
Εθνικό Μετσόβιο Πολυτεχνείο Σχολή Ηλεκτρολόγων
Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας, Πληροφορικής και Υπολογιστών

# Απεικόνιση αλγορίθμων Βαθιάς Μάθησης σε πλατφόρμες υλικού Graphcore IPU και NVIDIA GPU

Απόστολος Γερακάρης
Επιβλέπων: Διονύσιος Πνευματικάτος

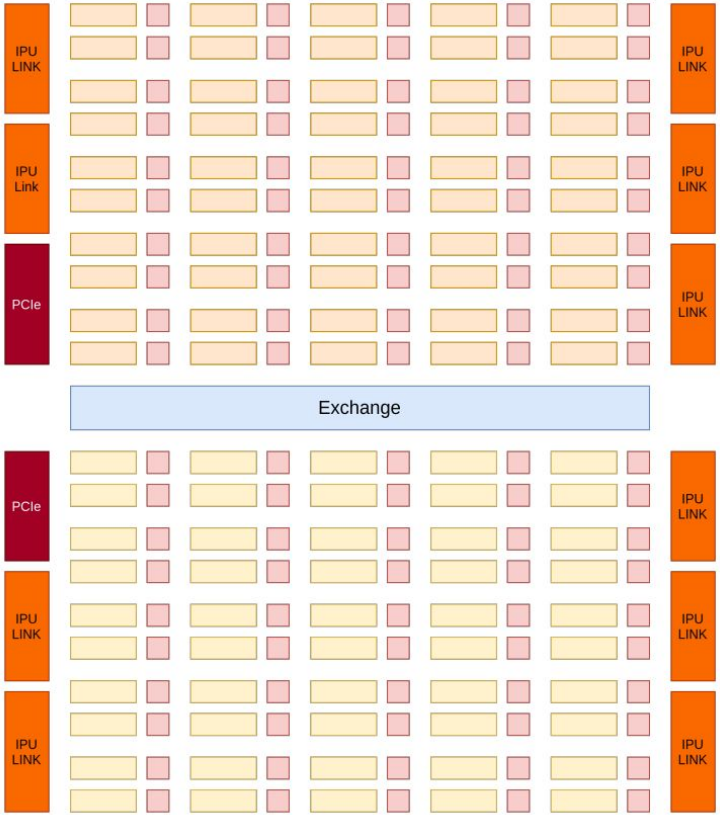
10 ΑΠΡΙΛΙΟΥ 2023

# Problem Definition

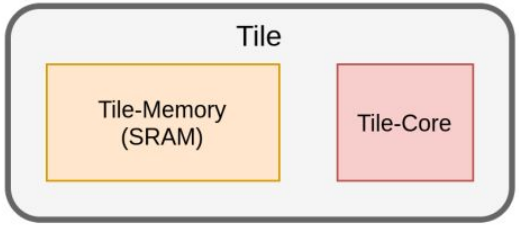
**Goal:** Evaluate the performance of Graphcore Intelligence Processing Units (IPUs) and Graphics Processing Units (GPUs) hardware platforms by deploying Machine Learning algorithms and Deep Learning models in both inference and training tasks:

- Inference Task - Eyeblink Conditioning
  - Implement a real-world scenario based on eyeblink conditioning, a widely studied model in cognitive neuroscience.
  - Explore solutions from the fields of ML and DL.
  - Evaluate the platforms' performance in terms of latency, throughput, and energy efficiency during the inference task.
- Training Task - CNN-based Model:
  - Train a Convolutional Neural Network (CNN) model on both hardware platforms.
  - Compare the platforms' performance in terms of training speed.

# Graphcore Intelligence Process Unit - Architecture

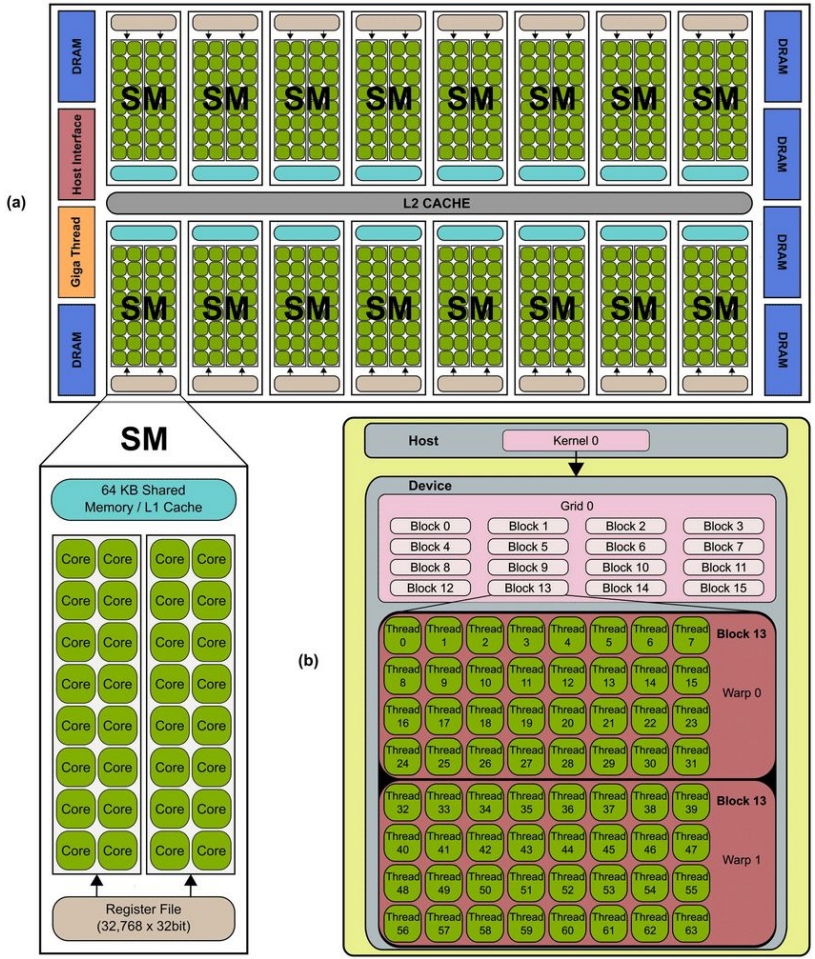



Simplified illustration of an IPU processor.




