Transforming Intelligence for the Edge:
Challenges and Opportunities in Modeling, Optimization, and Deployment

Professor Brett H. Meyer
Electrical and Computer Engineering
McGill University

September 20, 2022

Language Models at the Edge

- Advances in NLP → proliferation of language models
  - Simple commands, *a la* Alexa and Siri
  - Speech recognition, transcription, and translation
  - Question answering, etc
- It can be useful to perform these tasks at the edge when
  - Low-latency is a requirement
  - Privacy is important
  - Internet access is unreliable

Set a coffee timer.

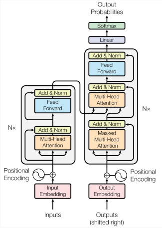
Is coffee poisonous to cats?

[Source: Bodum]

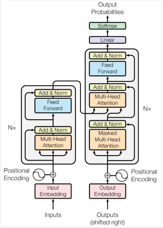
Are Transformers All We Need?

[Source: Lucas Beyer]

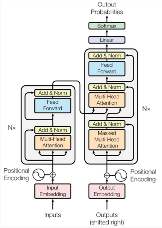
Computer Vision



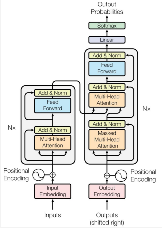
Natural Lang. Proc.



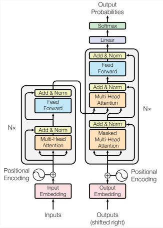
Reinf. Learning



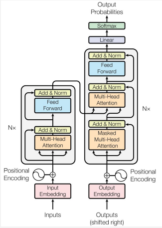
Speech



Translation



Graphs/Science



Transformer image source: "Attention Is All You Need" paper

“If there ever was a candidate for [the future of modeling], transformers certainly would be one.” –Delip Rao, *AI Research and Strategy*, September 14, 2022

Scarcity at the Edge

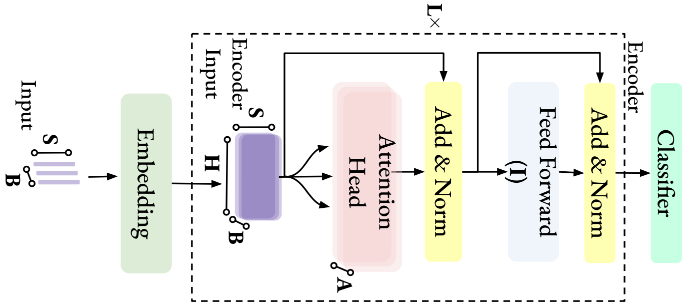
- Edge devices are *resource-constrained*
  - Less compute, less memory
  - Mobile: limited energy
- Edge devices are *heterogeneous* systems
  - Multiprocessor CPUs, mobile GPU, sometimes NPU
- Lots of ways to *optimize* models
  - Pipelining, partitioning, quantization, NAS, …
  - Very many options to consider!
- **Spoiler alert**: uneven library support makes things *interesting*

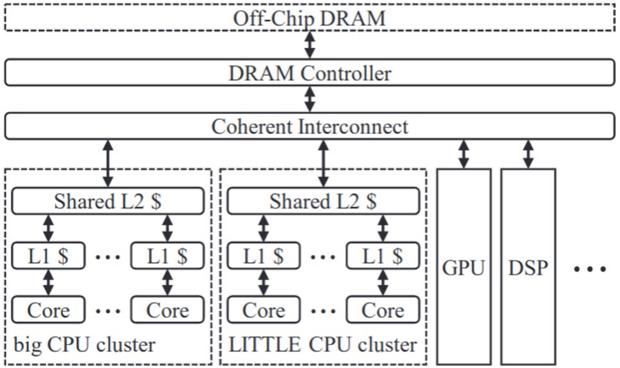
Anatomy of BERT Optimization and Deployment

Goal: performance and power optimization for the edge

Challenge: design space is large; model evaluation is expensive

1. Identify target hardware
2. Select optimization approach and search space
3. Take measurements to support metric estimation
4. Go! *Search*! **Optimize**!



