Exciting Directions for ML Models and the Implications for Computing Hardware

Jeff Dean
Chief Scientist
ai.google/research/people/jeff

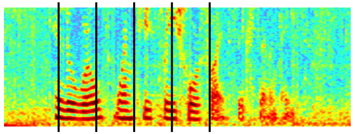
Amin Vahdat
VP ML, Systems & Cloud AI
ai.google/research/people/AminVahdat

*Presenting the work of **many** people at Google*

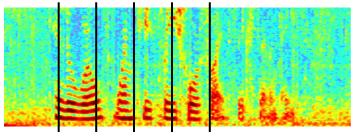
Some observations

- In recent years, ML has completely changed our expectations of what is possible with computers
- Increasing scale (compute, data, model size) delivers better results
- The kinds of computations we want to run and the hardware on which we run them is changing dramatically

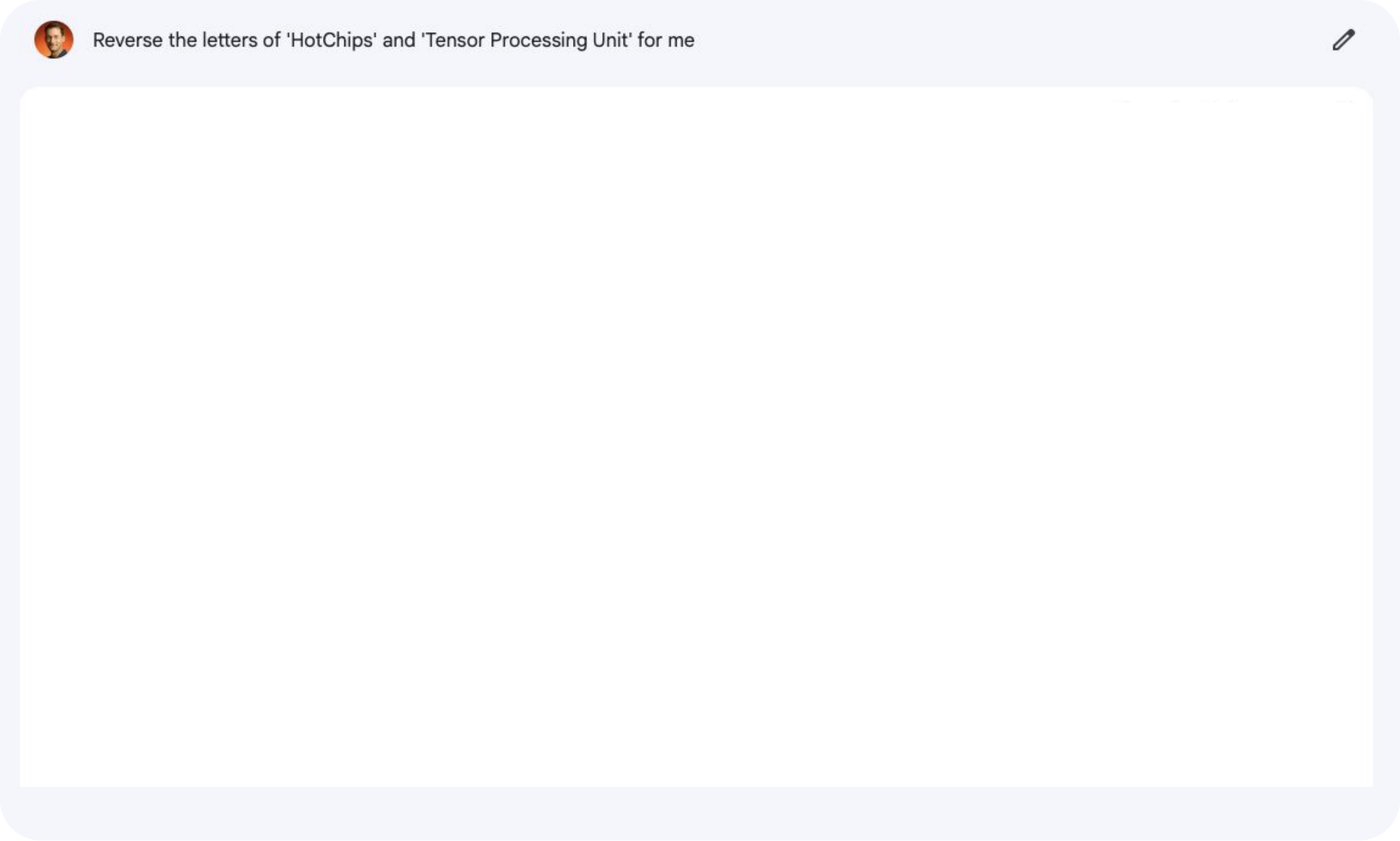
A decade of amazing progress in what computers can do