- Multiple Instruction Multiple Data (MIMD) architecture.
- Basic components: IPU-tile, IPU-exchange, IPU-links and PCIe interfaces.
- IPU follows the bulk-synchronous parallel (BSP) model of execution. The execution of a task is organized into step and each step is composed of 3 phases:
    - local computation phase
    - synchronisation phase
    - data-exchange phase
- Graphcore Poplar SDK (Software Development Kit), offers a comprehensive set of tools, libraries, and components designed to help developers program IPUs efficiently for AI and machine learning applications.
- Support for popular deep learning frameworks via the Graphcore SDK, such as Tensorflow, Keras, Pytorch

# NVIDIA GPU - Architecture




- Single Instruction, Multiple Data (SIMD) parallelism
- Consist of thousands of parallel processing cores, making them suitable for running computationally intensive tasks, such as deep learning workloads, with high performance.
- CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA.
- Execution flow:
  - host code execution,
  - kernel launch,
  - thread organization,
  - memory hierarchy,
  - synchronization
  - kernel completion.

# Hardware Specifications: Graphcore IPUs and Host CPUs

| Specifications | MK1 IPU | MK2 IPU |
|---|---|---|
| Architecture | MIMD | MIMD |
| IPU Cores | 1216 | 1472 |
| Memory (On-chip ) | 300 | 900 |
| Memory Bandwidth (TB/sec) | 45 | 47.5 |
| TeraFLOPS (FP16) | 125 | 250 |
| Max TDP (Watts) | 150 | 300 |

- MK2 offers higher processing power 1472 vs 1216 cores.
- The on-chip memory capacity in the MK2 IPU (900 MB) is triple that of the MK1 IPU (300 MB)

| Specification | MK1 Host | MK2 Host |
|---|---|---|
| Clock Speed | 2.70 GHz | 2.25 GHz |
| Number of cores | 24 (48 threads) | 64 (128 threads) |
| PCIe controller | PCIe 3.0 (48 lines) | PCIe 4.0 (128 lines) |
| L1 cache | 24 x 32 KB | 64 x 32 KB |
| L2 cache | 24 x 1024 KB | 64 x 512 KB |
| L3 cache | 33 MB | 256 MB |

- MK1 Host CPU: Intel Xeon Platinum 8168
- MK2 Host CPU: MD-EPYC 7742

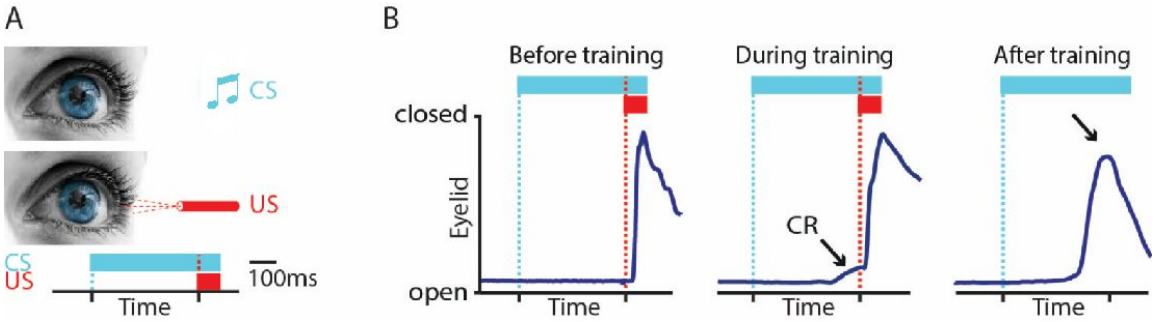
# Hardware Specifications: Tesla V100 GPU and Host CPU

| Specifications | Tesla V100 |
|---|---|
| Architecture | SIMD |
| Cores | 5,120 CUDA Cores and 640 Tensor Cores |
| Memory | 32GB HBM2 |
| Memory Bandwidth (GB/sec) | 900 |
| TeraFLOPS (FP16) | 125 |
| Max TDP (Watts) | 300 |

| Specification | V100 Host |
|---|---|
| Clock Speed | 2.0 GHz |
| Number of cores | 32 (64 threads) |
| PCIe controller | PCIe 3.0 (128 lines) |
| L1 cache | 32 x 64 KiB |
| L2 cache | 32 x 512 KiB |
| L3 cache | 64 MB |

V100 Host CPU: AMD EPYC 7551

# Inference Task: Eye-Blink Conditioning - Eyelid Closure Detection




Challenges:

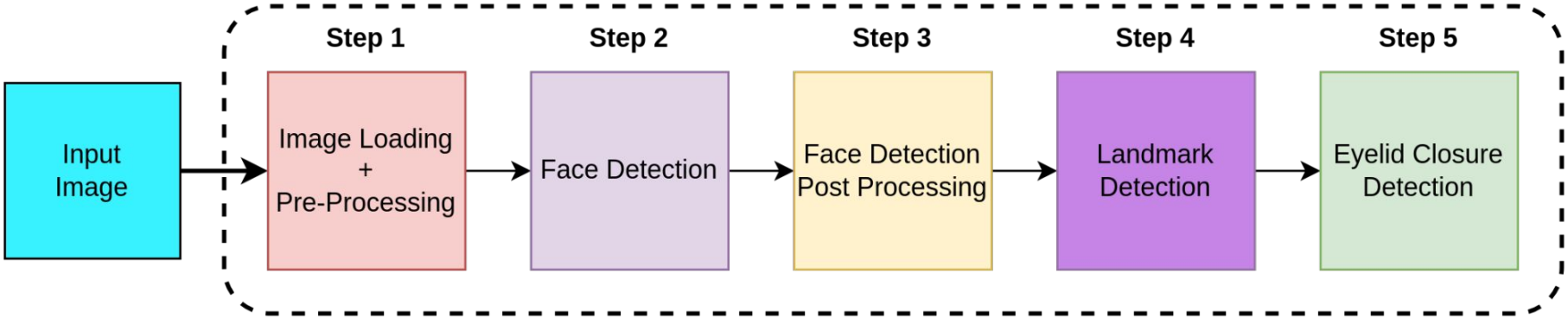
- Time-consuming manual data collection and analysis.
- Need for precise and accurate measurements.
- Off-line storage

Use cases:

- EBC has been used to study the neural processes involved in the acquisition of new motor skills in humans, providing insights into the changes that occur in the brain during motor learning.
- EBC has a wide range of potential applications in fields such as neuroscience, psychology, and clinical medicine, making it a valuable tool for understanding the brain and developing new therapies for neurological disorders.

# Inference Task: Eyeblink Conditioning - Eyelid Closure Detection

Goal: Automate and accelerate the process of eyeblink response detection from video in order to achieve real-time processing speed.



