calculating embeddings requires processing the entire input to generate a complete representation. OpenVINO Model Server supports streaming for text-based applications, but for this project, we will prioritize **efficient batch processing** and **response optimization** for better overall performance, especially with larger datasets.

3. **Cross-Modal Search Capabilities:**

- The API will also enable cross-modal search. This means that a user can query in one modality (e.g., text) and retrieve the most relevant results from another modality (e.g., image).
- **Example**: A user could submit a text query such as "a sunset over the ocean" and receive a list of images whose embeddings most closely match the query's text embedding. This enhances the API's functionality, making it a powerful tool for multimodal applications.
  By adopting this structure, the API will be **lightweight**, **scalable**, and **optimized for multimodal search applications**.

4. API Robustness:

- **High Volume Handling (Rate Limiting & Queuing)**: To handle a high volume of requests, we will implement rate limiting, ensuring that the system does not become overwhelmed by too many requests in a short period. Each user or API client will be limited to a set number of requests per minute, with possible exponential backoff in case of heavy load. Additionally, we will employ queuing mechanisms to ensure that requests are processed in an orderly manner, preventing system crashes or slowdowns.
    I. Example: **API Gateway** for rate limiting and queuing, such as AWS API Gateway or Nginx.
    II. Implementing **Backpressure** in queues to prevent system overload.

5. Scaling the API:

- Deployment on Cloud Services: To handle varying loads and ensure scalability, the API will be deployed on a cloud platform **(e.g., AWS, GCP, or Azure)**. We will use cloud-native features like Auto-Scaling to automatically increase or decrease the number of server instances based on traffic demand. This helps maintain consistent performance even during peak usage times.

4. Vector Databases Selection:

1. Client-Side Vector Database

In this architecture, the **client-side vector database** plays a crucial role in efficiently storing and retrieving embeddings generated by the OpenVINO Model Server. Embeddings are numerical representations of data (e.g., images, text, audio) that capture their semantic features, which are critical for similarity search. Here's a detailed look at how this works:

- **FAISS, ChromaDB, Weaviate, Pinecone**: These are popular vector databases used to store and perform similarity searches on embeddings. The choice of the database depends on the application's needs, such as scalability, speed, ease of use, and integration with other tools. Let's briefly discuss each:
    - **FAISS (Facebook AI Similarity Search)**: An open-source library for efficient similarity search and clustering of dense vectors. It's highly optimized for speed and is often used in applications requiring fast nearest neighbor searches across millions of vectors.
    - **ChromaDB**: A highly flexible vector database designed for machine learning applications. It allows you to store and manage embeddings and supports advanced querying features like filtering.
    - **Weaviate**: An open-source vector search engine designed to enable easy integration with machine learning models. Weaviate offers a rich set of features, including filtering, clustering, and real-time updates.
    - **Pinecone**: A fully managed vector database service that provides automatic scaling and optimized performance for similarity search applications. Pinecone is cloud-native and ideal for enterprise-level solutions.
- **Embeddings Storage**: After the OpenVINO Model Server generates embeddings (e.g., from an image, text, or audio), these embeddings are stored in the selected vector database on the client side. This approach has the advantage of reducing server load and allowing the client to efficiently manage storage and retrieval.

- **Client's Role**: The client manages both storage and retrieval, allowing for flexible query execution and optimized performance. Since the client handles the vector storage, there's less strain on the server, and users have more control over their data.

2. Server-Side Embedding Generation (OpenVINO Model Server)

The **server's responsibility** is solely focused on generating embeddings from raw data (e.g., text, images, audio). The OpenVINO Model Server uses pre-trained models or custom models to generate these embeddings. By offloading the embedding generation to the server, you ensure that the client can focus on retrieval and searching, making the system more efficient.

- **OpenVINO Model Server**: OpenVINO (Open Visual Inference and Neural Network Optimization) is a toolkit for high-performance deep learning inference. The OpenVINO Model Server provides an API to handle large-scale inference tasks. The server processes raw data, applies pre-trained models, and produces embeddings which are sent back to the client for storage in the vector database.