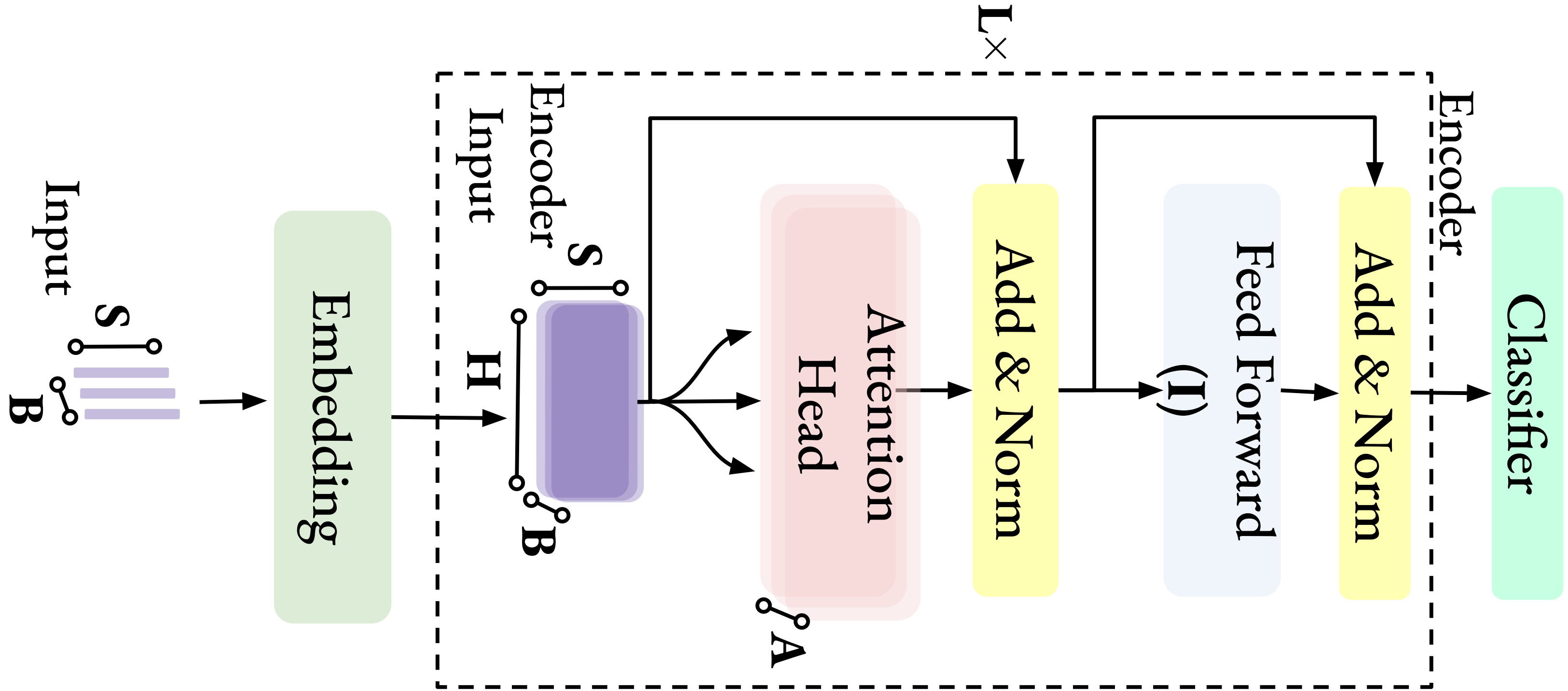


Kirin 970 SoC

This Talk: BERT Modeling and Optimization

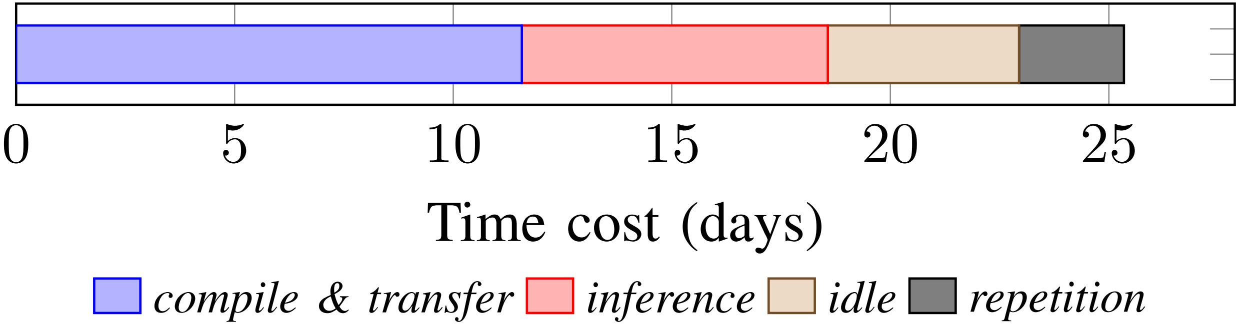
- Modeling performance on CPUs
- Pipelining for parallelism on heterogeneous CPU systems
- Partitioning for parallelism on CPU-GPU systems
- Cross-cutting challenges and opportunities

Bidirectional **E**ncoder **R**epresentations from **T**ransformers



So you want to find the optimal

- Why not just profile every model that is considered?
- Typical NAS experiments consider *many*, **many** possibilities
  - E.g., 100M! (J. Xu *et al.*, NAS-BERT, KDD 2021)
- Evaluating even 1% such a design space would take **forever**



Inference latency estimation is essential for model optimization

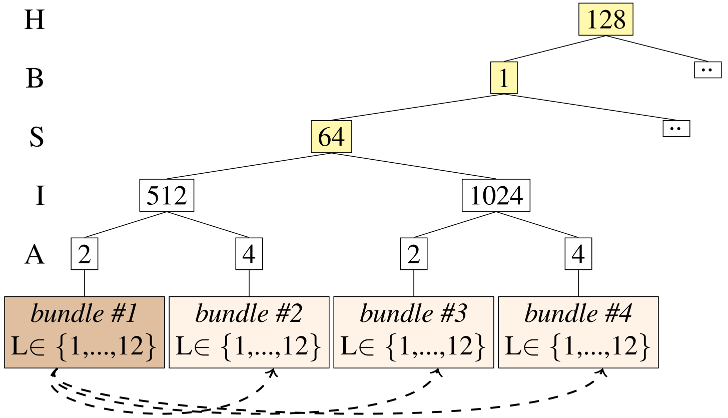
Wait, why are we measuring?

