High Performance Machine Learning Inference for Telco Clouds using GPU Clusters

Andrew Baldwin & Tero Rissa

0v4
5G Future X
Unleashing the potential of 5G

- x10k traffic
- >10Gbps peak data
- 10 years on battery
- 1M devices/km²

Optimization focus
Bit/s/Hz/m²/joule/$

Architecture design
- Cloud native
- Scalable
- Automated
- AI driven
- Open

"Unlimited experience"
5G
"Instant action"
"For everything"

Extreme Mobile Broadband
Massive machine communication
Critical machine communication
<1ms latency
Ultra reliability
Flexible, innovative and open massive scale access

### Classical BTS
- Stand alone solution for small scale 5G

### Cloud BTS vRAN1.0
- Radios connected via AirScale to radio cloud

### Cloud Optimized BTS
- Radios connect directly to radio cloud – RT function embedded in the AAS

### Full Cloud BTS vRAN2.0
- Radios connected directly to radio cloud

#### Open Interfaces allowing multi-vendor capabilities
- Zero touch and full automation through open API into analytics, AI and xRAN controller

#### Open API
- Cloud optimized 5G RF + antenna (w. L1, L2 RT)
- Airscale System module w. real-time baseband
- Airframe with 5G VNF (non-realtime baseband)
- Airscale System module w. real-time and non-real-time baseband
- Edge Cloud – RT enabled

#### AI & Analytics Management & Orchestration
- Core Cloud - Data Center
- xRAN Controller
- Data Center or Edge Cloud

#### Collaboration, Co-create, Innovate, Customize
Machine Learning in Telco Clouds

Modern Telcos are primarily networks of different sized data centers
- Core Datacenters for analytics and overall network management
- Edge Datacenters for aggregating and processing data from radio sites
- Dedicated Radio Hardware in towers and small cell sites, connected to Edge DCs

Machine Learning techniques, such as Deep Neural Networks, can be applied throughout the network to provide new kinds of data analysis and transformation using pre-trained models instead of hand-developed algorithms

Latency ranges:
- Sub-millisecond: Close to radio - Call state data, Scheduling algorithms
- Minutes: Core network - Network state analytics
Mobile Networks AI Vision

Covers all network technology generations - 2G to 5G and beyond

Individual network element optimized to its specific operating environment

Fully Automated

Open APIs

AI algorithms

5G Future X

Time to react

ms

ns

us
Challenges of running Machine Learning Models in Telco clouds

Two types of model training in focus:

- Supervised Learning using training data recorded and annotated, or generated by simulators with labels
- Self-supervised Learning using recorded data automatically labelled, e.g. timestamps – good for predictive time series

NN Model architectures are typically combinations of MLPs, RNNs (e.g. LSTM) and CNNs

Model complexity can range from K-ops to G-ops per inference

- Small models for microsec latency real-time usage, e.g. run with DSP
- Large models without latency or frequency requirements can use CPU
- Large models with higher frequency and lower latency requirements need accelerators
- Multiple models running inferences simultaneously

BUT Telco cloud today is all CPU-based

DC networks are fast (25-100Gb) & model network bandwidth low compared to operation count

Solution is to add shared local clusters of machine learning acceleration using GPUs
Our Machine Learning Inference Service

A network-API-based service for accelerated ML Inference

Service instances can be deployed to any cluster or size of Datacentre

Can run any kind of NN ML model, different frameworks

Allows efficient sharing of ML HW:
- Multiple simultaneous models per HW
- Split same work type across many HWs
Inference Service API Concepts & Principles

APIs defined with Websockets + Protobufts
- Efficient, secure & widely compatible
Client libraries for e.g. Python3, JS

Support most-common model frameworks
Requests should never fail (only slow down)
Be efficient, e.g. minimise power use

Model Management API
User creates serialised model files in local training environment
Upload model + verification data to service
Service tests & profiles model
Assign models to “model family ID”
Manages permissions for using models

Inference API
Request an inference session with a model family ID
Service allocates resources by loading models to HW accelerators (e.g. GPUs)
Pipelined stream of input data for inference requests
Service returns inference results as soon as possible
Model Lifecycle Management

Each model defines an interface by the input and output data structures. There can be many different compatible models with the same interface:

• Model managers focus on the model implementation
• Model users focus on the interface

Concept of “Model Family” defines an interface which can be resolved at inference time to many different but compatible models.

Example:

• A model family for accurate geolocation at a cell site
• Each cell site has a model trained on historical data from that call
• Client code requires only the model family Id for the geolocation model interface
• It will resolve at each site to a different compatible model specific to that cell
• All models can be updated with newer versions based on later data or even different model architecture with improved accuracy. Clients code is not changed
Model Definition

Service can support multiple model environments
  Example: Python 3.6 + tensorflow-gpu 1.10 + keras 2.2
Model definition consists of code + data, packaged in a serialised binary object

Code:
  • Model initialisation from data
  • CPU-thread-pool batch pre/post-processing functions
  • GPU-based Inference of a single batch

Data: Network weights

Model is loaded within service in a sandboxed environment (using “bubblewrap”)

Verification data also required for model registration - Set of sample input/output pairs generated during model training
Model Verification & Profiling

Verification data is used during model registration data to load and test model.