- **High-Performance Inference**: OpenVINO's strength lies in optimizing inference, making it suitable for applications requiring fast and accurate embedding generation. This allows for real-time image, video, or audio processing.

3. Similarity Search Implementation

The **vector database** on the client side supports efficient similarity search, which is a key component of the application. This allows users to perform cross-modal retrieval, meaning that the database can handle queries from one type of media (e.g., text) to retrieve another type (e.g., images, videos, audio).

- **Cross-modal Retrieval**: The idea behind cross-modal retrieval is that the embeddings generated by OpenVINO can be used to link different types of media. For example:

    - A user might input a **text description** of an image, and the system retrieves the most semantically similar images.

    - Alternatively, a user might input a **query image** to retrieve similar images or videos.

- The underlying vector space (created by embeddings) allows different types of data to be compared and matched based on their semantic meaning, rather than exact data matches. This enables applications like **content-based image search**, **video search**, or even **audio-based searches**.

- **Similarity Search Algorithms**: The vector database uses algorithms like **k-nearest neighbors (k-NN)**, **approximate nearest neighbor (ANN)**, or **hierarchical navigable small world graphs (HNSW)** to search for similar vectors. This allows for highly efficient querying over large datasets.

- **Example**: A user inputs the query “sunset over the beach” in natural language. The system would search for images or videos with similar semantic features (like the colors, objects, or contexts in the description). The database searches the embeddings for the closest matches and returns those results.

4. Design Benefits

- **Separation of Concerns**: This design separates the concerns of embedding generation (handled by the OpenVINO Model Server) and data management (handled by the client-side vector database). This allows each component to focus on its core functionality, improving the overall system’s efficiency and scalability.

- **Scalability**: By leveraging cloud-native databases like Pinecone or distributed vector databases like FAISS, the system can scale easily to handle millions of embeddings without sacrificing search speed or accuracy.

- **Flexibility**: The design allows flexibility in choosing a vector database based on the application's specific needs. For instance, a small-scale application might opt for FAISS, while a large-scale commercial application might lean towards Pinecone for managed services and

ease of scaling.

- **Efficient Retrieval**: Since embeddings are stored on the client-side database, the retrieval of relevant data is fast, and search queries are processed locally without waiting for the server. This reduces the server's load and ensures a smooth user experience with real-time retrieval.

5. OpenVINO Model Server Focus

The OpenVINO Model Server focuses solely on inference. Since embedding generation can be computationally intensive, offloading this task to a specialized server allows it to perform high-performance computations without being bogged down by storage and retrieval operations. By having this dedicated role, OpenVINO is optimized for speed and accuracy in generating embeddings, ensuring the system's overall performance.

5. End-to-End Integration and Demo Application

With all components implemented—data preprocessing, embedding generation, API exposure, and vector storage—the final step is to integrate the entire system into a seamless pipeline powering the core application: **image search based on a text prompt**.

Workflow Overview

1. **User Prompt Input (Text/Image/Audio/Video)**
   The user interacts with the frontend and submits a query. This query can be a natural language prompt (e.g., “A red sports car in the desert”) or an input file such as an image, video, or audio sample.

2. **Preprocessing Layer (Python/Mediapipe)**
   Based on the input type, the preprocessing graph prepares the data:

   - Text is tokenized using a transformers-based tokenizer.
   - Image, audio, and video inputs are preprocessed into model-acceptable formats using Python nodes inside a Mediapipe graph.

3. **Embedding Generation via OpenVINO Model Server (OVMS)**
   The preprocessed input is sent via REST API to the OpenVINO Model Server, which performs inference using an optimized model from the HuggingFace hub. The server returns a multimodal embedding vector representing the semantic content of the input.

4. **Client-Side Embedding Storage and Retrieval**

   - If the workflow is indexing a dataset, embeddings of all local images (or other media) are computed and stored in the selected vector database (e.g., FAISS, ChromaDB, Weaviate, or Pinecone).
   - If the workflow is querying, the client computes the embedding for the input query and searches the vector database for the closest matches based on vector similarity.