- FLOPS, parameters, etc, are poor proxies for latency
  - The same model executes in different time on different systems
- Past work has proposed evaluating and counting operations
  - E.g., NAS-BERT and others.
- This results in high error! It can't capture:
  - Caching
  - Parallelism
  - Intermediate tensor allocation

And all of the above vary from architecture to architecture

BERTPerf: Latency Modeling for ARM big.LITTLE

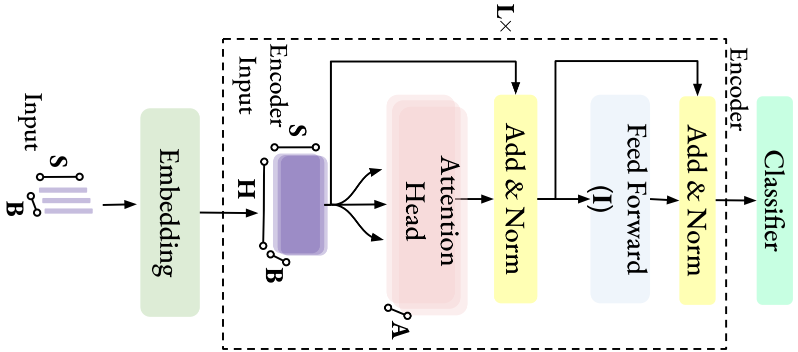
- Mobile GPUs are not ubiquitous; *CPU performance still matters!*
- BERTPerf is a BERT latency model for CPUs to support NAS
  - Predict model inference latency given BERT hyperparameters
  - *Goal*: minimize the number of measurements necessary
- Models are grouped into *bundles* with variable depth *L*
  - *A* and *I* have the least impact on latency
  - Systematically sample *L* in bundle #1 and measure
  - Bundles #2-4 behave similarly



[M. Abdelgawad *et al.*, SIPS'22]

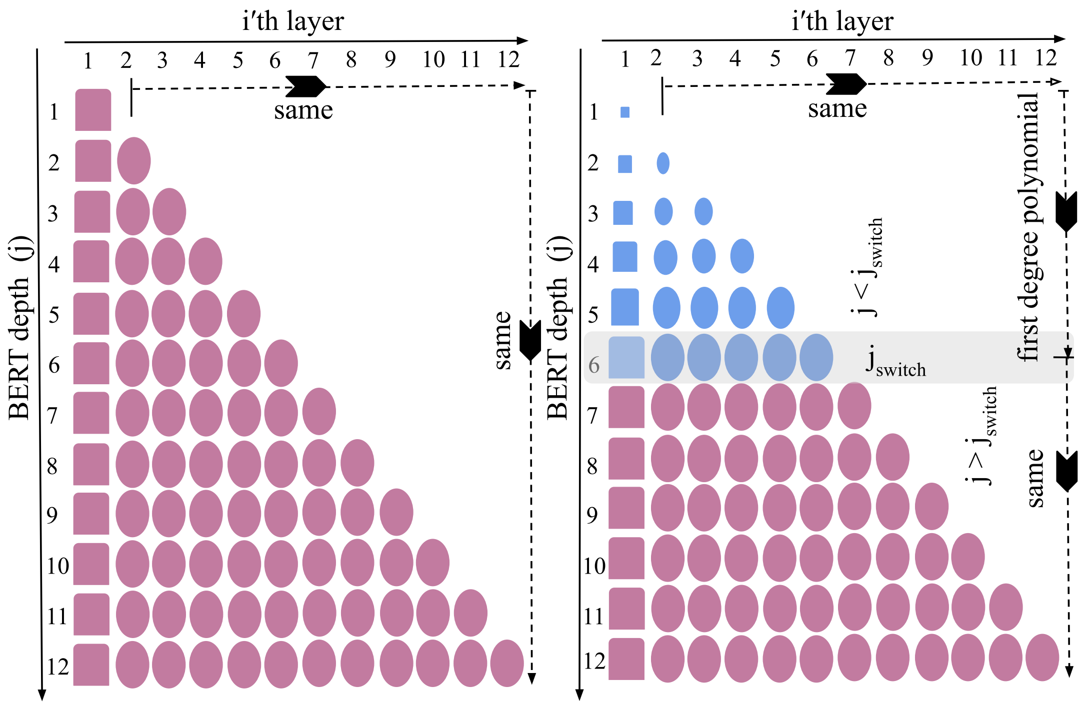
How many BERTs are there?