Is model functional?
- Return errors & exception data to user

Are model results as expected?
- Return any mismatches between expected and observed results

How does model perform?

Profile model:
- Can be heterogenous HW environment, e.g. M40s + P4s + V100s. Need to run on each different type.
- Measure GPU memory consumption when loaded and in use
- Measure time taken to load model
- Time per inference with different batch sizes to find latency/throughput tradeoff
- Runs model in controlled environment and use NVML library to gather statistics
Service Implementation Architecture

**Typical deployment**
Containerised with docker & nvidia-docker
Can be on one or multiple physical machines

- **Model Management API**
  - docker
    - Python 3.6 Control Server
      - 1 per service instance
  - nvidia-docker
    - Python 3.6 Worker Manager
      - 1 per GPU

- **Inference API**
  - docker
    - Python 3.6 Data Distributor
      - 1 per multiple clients
  - bubblewrap
    - Python 3.6 Model Executor
      - tensorflow-gpu
      - 1 per model per GPU

**Shared Object Store**
Model Definitions
Verification Data
1 per service instance
Optimising processing of inference batches

API can accept client inference requests in any batch size $\geq 1$
Replaceable plugin policy allows different performance optimisations to be implemented

- Break batches into optimal size for hardware
  - Throughput and memory
  - Discovered during the verification and profiling
- Split batches and run in parallel across multiple HW & instances of same model to reduce overall batch latency
- Combine batches of single or multiple clients for identical model to achieve optimal batch size and improve overall throughput & HW utilisation
  - Optimize utilization vs low latency
Pipelining requests

Service is fully pipelined to maximise possible throughput
Each client decides how many inference requests to queue concurrently on the channel (pipeline length)
- If results are needed strictly in sequence before next request, pipeline will be 1.
- If results are independent of future requests, pipeline can be >1.
  - Typically results in better throughput with same latency as pipeline 1
Websocket requests & replies have IDs allowing re-ordering
If requests go to different HW types, can complete at different rate.
Inference pre & post processing handled by thread pool – can cause reordering of batches.
Inference Latency

Between datacentres, geography & speed-of-light-in-fibre determines network latency
• Best case each 200km adds 1ms

Within a datacentre, OS architecture determine network latencies
• From client user-space process to server user-space process are on average few **100s us**
• Default deployment allowing batch optimisation requires 2 requests & replies \(\rightarrow\) up to **1ms** overhead
• Model execution overhead for python->tensorflow-gpu 1.10 *currently* \(~500us\)

For a simple model, total client-observed request-response latency currently about **1.5ms**
(Pipelining and batching makes this value independent of inference throughput)

On top of this is batch pre/post processing and model execution time
• These can be optimised by service client by model changes
• Typically total inference latency of **5-10ms** is observed

Two directions for latency improvements:
• Alternative and optimised model execution environments (c++, TensorRT)
• Direct connection to model for real-time clients preferring low latency over batch optimisation
Measuring and Optimising TensorFlow GPU memory usage

By default, first TensorFlow process will pre-allocated all GPU memory in all visible GPUs
• After that it is not possible to run other TensorFlow process on same GPU set

Alternatively, allocated memory on a GPU can be limited using config option:
  config.gpu_options.per_process_gpu_memory_fraction = [0.0-1.0]
• We override of the TensorFlow default config for models to control this value.

During profiling, it determines the minimal memory required by a model while maintaining correct functionality and performance, allowing better runtime GPU selection.

TensorFlow 1.10 also now includes experimental support for Nvidia unified memory:
  config.gpu_options.experimental.use_unified_memory = True
This should allow oversubscription on a single GPU so that more models can be loaded than physical GPU RAM would allow, with unused models being paged to host memory as needed.
• Keeping models in memory removes the model loading time
• Benefit when clients requesting more models than there is physical RAM in the GPU
Future Directions

Increase supported model environments as needed

Optimise architecture for real-time & lower latency use cases

Explore automated model transformation, e.g. for quantizing to float16
Mobile Networks Long Term ML R&D

Algorithms for predictive time series analysis

Alternatives for LSTMs

Inference Hardware

Custom HW can provide 100x performance, cost & power benefits

Overall 1000x cost, performance & power improvement possible opening ML to new set of use cases

Training Frameworks

Training also involves hyperparameter search

Quantized & optimized model

Performance aware GD

Performance driven algorithms

Topology options

Memory architecture

Hardware limitations
Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use of Nokia’s customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback").

Such Feedback may be used in Nokia products and related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.