# Eyeblink Conditioning - Recording Settings of Eyeblink Test Set

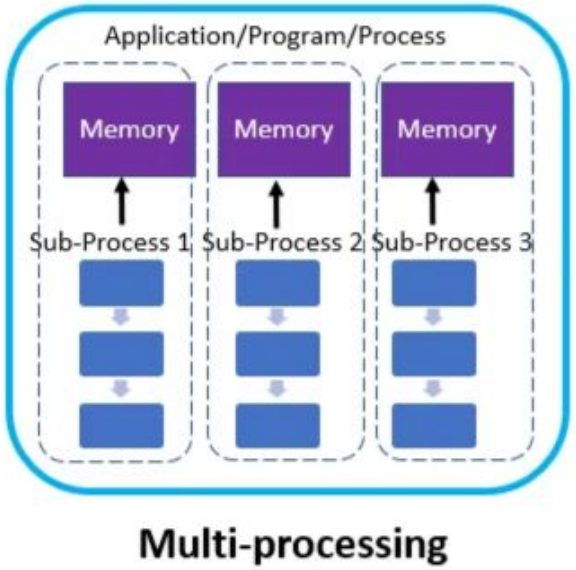
- Grayscale images of resolution 640 × 480 pixels.
- The subject is recorded with a high speed camera which is approximately one meter away.
- Every image depicts a single face.

| Yaw | Roll | Pitch | Occlusion | Lighting | Face size | Frame rate |
|---|---|---|---|---|---|---|
| ±20° | ±25° | ±40° | Glasses | Setting is well-lit, but videos suffer from light flicker | 20%+ of image | 500 FPS |

# Step 1: Image loading and Pre-Processing

- Python Imaging Library (PIL) is utilized for loading images and pre-processing tasks.
- PIL library is combined with Python Multiprocessing library to enforce a data-parallel solution.
  - Utilize the `multiprocessing.Pool` class to spawn a pool of workers (processes)
  - Split input data into smaller chunks
  - Process each chunk using a separate process





# Step 2: Face Detection - Algorithm Selection

The selection of the appropriate algorithms is of great importance!

Investigated Face Detectors:

- Histogram of Oriented Gradients (HOG)
  - Widely used (39313 citations) on the field of object and/or face detection.
  - A Pre-trained implementation is publicly available through the machine learning open-source Dlib library.
- Multi-task Cascaded Convolutional Networks (MTCNN)
  - A very popular and robust early CNN-based face detector.
  - A pre-trained implementation is publicly available through the Python Package Index (PyPI) repository
- BlazeFace
  - A lightweight and well-performing face detector developed by Google in the year 2020
  - A pre-trained implementation is available via the Mediapipe Library

# Step 2: Face Detection - Performance Evaluation of Face Detectors

Evaluation Datasets:

- Annotated Facial Landmarks in the Wild (AFLW)
  - Challenging dataset: images taken from real-life situations.
- BioID
  - Consists of 1521 grayscale images (384 × 288 pixel.
  - Strongly resembles our Eyeblink test-set.

Evaluation Metrics:

- Accuracy
- Execution Speed

$$Recall = \frac{TP}{TP + FN}$$

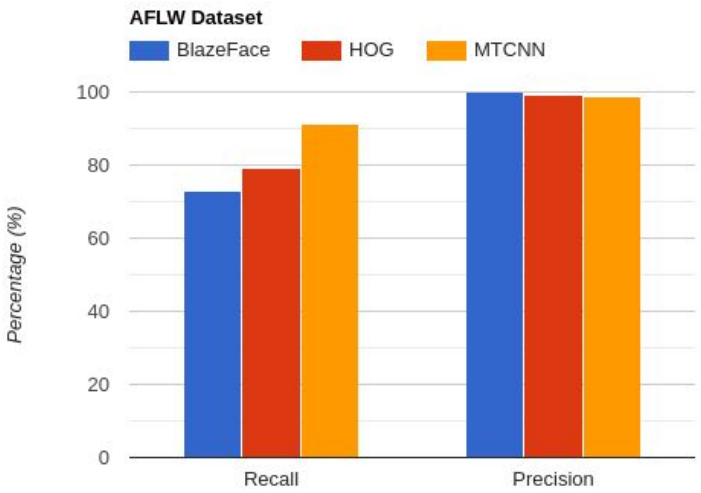
$$Precision = \frac{TP}{TP + FP}$$

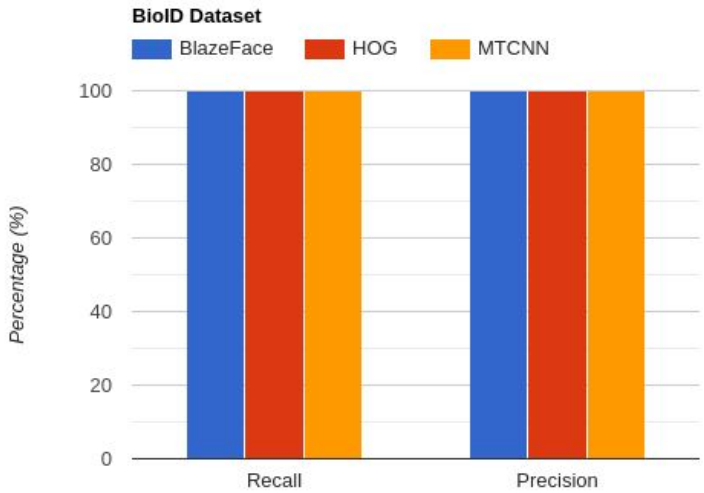
$TP = true\ positives,\ FN = false\ negatives,\ FP = false\ positives$

| Algorithm | Faces | FN | FP | TP | Time per Image (ms) |
|---|---|---|---|---|---|
| HOG | 3425 | 687 | 7 | 2738 | **29.23** |
| MTCNN | 3425 | **294** | 37 | **3131** | 347.4 |
| BlazeFace | 3425 | 940 | **1** | 2485 | 32.727 |

Detection results on AFLW dataset

# Step 2: Face Detection - Performance on AFLW and BioID Datasets

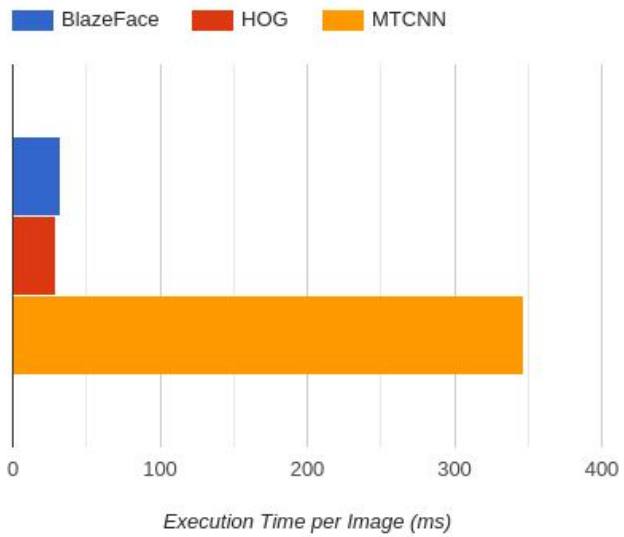



## Comments

- MTCNN is the most accurate face detector
- All three detectors produce 100% correct predictions on the BioID dataset. A strong indication that they will perform good in our test set.