Input	Output
Pixels:	“leopard”
Audio: 	“How cold is it outside?”
“Hello, how are you?”	“Bonjour, comment allez-vous?”
Pixels:	“A cheetah lying on top of a car”

A decade of amazing progress in what computers can do

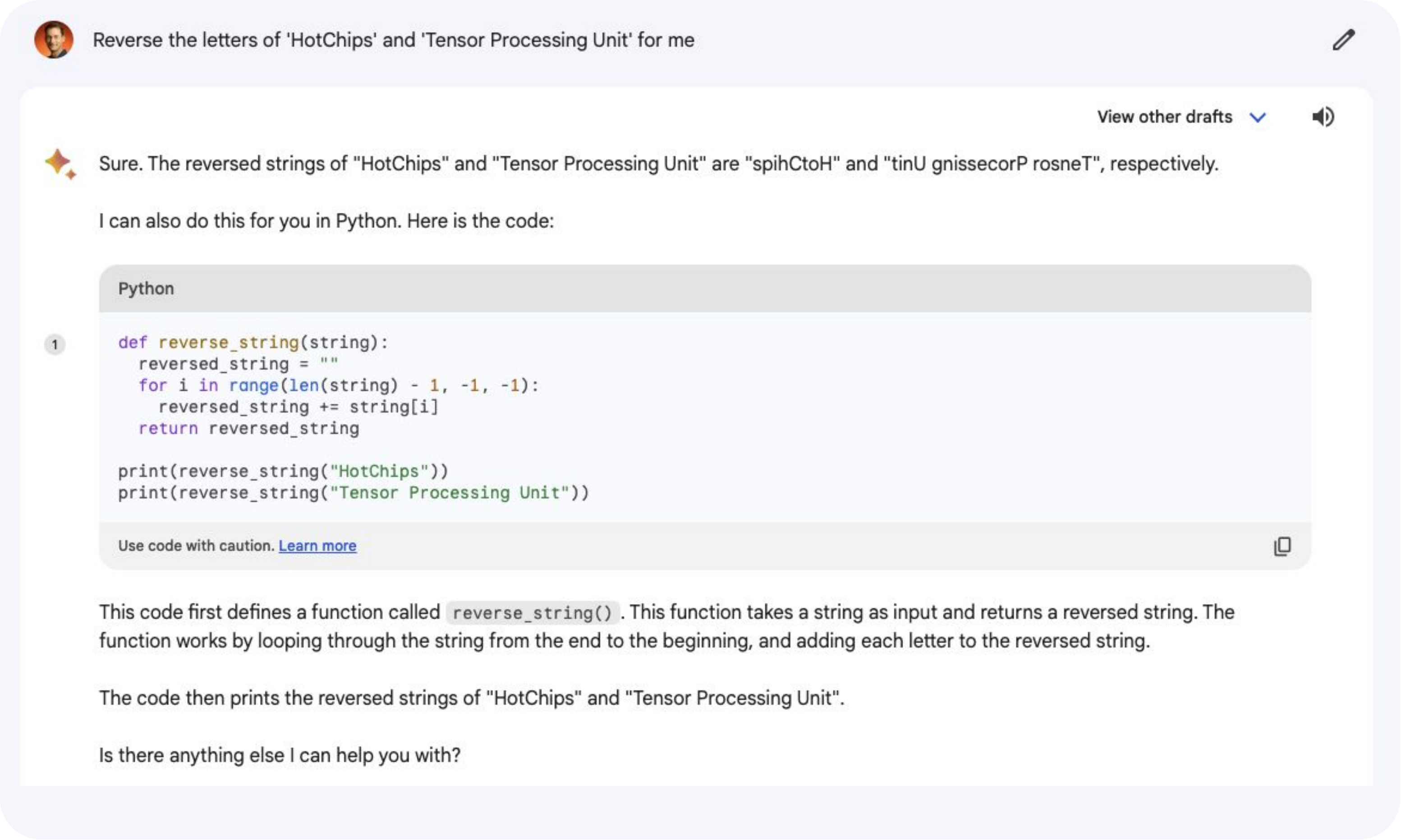
Output	Input
Pixels:	“leopard”
Audio: 	“How cold is it outside?”
“Hello, how are you?”	“Bonjour, comment allez-vous?”
Pixels:	“A cheetah lying on top of a car”