- Many options between *BERT-tiny* and **BERT-base**
  - Encoder depth L = {*1*, 2, … **12**}
  - Embedding size H = {*128*, 256, 384, …, **768**}
  - Batch size B = {*1*, 2, **4**}
  - Sequence length S = {*64*, 128, 256, 384, **512**}
  - Feed forward network width I = {*512*, 1024, 1536, 2048, **3072**}
  - Attention head count A = {*2*, 4, …, **12**}
- $A = H/64$; $I = 4H$; $B \propto 1/H$
- We consider 4,200 options in total



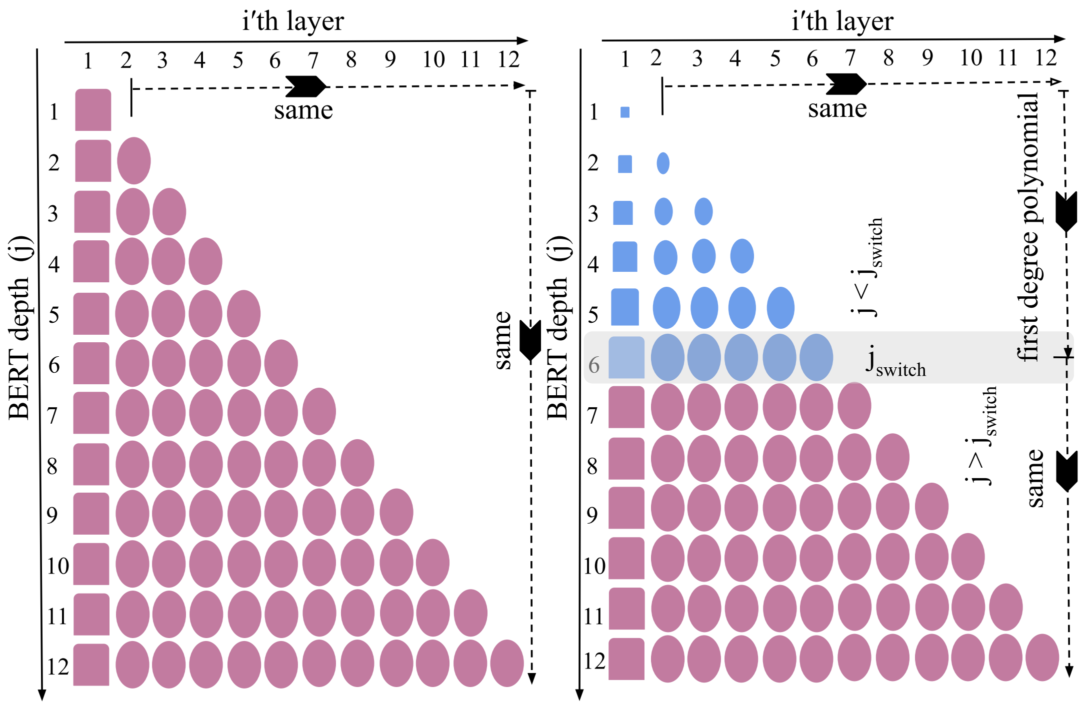
L is for Latency

- BERT latency depends on encoder depth *L*
- The latency of encoder 1 is different from the rest
  - All parameters come from memory
- Later encoders tend to have similar latency
  - Parameters are at least partially cached



L is for Latency

- Two bundle types:
  - Constant layer latency
  - Piece-wise linear latency
- Always measure j={1, 12}
- PWL models: linear first, constant after j_{switch}
  - j_{switch} varies across bundles
  - Binary search!



Experiments and Results

- Design space: 4,200 models
- Latency measured on Kirin 970 big and LITTLE clusters
- BERTPerf can predict all model latencies with <2% error
  - This requires profiling 19% of the design space

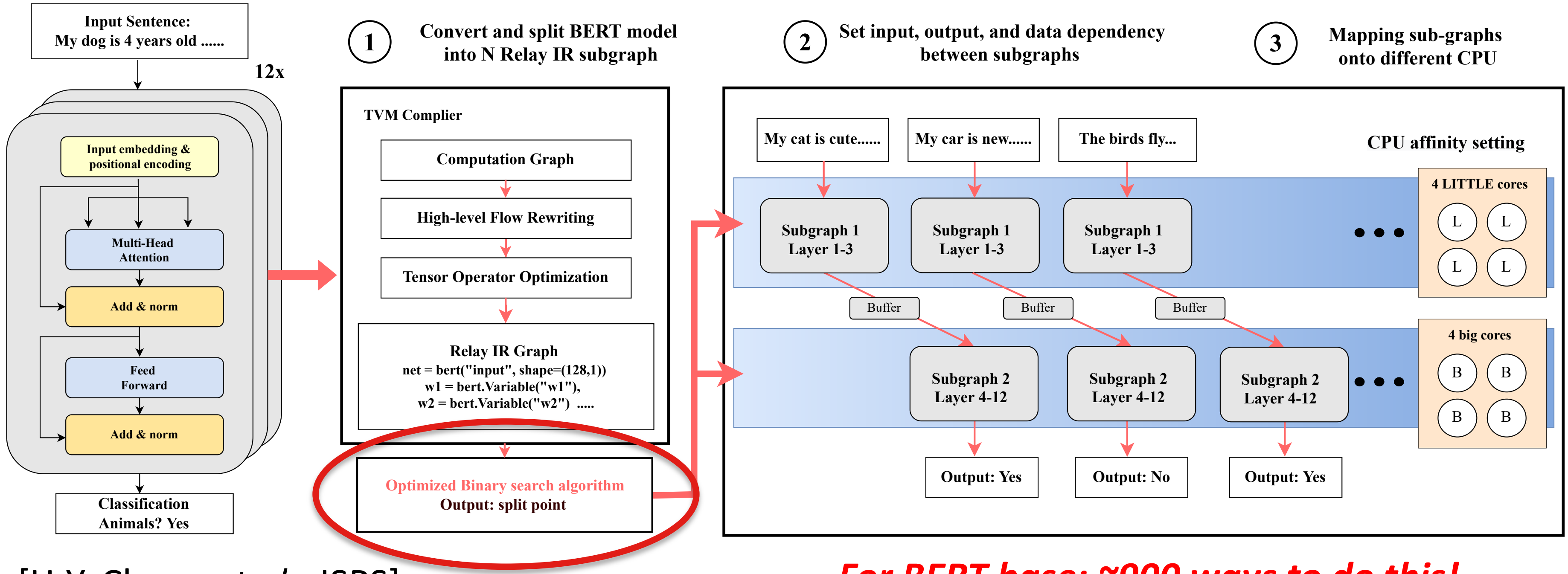
Maximum error (%)	Operator-wise (%)	MLP (%)	BERTPerf (%)
±0.5	19.3	21.4	30.9
±1	28.2	37.5	56.08
±1.5	40.4	48.4	83.09
±2	50.3	56.2	100
±9	94.2	100	-
±13	100	-	-