5. **Similarity Search Results and User Interface Display**
   The client retrieves the top-k most similar embeddings from the vector database. The corresponding results (e.g., images) are displayed to the

user in the frontend interface, forming the core of the image search application.

System Benefits

- **Modularity**: Each component (preprocessing, inference, and retrieval) is independently designed and scalable.
- **Scalability**: OpenVINO ensures low-latency inference, and vector databases can scale to millions of embeddings.
- **Flexibility**: The system supports a variety of input types including text, images, video, and audio.
- **Practical Application**: Enables advanced search engines, recommendation systems, content tagging, and classification.

Final Deliverables

- REST API endpoints in OpenVINO Model Server for embedding generation.
- Python-based preprocessing graphs for multimodal data.
- Client-side integration with a vector database for storing and retrieving embeddings.
- A working demo application showcasing image search based on a natural language prompt.
- Optional extensions: audio-to-image search, video-based retrieval, user interface improvements.

This project requires a structured approach with carefully planned phases. The timeline outlines key stages, objectives, activities, and deliverables. Given the complexity of multimodal embedding models and data integration (text, image, audio, video), sufficient time is allocated for research, development, testing, and potential delays.

Phase	Dates	Goals	Key Activities
Community Bonding	**May 8 – June 1**	- Finalize scope: Full support for **image**; prototype **audio/video**. - Choose sample app format: Gradio, CLI, or Jupyter Notebook. - Study OpenVINO Runtime + OVMS graph API. - Set up GitHub repo, dev environment (Docker, dependencies).	- Final model & app format decisions - Architecture draft - Clean project skeleton on GitHub
Coding Phase 1 (Initial Implementation)	**June 2 –June 22**	-Implement Python preprocessing pipeline: -Image: Resize, normalize, transform -Audio: Convert to spectrogram, resample -Video: Frame extraction, sampling strategy - Integrate HuggingFace Transformers preprocessing + inference with OpenVINO - Build OVMS-compatible pipeline for each - Set up REST API endpoints in OVMS	- Working multimodal pipeline with image/audio/video support - Sample JSON inputs and model outputs - API endpoint spec & test curl scripts

Coding Phase 2 (Testing + midterm prep)	**June 23-July 14**	- Validate all endpoints: -Evaluate runtime, response structure - Add logging, error handling, fallbacks - Create small vector DB (Database to be selected on benchmarking simultaneously) - Run retrieval tests: text -> image	- Fully functional REST API for image, audio, video embeddings - Evaluation notebook with test examples - Ready for midterm!
Midterm Evaluations	**July 14 – July 18**	Submit midterm evaluation (contributor & mentor)	Feedback from mentors
Coding Phase 3 (App dev + UI)	**June 23 – July 14**	Build sample app: -Choose: CLI / Gradio / Jupyter -Image search via text prompt - Populate image folder ->embeddings->vector db - Compare with query embeddings	- Working demo app: Search images via text using generated embeddings
Coding Phase 4 (Polish + additional features)	**August 5 – August 24**	- Refactor code into modules - Add README, usage examples, Dockerfile - Test video/audio embeddings work in pipeline (proof of concept) - Write integration tests -Add additional features like deployment on cloud services, rate limiting, to be added based on consultation with mentors.	- Finalized codebase - Documented APIs, app, and deployment

Final Submission	**August 25 – September 1**	- Submit final work product and contributor evaluation - Mentor review period begins	- Final GitHub repo - Demo video (optional) - Evaluation submitted

General Questions

I. **How do you know OpenVINO?**

I was introduced to OpenVINO while working at the Vector Institute on a project focused on deploying computer vision models for real-time diagnostics in medical imaging. My colleague and I were tasked with optimizing deep learning pipelines for edge devices, where we encountered performance bottlenecks with traditional inference frameworks. OpenVINO proved to be an ideal solution, and through this experience, I gained hands-on exposure to its Model Optimizer and Inference Engine, successfully optimizing models for Intel hardware to meet real-time performance constraints.