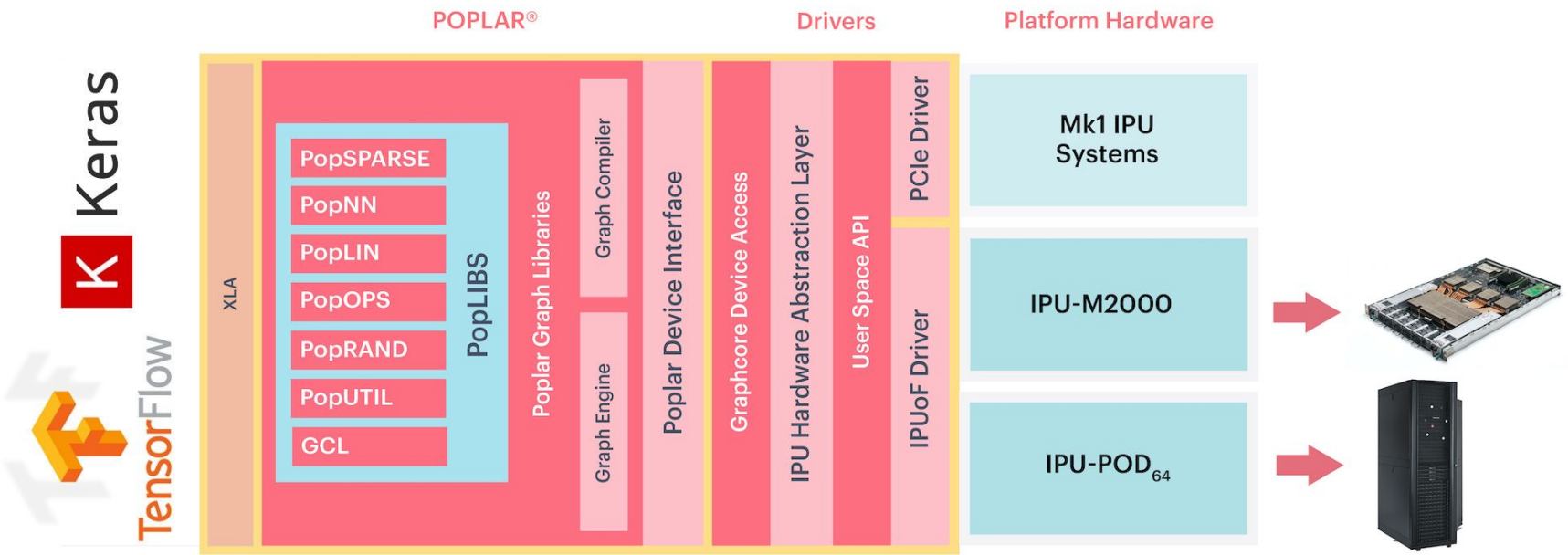
# Step 2: Face Detection - Performance Evaluation of Face Detectors on AFLW and BioID Datasets



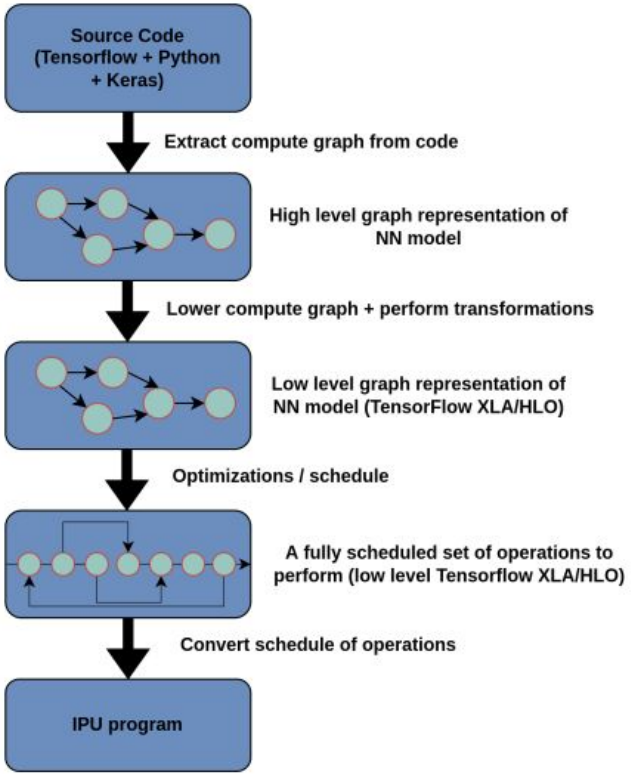

- MTCNN is significantly slower than BlazeFace and HOG detectors, primarily due to its more complex architecture, which uses multiple layers of convolutional neural networks to detect faces and facial landmarks, requiring more computational resources and longer processing times.
- In contrast, BlazeFace and HOG use simpler approaches for face detection, making them more suitable for real-time applications that require faster processing times.