Observations

- If 5% latency prediction error can be tolerated, profile less
  - 11% of the design space on the big cluster
- LITTLE cluster latency is easier to predict
  - 2% error can be achieved with <10% of the design space
  - Why? Most bundles have constant layer latency– likely due to cache size

PipeBERT: big.LITTLE Pipelining for Edge Throughput

- Edge SoCs are heterogeneous; why not use all CPU cores?



[H-Y. Chang *et al.*, JSPS]

For BERT-base: ~900 ways to do this!

PipeBERT for Better Throughput

- Use binary search with hardware latency feedback to split models
  - Requires ~1% of the time for exhaustive search

BERT models	Homogeneous Throughput (Inference/s)		With PipeBERT Heterogeneous Throughput (Inference/s)	PipeBERT Throughput Improvement (%)
	Big	LITTLE		
BERT-base	0.73	0.43	1.26	72.6
ALBERT	0.67	0.38	1.04	55.2
SqueezeBERT	1.62	0.55	1.94	19.8
MobileBERT	4.98	1.9	5.94	19.3
DistillBERT	1.47	0.91	2.52	71.4
			Average	48.6

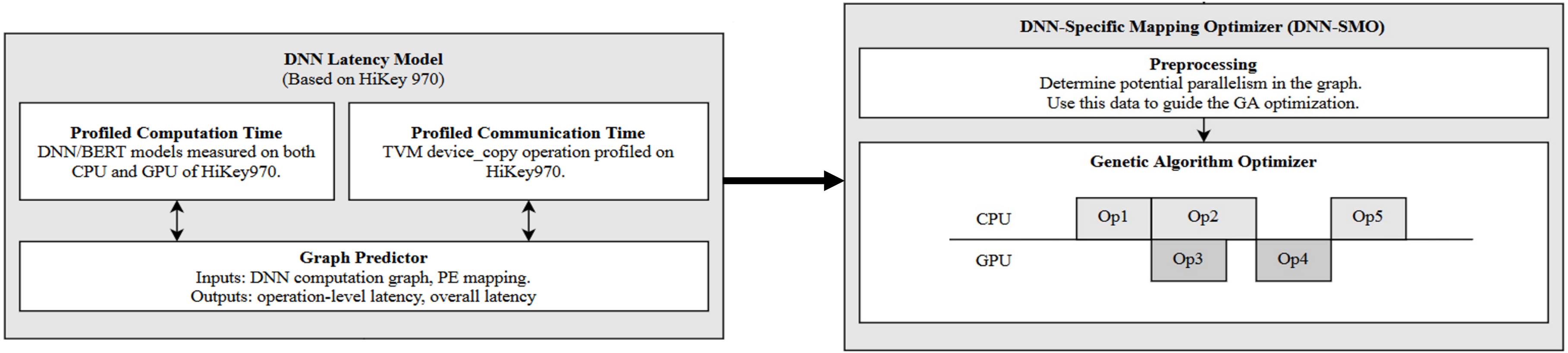
PipeBERT for Better Energy Efficiency Trade-offs

- Least energy per inference? LITTLE cluster: up to about 30% impr
- Best energy-delay trade-off? PipeBERT: 60% impr on average

BERT models	Average active power (W)			Energy efficiency (inference/J)			Energy-delay product (J×s)		
	4B	4L	PipeBERT	4B	4L	PipeBERT	4B	4L	PipeBERT
BERT-base	4.79	1.32	5.69	0.15	0.31	0.22	8.99	7.14	3.59
ALBERT	4.75	1.44	5.67	0.14	0.25	0.19	10.58	9.97	4.96
SqueezeBERT	5.21	1.09	5.35	0.31	0.50	0.35	1.99	3.60	1.42
MobileBERT	5.16	0.97	4.39	0.97	1.89	1.33	0.21	0.27	0.15
DistilBERT	4.71	1.54	5.43	0.31	0.55	0.46	2.18	2.17	0.86