bard.google.com



Reverse the letters of 'HotChips' and 'Tensor Processing Unit' for me

bard.google.com



Reverse the letters of 'HotChips' and 'Tensor Processing Unit' for me

Sure. The reversed strings of "HotChips" and "Tensor Processing Unit" are "spihCtoH" and "tinU gnissecorP rosneT", respectively.

I can also do this for you in Python. Here is the code:

```python
def reverse_string(string):
  reversed_string = ""
  for i in range(len(string) - 1, -1, -1):
    reversed_string += string[i]
  return reversed_string

print(reverse_string("HotChips"))
print(reverse_string("Tensor Processing Unit"))
```

Use code with caution. Learn more

This code first defines a function called `reverse_string()`. This function takes a string as input and returns a reversed string. The function works by looping through the string from the end to the beginning, and adding each letter to the reversed string.

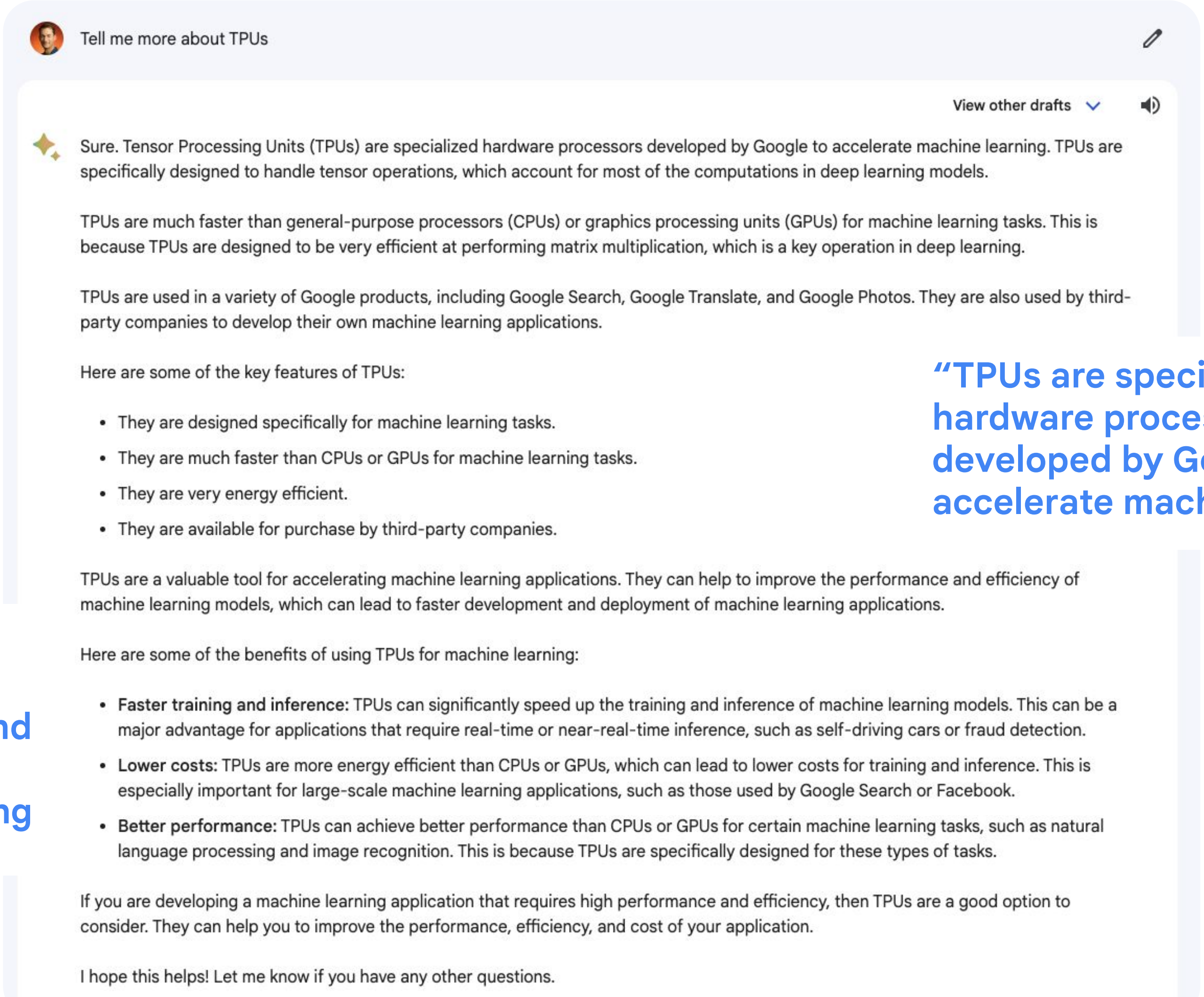
The code then prints the reversed strings of "HotChips" and "Tensor Processing Unit".