II. **What do you know about OpenVINO?**

OpenVINO (Open Visual Inference and Neural Network Optimization) is a comprehensive toolkit developed by Intel for optimizing and deploying deep learning models on Intel hardware platforms, including CPUs, GPUs, FPGAs, and VPUs. The toolkit includes several key components:

1. **Model Optimizer**: This tool converts pre-trained models from popular deep learning frameworks (such as TensorFlow, PyTorch, and Caffe) into an optimized intermediate representation (IR) format, which is designed to run efficiently on Intel hardware.

2. **Inference Engine**: This component facilitates the deployment of models on various Intel hardware platforms. It enables the execution of the optimized models in a highly efficient manner, providing APIs for easy integration with applications.

3. **Pre-trained Models**: OpenVINO provides a repository of pre-trained models that are optimized for Intel hardware, making it easy to quickly implement and test solutions.

4. **Development Tools**: The toolkit includes a variety of development tools and libraries to help with model optimization, performance tuning,

and inference deployment.

OpenVINO enables faster and more efficient inference for AI applications, particularly in edge computing and real-time scenarios, by leveraging Intel's hardware capabilities. Its flexibility in supporting a wide range of Intel devices and its ease of integration into existing pipelines make it a powerful solution for AI developers.

III. Have you already contributed to the OpenVINO project? (please include links)

Yes, I have contributed to the OpenVINO project by submitting multiple pull requests.The links:

- Support quantized::conv1d_relu
- Extend Python API with a new model reshape
- Support aten::_to_copy
- Keras PR #21050
- numpy.flip #keras

In addition to these pull requests I am also working on the following issues and would be creating a pull request for them soon

1. [OP CONFORMANCE][TEMPLATE] Interpolate test fails in op conformance#23553
2. If operation is incorrectly reporting profiling data in GPU
3. Align behavior of ONNX Frontend operator AveragePool-7, 10, 11, 19 with original framework#20553

IV. How could you apply it to your professional development?

OpenVINO will allow me to develop a deep understanding of optimizing deep learning models for production environments, particularly in edge computing scenarios where performance and efficiency are paramount. OpenVINO's capabilities in model quantization, pruning, and low-precision inference provide a unique opportunity to directly work with techniques that reduce computational load while maintaining or improving model accuracy. This is critical in applications where running large models on limited hardware resources—such as edge devices or IoT—is a common challenge.

Specifically, OpenVINO provides tools for accelerating inference through graph optimizations, operator fusion, and execution on Intel's specialized hardware (CPUs, GPUs, VPUs, etc.). Learning how OpenVINO achieves these optimizations and contributing to its ongoing development will allow me to gain a deeper understanding of how to implement these optimizations for other

hardware architectures. For instance, understanding the intricacies of CPU and VPU optimizations and how OpenVINO maps models to these devices will enhance my ability to design and deploy machine learning applications with hardware acceleration in mind. I can also apply these insights to other deployment environments and AI solutions outside of OpenVINO.

Additionally, OpenVINO's support for multiple frameworks (TensorFlow, PyTorch, ONNX, etc.) and deployment scenarios means I will gain experience in cross-platform integration and optimizations. For example, I could work on optimizing a model trained in TensorFlow and ensure it runs efficiently on a specific Intel device using OpenVINO's model optimizer. This kind of cross-framework integration requires a thorough understanding of model architecture and performance bottlenecks, as well as the nuances of converting models between different formats without losing critical performance metrics.

Moreover, OpenVINO allows fine-grained control over memory management, allowing users to profile model performance and adjust execution strategies for optimal throughput. This will provide me with hands-on experience in performance tuning and benchmarking, both of which are key in deploying AI systems at scale. Understanding how to optimize both computational performance (e.g., parallelism, batching) and memory access patterns (e.g., memory pools, device-specific optimizations) will be invaluable when developing AI systems for production environments.

The direct application of such optimizations will allow me to better design AI systems with an explicit focus on efficiency—especially in resource-constrained environments such as mobile devices, autonomous vehicles, or industrial IoT applications. This will align directly with my goal to specialize in the deployment of AI models on embedded and edge devices, where performance, energy efficiency, and real-time response are critical.