Fast Heterogeneous Task Mapping for Edge Latency

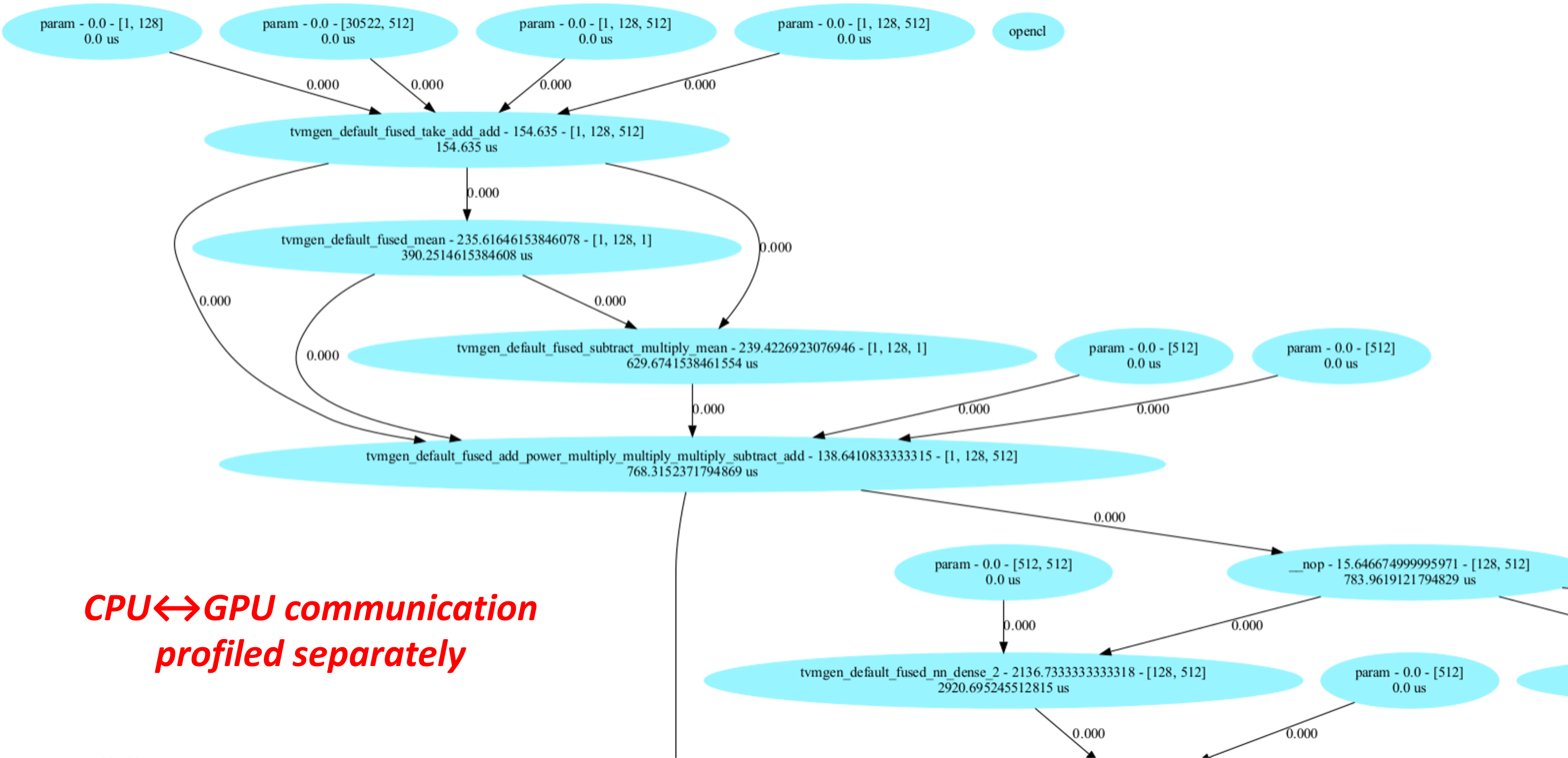
- Edge SoCs are heterogeneous; why not use the CPU and GPU?



For BERT-base: 2^{50} ways to do this!