Selected algorithm: **BlazeFace!**

# Step 2: Face detection - Deploy BlazeFace on MK1 and MK2 IPU with Tensorflow and Keras



# Step 2: Face detection - Deploy BlazeFace on MK1 and MK2 IPU with Tensorflow and Keras




- In order to run BlazeFace CNN-based face detector on an IPU-based system we need to construct an IPU-program.
- The IPU program is created using the Graphcore Poplar SDK and its associated libraries, tools, and compilers. It translates high-level model representations from popular machine learning frameworks like TensorFlow to an efficient low-level format that can be directly executed on the IPU.
- Multi-step process to optimize the performance of machine learning models on IPU chips.
- XLA (Accelerated Linear Algebra) compilation

# Step 2: Face detection - Optimize Inference of BlazeFace on MK1 and MK2 IPU

**Optimization: Maximize Compute Capabilities - Investigate Optimal Batch Size**

| Batch_Size | Run_time (s) | Throughput | Time/image (ms) | Compilation_time (s) |
|---|---|---|---|---|
| 1 | 160.483 | 398.796 | 2.508 | 26.8 |
| 2 | 84.241 | 759.725 | 1.316 | 28.3 |
| 4 | 45.19 | 1416.243 | 0.706 | 34.6 |
| 8 | 30.209 | 2118.574 | 0.472 | 38.9 |
| 16 | 21.904 | 2921.841 | 0.342 | 49.1 |
| **32** | **18.696** | **3423.192** | **0.292** | 64.8 |

Exploring optimal batch size for inference on MK1 IPU

- Optimal batch size for Mk1: 32

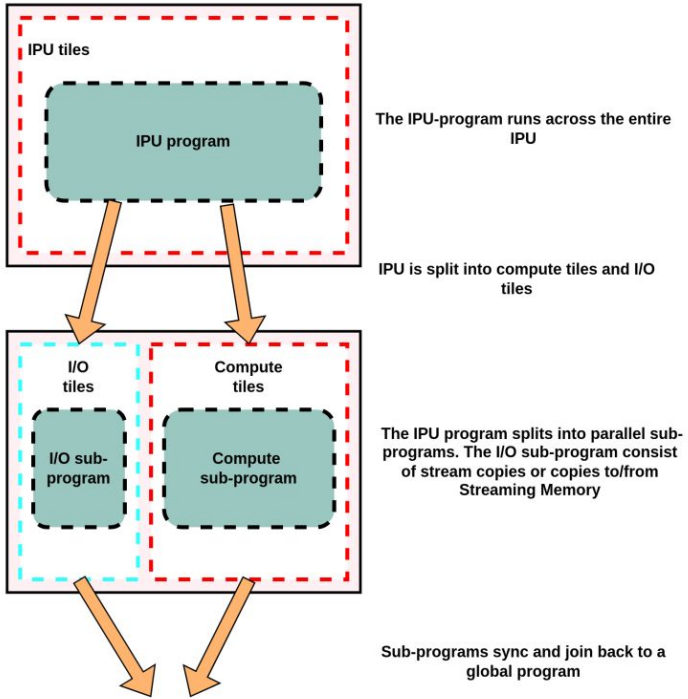
| Batch_Size | Run_time (s) | Throughput | Time/image (ms) | Compilation_time (s) |
|---|---|---|---|---|
| 1 | 134.029 | 477.509 | 2.094 | 126.7 |
| 2 | 70.832 | 903.546 | 1.107 | 131.8 |
| 4 | 42.739 | 1497.461 | 0.668 | 139,2 |
| 8 | 25.541 | 2505.775 | 0.399 | 144.7 |
| 16 | 21.291 | 3005.965 | 0.333 | 156,1 |
| 32 | 19.379 | 3302.544 | 0.303 | 236,4 |
| **64** | **14.1394** | **4526.370** | **0.221** | 240,2 |
| 128 | 15.3025 | 4182.311 | 0.2391 | 292.6 |

Exploring optimal batch size for inference on MK2 IPU

- Optimal batch size for MK2: 64

# Step 2: Face detection - Optimize Inference of BlazeFace on MK1 and MK2 IPU

**Optimization: Minimize Communication Overhead - I/O Tiles**




IPU tiles are split into two sets of tile groups:

1. **I/O Tiles**: perform only I/O operations to fetch and receive data from outside the chip.
2. **Compute Tiles**: perform only computations.

The IPU-program is also splitted into two sub-programs to run in parallel

The number of I/O tiles should be carefully tuned, since these tiles cannot participate in computations and using a large number can affect performance. It is considered more optimal to select a number of I/O tiles that is a power of two.

# Step 2: Face detection - Optimize Inference of BlazeFace on MK1 and MK2 IPU

**Optimization: Minimize Communication Overhead - Prefetch & Asynchronous Callback**

- The Prefetch option allows TensorFlow and Poplar to move input data logically closer to the IPU.
- The Asynchronous Callback, spawns an extra thread for dequeuing the processed data back to host as soon as they are ready.

A grid search experiment was conducted to explore the optimal values for both prefect depth and I/O tiles:
- MK1 IPU: 64 I/O tiles, 2 Prefetch Depth
- MK2 IPU: 64 I/O tiles, 2 Prefetch Depth

| Device | Batch_Size | Run_time (sec) | Throughput | Time/image (ms) |
|---|---|---|---|---|
| MK1 | 32 | 7,052 | 9074.475 | 0.107 |
| MK2 | 64 | 5,558 | 11514.051 | 0.079 |

Inference of BlazeFace on MK1 and MK2 IPUs with optimizations.

Performance improved by 36.5% for MK2 IPU and 28% for MK1 IPU

# Step 2: Face detection - Deploy BlazeFace on V100 Tesla GPU with Tensorflow and Keras

**Settings:**

- XLA compilation

| Batch_Size | Run_time (s) | Throughput | Time/image (ms) |
|---|---|---|---|
| 32 | 9.81 | 6521.81 | 0.1532 |
| 64 | 5.416 | 11816,838 | 0.0846 |
| 128 | 2.792 | 22968.123 | 0.0436 |
| 256 | 1.704 | 37705.264 | 0.0266 |
| 512 | 0.815 | 79123.324 | 0.0127 |

Inference on Tesla V100 GPU. Throughput (images/sec)

# Step 2: Accelerated Face Detection - Performance Evaluation of Hardware for Face Detection

| Implementaion | Time/Image (ms) | Speedup |
|---|---|---|
| BlazeFace serial version | 32.727 | - |
| MK1 BlazeFace implementation | 0.107 | 306 |
| MK2 BlazeFace implementation | 0.079 | 414 |
| Tesla V100 BlazeFace implementation | 0.0127 | 2576 |

The Tesla V100 GPU outperforms IPUs when performing BlazeFace inference, primarily due to its larger available memory. The increased memory capacity enables the GPU to run inference with larger batch sizes, thereby improving throughput and overall performance compared to IPUs.

# Step 3: Face Detection Post-Processing

- Blazeface model outputs 896 detections for every input image ⟶ Output Tensor: [b, 896, 17], where b = batch size
- Each output detection contains:
  - Four bounding box coordinates (x_center, y_center, width, height),
  - Twelve landmark coordinates (x1, y1, ..., x12, y12), and
  - a Confidence score
- All predicted coordinates correspond to the 128 x 128 pixel input image size ⟶ map them on the original image dimensions (640x480 pixels).