Finally, contributing to OpenVINO will provide me with experience working in an open-source, collaborative environment, where best practices for code reviews, version control, and continuous integration are essential. I expect that interacting with experienced engineers and experts in AI model optimization will enhance my ability to write clean, efficient code while adhering to rigorous standards. By reviewing pull requests, debugging issues, and testing optimizations, I will improve my skills in software engineering, collaborative workflows, and scalable software architecture—skills that are crucial for my growth as a machine learning engineer.

V. Describe any other career development plan you have for the summer in addition to GSoC.

In addition to my work on the GSoC project, I am also dedicating time to a startup I am co-developing with a group of friends under the University of Toronto's Entrepreneurship Hatchery. We are building a database-focused application that leverages cutting-edge technologies to address challenges in data management and optimization. The startup has recently secured funding, which provides us with the resources and support to accelerate our development.

This project offers me a unique opportunity to apply my technical expertise in cloud computing, data engineering, and AI, while also honing my skills in entrepreneurship, project management, and collaborative teamwork. The experience has been invaluable in learning how to navigate real-world technical challenges while also driving business innovation.

Parallel to this, my work on OpenVINO is equally important to my professional development. OpenVINO's focus on optimizing AI workloads for various hardware accelerators directly complements the work I am doing in my startup, where efficient processing and data handling are key. OpenVINO is empowering me to deepen my understanding of AI and machine learning at scale, and its application in both my GSoC project and startup is fostering a holistic approach to solving problems in high-performance computing environments.

The integration of OpenVINO with our database application in the startup allows us to incorporate AI optimizations that will significantly enhance performance, providing more efficient data processing and computation. Thus, both GSoC and my startup experience are helping me build expertise not only in AI but also in developing scalable, real-world applications.

Together, these endeavors are preparing me for a career at the intersection of advanced technologies and business innovation, equipping me with the technical and entrepreneurial skills necessary for tackling challenges in today's rapidly evolving tech landscape.

VI. Why should we pick you?

I believe I would be a valuable addition to the OpenVINO team due to my solid technical foundation, my previous experiences with AI optimization, and my sincere interest in advancing the OpenVINO framework. While I have contributed to OpenVINO through two pull requests, I view these as just the beginning of my journey with the project. They have provided me with an important understanding of its structure, but I am eager to deepen my technical expertise and work on more complex, impactful contributions.

My experience with OpenVINO has allowed me to explore the nuances of optimizing AI models for deployment, and I am fascinated by its potential to significantly improve the efficiency of AI applications across various industries. I am particularly excited about the opportunity to work on OpenVINO's integration with Intel hardware and contributing to its ongoing improvements in optimizing neural networks for better performance and scalability.

In addition to my technical skills, I bring a collaborative mindset and a genuine desire to learn from experienced mentors within the community. I understand that OpenVINO has a highly technical and specialized ecosystem, and I am committed to putting in the effort to contribute meaningfully while growing my own skill set. I am confident that my background in machine learning and AI, combined with my passion for open-source software and model optimization, will allow me to contribute effectively to OpenVINO's goals.

Ultimately, I am motivated by the opportunity to work on real-world AI challenges, learn from the best, and make a tangible impact on OpenVINO's development. I am ready to take on new challenges, improve my skills, and contribute to the success of the OpenVINO project.

Tasks

The PRs I have created till now:

- **Link to Pull Request**: **Support quantized::conv1d_relu**
- **Link to Pull Request**: **Support aten::_to_copy**
- **Link to Pull Request: Extend Python API with a new model reshape overload**
- **Link to Pull Request: numpy.flip #keras**
- **Link to Pull Request**: **Keras PR #21050**

References

1 Google, “GSoC logo horizontal,” *Wikimedia Commons*, [Online Image]. Available: https://commons.wikimedia.org/wiki/File:GSoC-logo-horizontal.svg. [Accessed: Mar. 19, 2025].

2 Intel Corporation, “OpenVINO Logo,” *Intel Official Website*, [Online Image]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/images/2022-07/openvino-logo.png. [Accessed: Mar. 19, 2025].