Is there anything else I can help you with?

bard.google.com

Tell me more about TPUs

bard.google.com



Tell me more about TPUs

Sure. Tensor Processing Units (TPUs) are specialized hardware processors developed by Google to accelerate machine learning. TPUs are specifically designed to handle tensor operations, which account for most of the computations in deep learning models.

TPUs are much faster than general-purpose processors (CPUs) or graphics processing units (GPUs) for machine learning tasks. This is because TPUs are designed to be very efficient at performing matrix multiplication, which is a key operation in deep learning.

TPUs are used in a variety of Google products, including Google Search, Google Translate, and Google Photos. They are also used by third-party companies to develop their own machine learning applications.

Here are some of the key features of TPUs:

- They are designed specifically for machine learning tasks.
- They are much faster than CPUs or GPUs for machine learning tasks.
- They are very energy efficient.
- They are available for purchase by third-party companies.

TPUs are a valuable tool for accelerating machine learning applications. They can help to improve the performance and efficiency of machine learning models, which can lead to faster development and deployment of machine learning applications.

Here are some of the benefits of using TPUs for machine learning:

- **Faster training and inference:** TPUs can significantly speed up the training and inference of machine learning models. This can be a major advantage for applications that require real-time or near-real-time inference, such as self-driving cars or fraud detection.
- **Lower costs:** TPUs are more energy efficient than CPUs or GPUs, which can lead to lower costs for training and inference. This is especially important for large-scale machine learning applications, such as those used by Google Search or Facebook.
- **Better performance:** TPUs can achieve better performance than CPUs or GPUs for certain machine learning tasks, such as natural language processing and image recognition. This is because TPUs are specifically designed for these types of tasks.

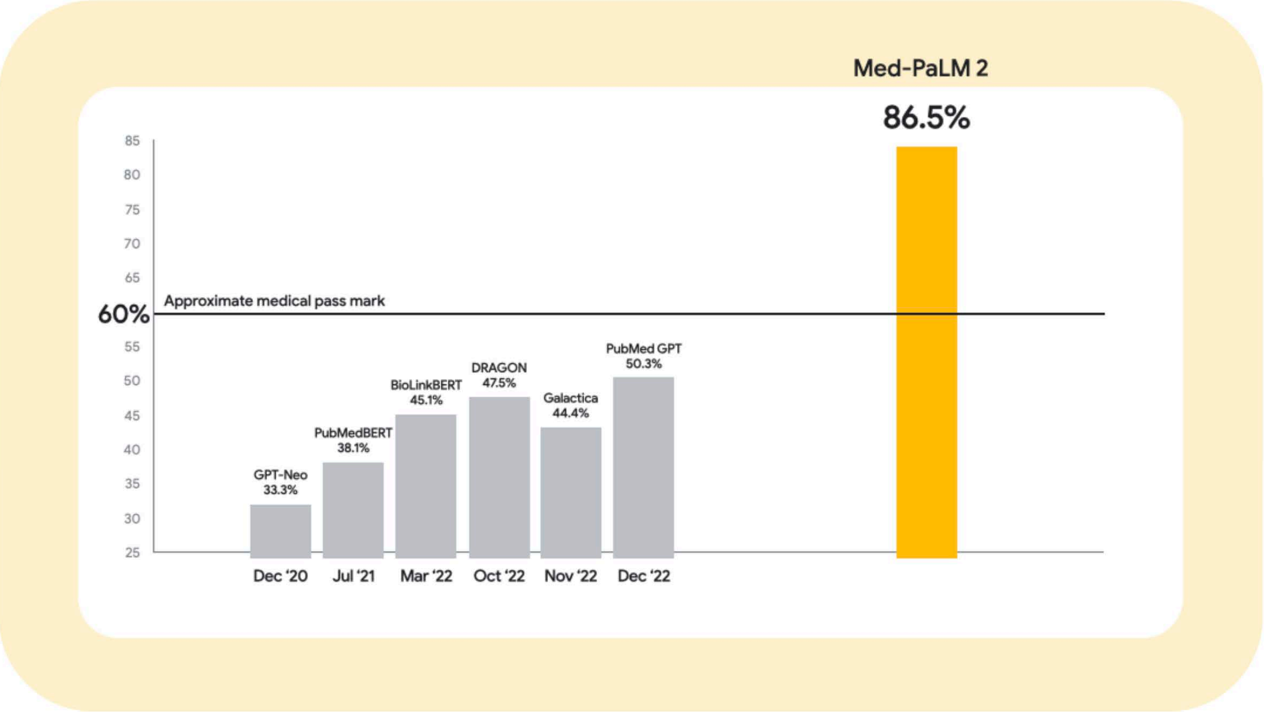
If you are developing a machine learning application that requires high performance and efficiency, then TPUs are a good option to consider. They can help you to improve the performance, efficiency, and cost of your application.