# Step 4: Landmark Detection

Selected Algorithm: Ensemble of Regression Trees (ERT)

- Reliable and fast landmark-detection model.
- A pre-trained implementation is available through the open-source Dlib library.

Two versions are used:

- 68-landmark detector.
- 12-landmark detector.

68-landmark output prediction

12-landmark output prediction

# Step 4: Landmark Detection - Work-sharing with multiple processes

- Python multiprocessing library is utilized to spawn multiple instances for parallel execution.
- Each spawned process will handle the landmark detection of N / P images, where N is defined as number_of_images divided by the number of processes P.

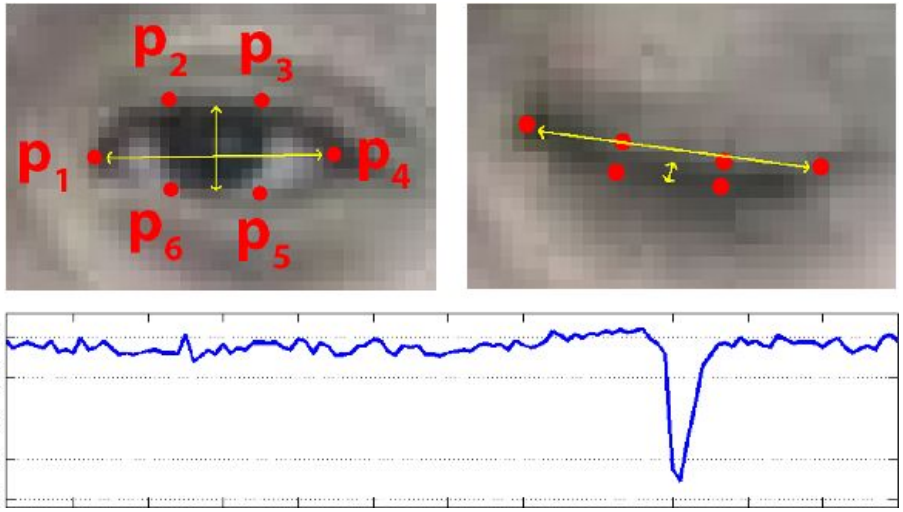
Settings:

- P = # of physical cores in each CPU.
- N = batch_size
  - 32 for MK1 Host
  - 64 for MK2 Host
  - 512 for GPU Host

**Acceleration Results**

| System | # Processes | Time/Image (ms) | Speedup |
|---|---|---|---|
| Original serial version | 1 | 1.06 | - |
| AMD EPYC 7551 (GPU) | 32 | 0.071 | 14,9 |
| Intel Xeon Platinum 8168 (MK1) | 24 | 0.078 | 13,6 |
| MD-EPYC 7742 (MK2) | 64 | 0.06 | 17,7 |

# Step 5: Eye Aspect Ratio (EAR)




- The EAR metric is used as an estimate of the eye opening state. Six points are needed: two in the corners, two on the upper eyelid and two on the lower eyelid. If these are accurate enough, they could be used directly for the calculation of eyelid closure.
- EAR is mostly constant when an eye is open and is getting close to zero while closing an eye.
- This stage is combined with the landmark detection step and is executed in parallel by multiple processes.

# Combined Implementation for Eyelid Closure Detection

| Hardware | Landmark model | Time per image (ms) |
|---|---|---|
| MK2 IPU | 68-point | 0.7116 |
| MK2 IPU | 12-point | 0.6938 |
| MK1 IPU | 68-point | 0.761 |
| MK1 IPU | 12-point | 0.731 |
| Tesla V100 | 68-point | 0.642 |
| Tesla V100 | 12-point | 0.603 |

The combined implementation consists of 4 steps:

1. Image loading, decoding and pre-processing (CPU)
2. Face Detection (IPU or GPU)
3. Face Detection Post-Processing (CPU)
4. Landmark Detection & EAR calculation (CPU)

IPU settings:

- XLA compilation
- 64 I/O tiles
- Asynchronous Callback
- Batch size:
  - 32 for MK1
  - 64 for MK2

GPU settings:

- XLA compilation
- Batch size: 512

CPU settings:

- # of Processes = # of physical cores in each CPU

# Inference Task: Hardware Evaluation - Low Latency Experiments

| Batch size | MK1 IPU | | MK2 IPU | | V100 GPU | |
|---|---|---|---|---|---|---|
| | Latency | Time/image (ms) | Latency | Time/image (ms) | Latency | Time/image (ms) |
| 1 | **1.97** | 1.97 | **1.701** | 1.701 | **3.724** | 3.724 |
| 2 | 2.108 | 1,054 | 1.841 | 0,921 | 3.789 | 1,895 |
| 4 | 2.337 | 0,585 | 2.077 | 0,519 | 3.794 | 0,949 |
| 6 | 2.894 | 0,482 | 2.293 | 0,382 | 3.873 | 0,645 |
| 8 | 3.058 | 0,382 | 2.831 | 0,353 | 3.83 | 0,478 |
| 10 | 3.526 | 0,352 | 3.112 | 0.311 | 3.829 | 0.382 |
| 12 | 4.283 | 0,356 | 3.545 | 0,295 | 3.866 | 0,322 |
| 14 | 4.478 | 0,319 | 3.734 | 0,266 | 3.972 | 0,283 |
| 16 | 5.042 | 0,315 | 4.148 | 0,259 | 4.189 | 0,261 |

| Batch size | Energy Efficiency | | |
|---|---|---|---|
| | MK1 | MK2 | V100 |
| 1 | 4,74 | 8,02 | 4,62 |
| 2 | 8,45 | 14,86 | 13,2 |
| 4 | 15.61 | 24,97 | 21,97 |
| 6 | 18,85 | 29,48 | 28,16 |
| 8 | 20,96 | 37,07 | 36,68 |
| 10 | 22,06 | 42,51 | 45,18 |
| 12 | 25,01 | 44,66 | 45,80 |
| 14 | 27,27 | 46,53 | 48,94 |
| 16 | **27,58** | **51,05** | **48,96** |