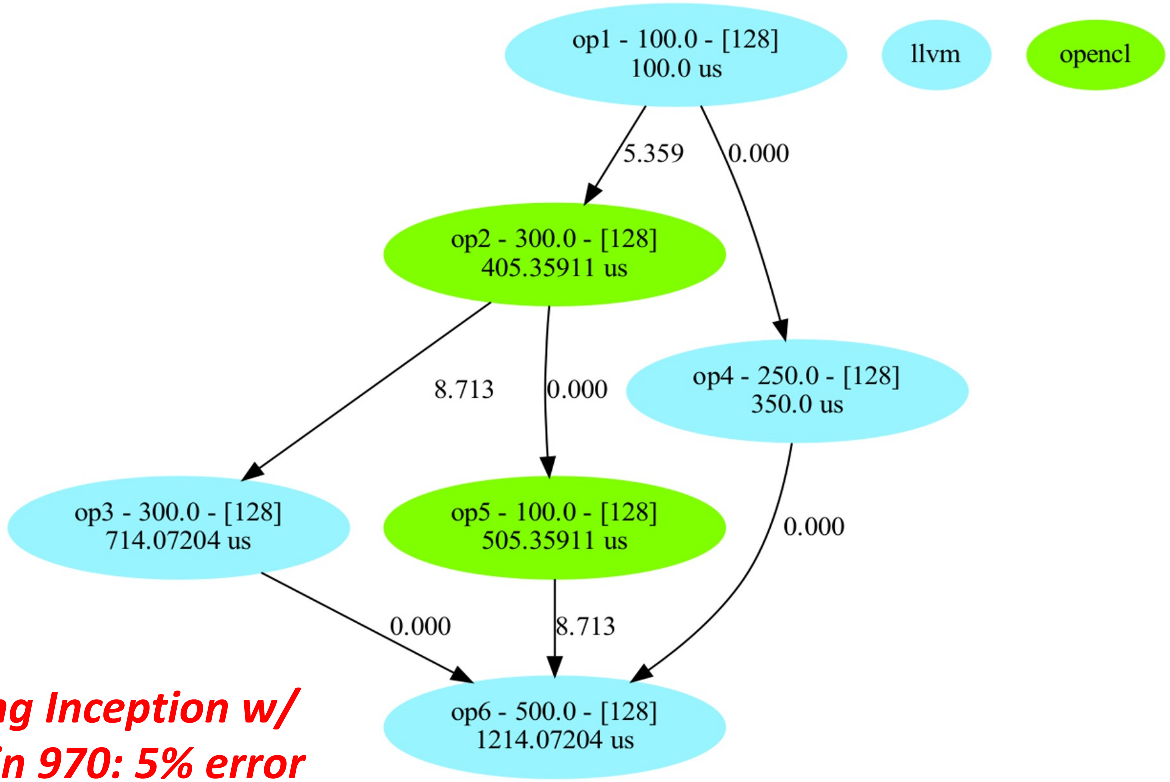
[M.L. Kornelsen *et al.*, ASAP'22]

Profiling with Apache TVM



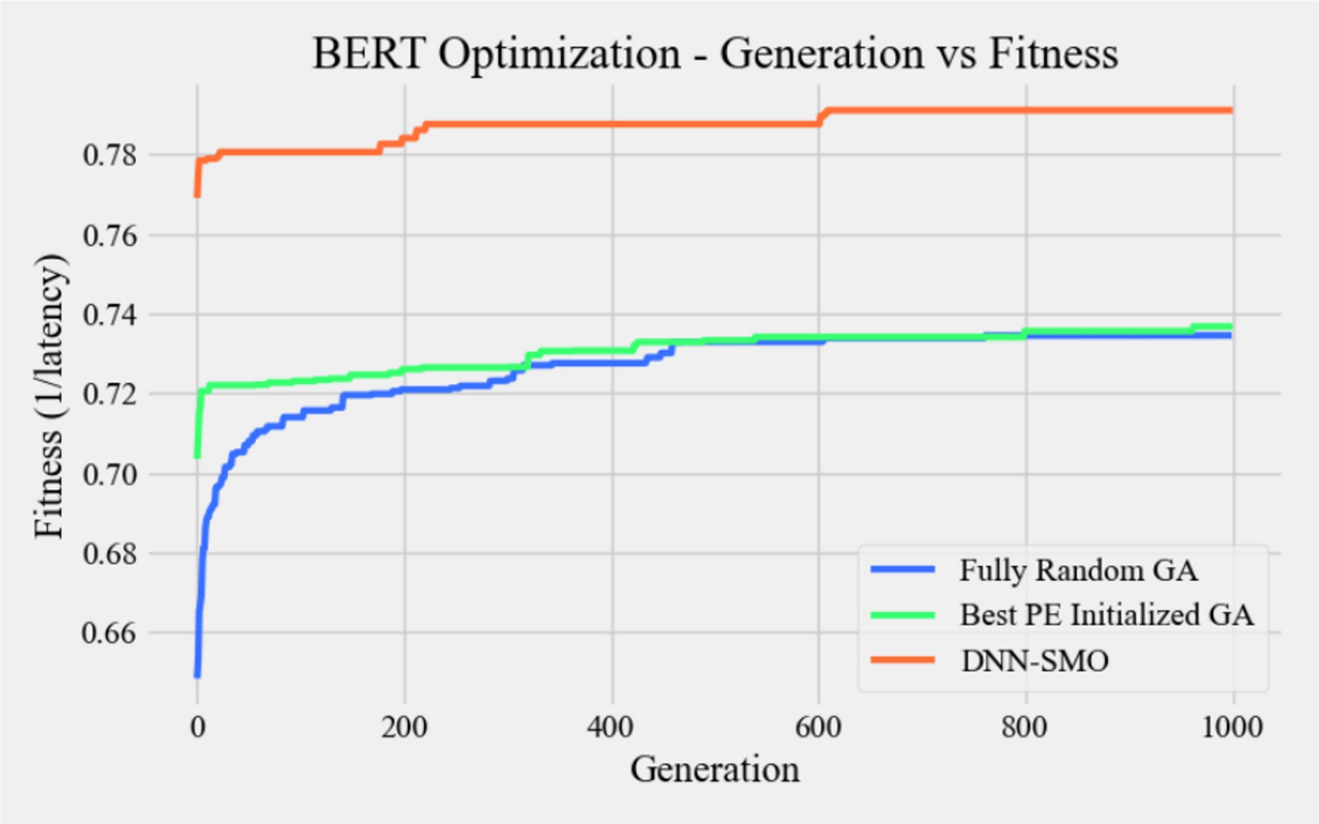
CPU↔GPU communication profiled separately