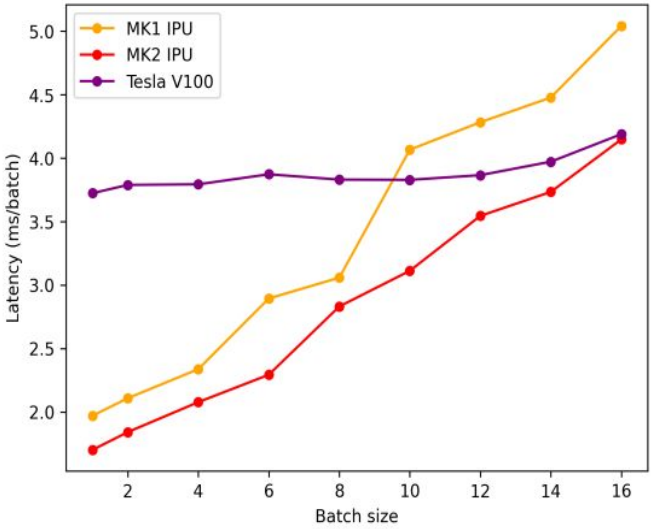
Settings:

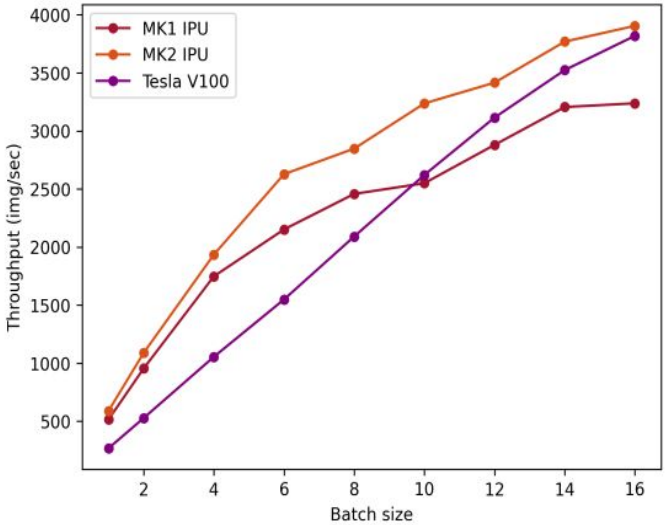
- I/O tiles: 32

Evaluation Metrics:

- Latency (ms per batch)
- Throughput (images per second)
- Power Consumption
  - IPU: gc-monitor tool
  - GPU: nvidia-smi tool
- Energy Efficiency (images / sec / Watt)

# Inference Task: Hardware Evaluation - Qualitative evaluation of Low Latency Experiments




Both MK1 and MK2 IPU chips perform better in terms of latency and throughput for smaller batch sizes (1 to 8) compared to the Tesla V100 GPU where there is no significant increase in latency as the batch size increases.

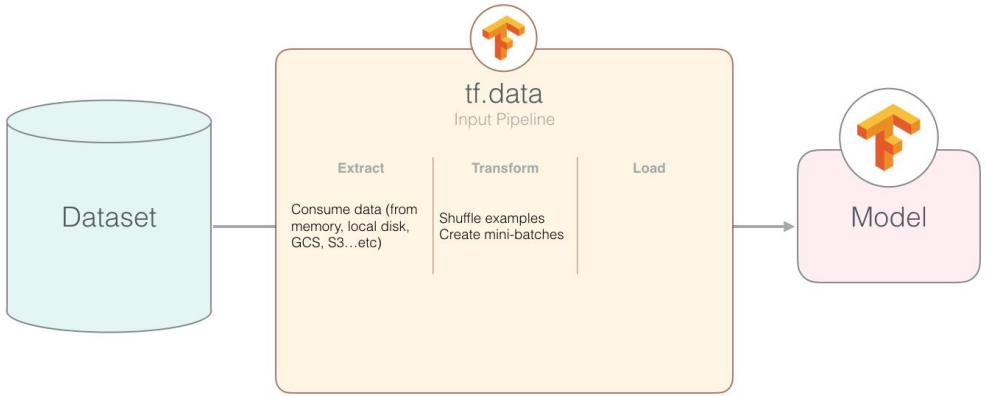
# Training BlazeFace CNN-Based Model on IPU-based and GPU-based systems

**Training Datasets**:

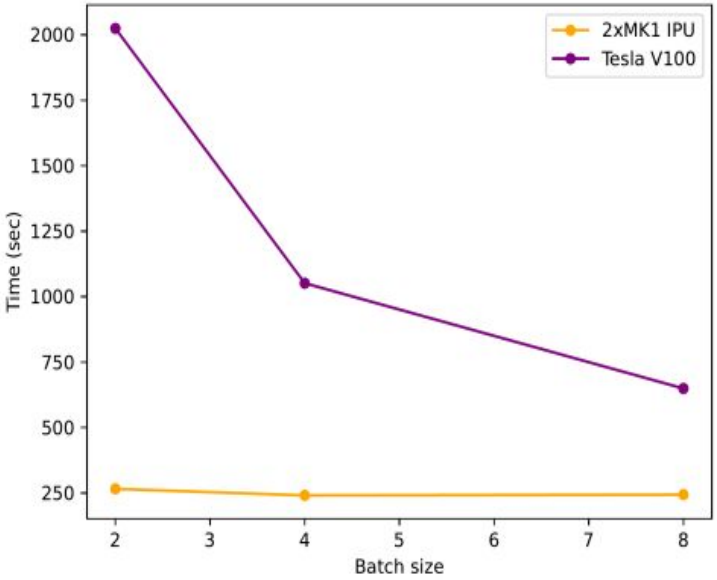
- FDDB: Consists of 2845 images with 5171 annotated faces collected from journalistic articles
- 300W-LP: Consists of 61,225 images with a single annotated faces