Predict Latency by Composing Layer/Comm Latency



Validated using Inception w/ ARMCL on Kirin 970: 5% error

Experiments and Results

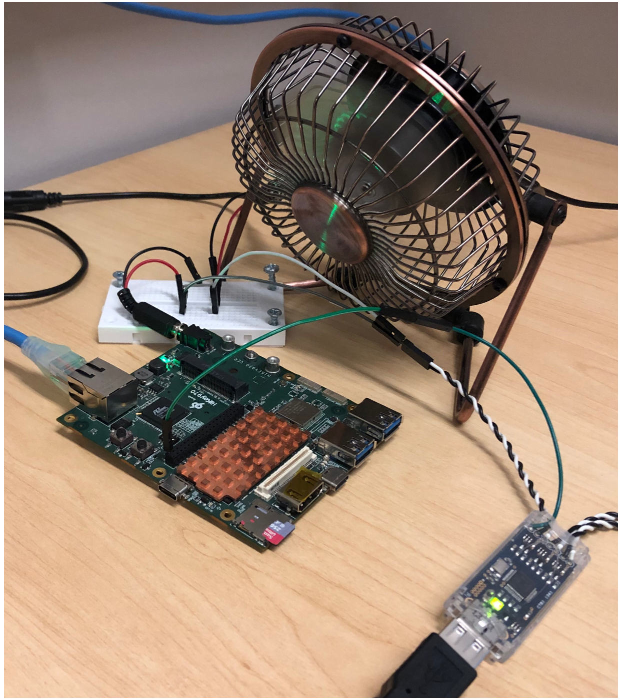


Observations

- BERT models have large sequential components (e.g., FFN)
  – 10-15% improvement for BERT
  – Latency improvement improves with model size
- Other models have parallelism that is more easily exploited
  – 24% improvement for SqueezeBERT
  – 31% improvement for Inception

So you want to find the optimal, redux

- The design space matters a lot
  - How to identify a set of candidates?
  - How do you sample it to build an estimator?
  - How do you explore it? Algorithms matter
  - *Training* (i.e., pre-training, fine-tuning) *is time consuming; avoid it!*
- Measurement on hardware is difficult
  - Profiling tools are limited
  - Must isolate the energy consumed on cores
  - Controlling the system for stable measurement requires substantial effort
  - *Measurement is time consuming; avoid it!*



So you want to find the optimal, redux

- OTS models are not always available in the format you want
  - ONNX? PyTorch? TVM? TFLite? TensorRT?
  - *Conversion is a minefield!*
- BERT (c. 2018!) operations are not uniformly supported
  - Apache TVM: great* for multicore CPU; poor performance for GPU
  - ARMCL: requires manual development of BERT components
  - TFLite: BERT not supported for Mali GPU
  - *Everything is broken; pick your poison!*

So you want to find the optimal, redux

- Quantization varies by toolchain and hardware architecture
  - TensorRT: INT8, FP16, FP32, but only for NVIDIA GPUs; poor accuracy
  - ONNX: INT8, FP16, FP32 for CPUs; only FP16 and FP32 for GPUs
  - TVM: *quantization does not improve inference latency*
- Documentation? What documentation?
  - *Yeah, there isn't any*, for either edge devices or software toolkits

Want to learn more? Check out our papers!

- M. Abdelgawad *et al.*, BERTPerf: Inference Latency Predictor for BERT on ARM big.LITTLE Multi-Core Processors, at *SIPS'22*
- H-Y. Chang *et al.*, PipeBERT: High-throughput BERT Inference for ARM Big.LITTLE Multi-core Processors, in press for *J. Signal Process. Syst.*
- M. L. Kornelsen *et al.*, Fast Heterogeneous Task Mapping for Reducing Edge DNN Latency, at *ASAP'22*

Or, check out our posters!

- **Hung-Yang Chang**: *NAS plus Pipeline for High Throughput Edge Inference BERT*
- **Negin Firouzian**: *Latency and Accuracy Predictors for Efficient BERT Hardware-aware NAS*
- **Murray Kornelsen**: *ARMCL BERT: Novel Quantizable BERT Implementation for ARM SoCs*
- **Lily Li**: *BERT Inference Energy Predictor for Efficient Hardware-aware NAS*
- **Dr. S. Hasan Mozafari**