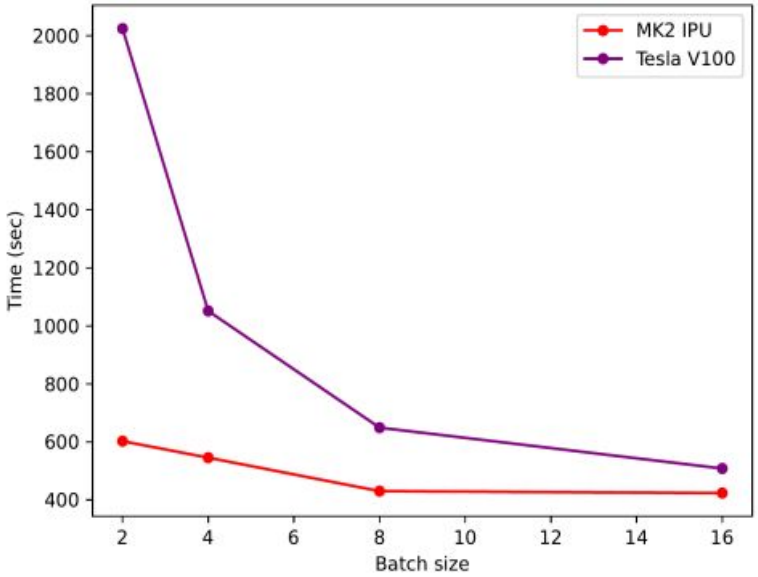
**Libraries & tools**





# Training results on FDDB Dataset



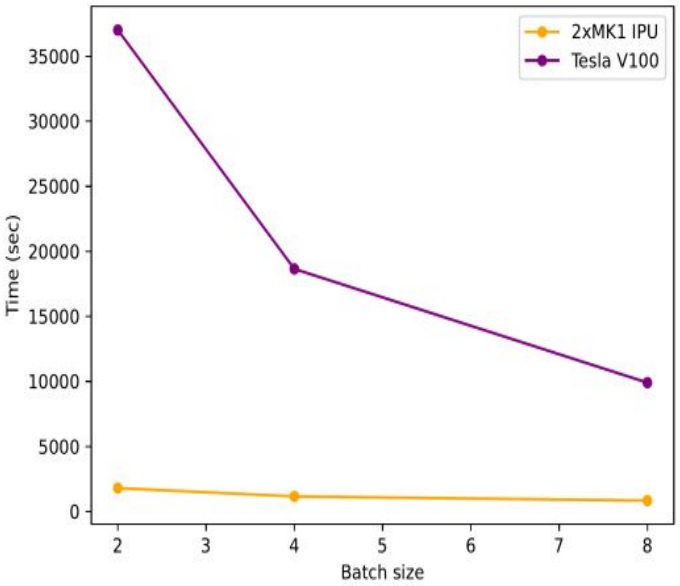



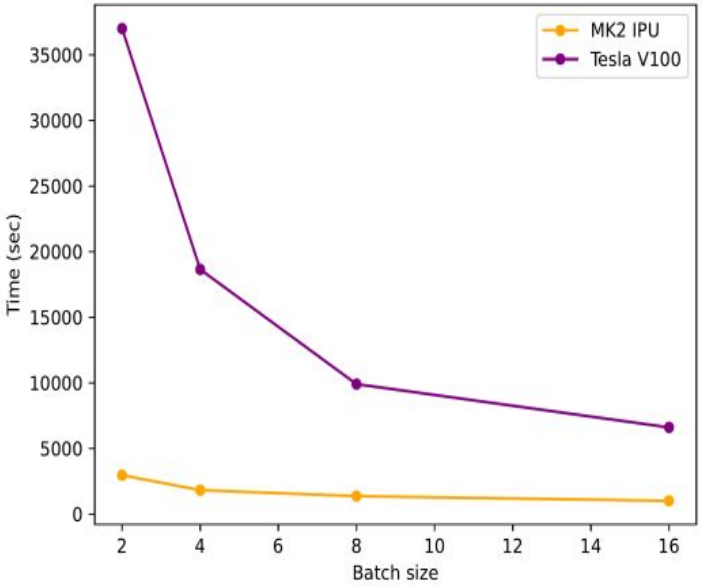

Settings:

- 150 epochs
- XLA compilation

Both the MK1 and MK2 IPUs outperform the Tesla V100 GPU in terms of training speed, especially for small batch sizes.

# Training Results on 300W-LP Dataset




Settings:

- 100 epochs
- XLA compilation

Again, both the MK1 and MK2 IPUs outperform the Tesla V100 GPU in terms of training speed, especially for small batch sizes.

# General comments on IPU and GPU systems for image-based workloads

- In our use-case (Eyelid Closure Detection) the Tesla V100 GPU outperforms IPUs when performing BlazeFace inference, primarily due to its larger available memory. The increased memory capacity enables the GPU to run inference with larger batch sizes, thereby improving throughput and overall performance compared to IPUs.

- IPUs, on the other hand, perform better at small batch sizes. We argue that this is due to their architecture, which is optimized for matrix operations and high-speed communication between the processing elements.

- The limited on-chip memory of the IPUs is an important factor that must considered carefully carefully when deploying training or inference experiments, especially for large models.

- The choice of hardware platform should be based on careful consideration of factors such as model size, memory requirements, latency requirements and available expertise in parallel programming and distributed systems.

- The GPU ecosystem is mature and well-established, offering an abundance of tools, libraries, and community support. Developers can access a vast array of resources, facilitating the discovery of solutions and optimization of ML models for GPU execution.

- Since IPUs have a unique architecture and programming model, there might be a steeper learning curve when adapting existing code or developing new models for the IPU platform, especially for developers who are already familiar with GPU programming.

# Synopsis

- We designed and accelerated a Machine Learning application to detect the amount of human eyelid closure in video data in a real-time processing speed of 500 frames per second.

- We explored solutions from the fields of Machine Learning and Deep Learning for the Face Detection and Landmark Detection stages of our application.
  - Selected Face Detector: BlazeFace
  - Selected Landmark Detector: Ensemble of Regression Trees (ERT)

- We conducted Inference experiments on IPU-based and GPU-based hardware systems.
  - GPU outperformed both MK1 and MK2 IPUs.
  - The limited memory of the IPUs is an important factor that must be considered carefully.

- We constructed an experimental end-to-end training pipeline for BlazeFace CNN-based model.

- We conducted Training experiments on IPU-based and GPU-based hardware systems.
  - Both the MK1 and MK2 IPUs outperformed the Tesla V100 GPU in terms of training speed, especially for small batch sizes.

# Future Work

- Future research can focus on improving the accuracy and accessibility of eye blink detection and exploring the potential of intermediate stages of face and landmark detection.

- Additionally, research can be conducted to evaluate the performance of IPU-based and GPU-based hardware platforms in various real-world scenarios to gain more insights into their relative strengths, weaknesses and suitability. This knowledge can then be used to develop more efficient and effective software and hardware solutions, ultimately advancing the field of machine learning and improving its practical applications.

- The current setup requires specialized hardware, such as cameras, to be effective. Future work can focus on developing eye blink detection solutions that can be implemented on portable devices, such as smartphones or smart glasses. This would make eye blink detection more accessible and usable in everyday life and provide additional data for research on eye blink patterns and their relation to attention, fatigue, and stress.

Thank You!

Questions?