I hope this helps! Let me know if you have any other questions.

“TPUs are specialized hardware processors developed by Google to accelerate machine learning.”

“They can help improve the performance and efficiency of machine learning models…”

Med-PaLM 2



Towards Expert-Level Medical Question Answering with Large Language Models
Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Le Hou, Kevin Clark, Stephen Pfohl, Heather Cole-Lewis, Darlene Neal, Mike Schaekermann, Amy Wang, Mohamed Amin, Sami Lachgar, Philip Mansfield, Sushant Prakash, Bradley Green, Ewa Dominowska, Blaise Aguera y Arcas, Nenad Tomasev, Yun Liu, Renee Wong, Christopher Semturs, S. Sara Mahdavi, Joelle Barral, Dale Webster, Greg S. Corrado, Yossi Matias, Shekoofeh Azizi, Alan Karthikesalingam and Vivek Natarajan

https://sites.research.google/med-palm/

https://arxiv.org/abs/2305.09617 (Med-PaLM 2 paper)
https://www.nature.com/articles/s41586-023-06291-2 (Med-PaLM 1 paper)

Multimodal models

PaLI: Scaling Language-Image Learning in 100+ Languages
[Link]

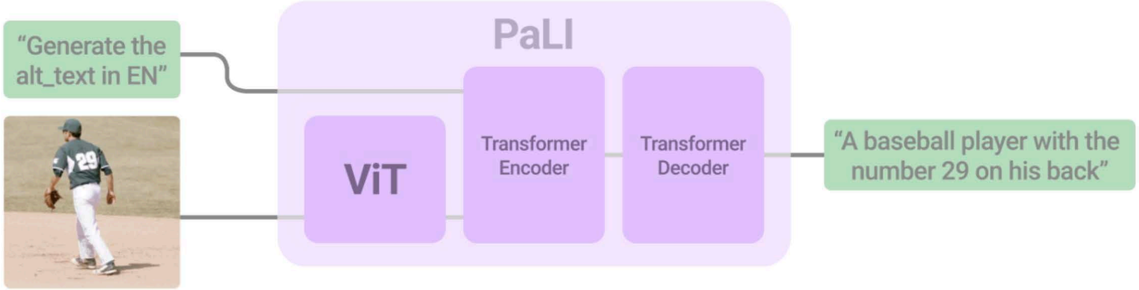

"Generate the alt_text in EN"
PaLI
ViT
Transformer Encoder
Transformer Decoder
"A baseball player with the number 29 on his back"

Imagen: a text-to-image diffusion model
[Link]

Sprouts in the shape of text 'Imagen' coming out of a fairytale book.

A transparent sculpture of a duck made out of glass. The sculpture is in front of a painting of a landscape.

Rest of the talk

- Important trends in ML Models
- Some implications for computer architects
- Designing ML hardware and deploying it to keep up with fast-moving field

Rest of the talk

- Important trends in ML Models
- Some implications for computer architects
- Designing ML hardware and deploying it to keep up with fast-moving field

What is it going to take to deliver major increases in compute capacity & efficiency to continue to advance the field of ML?

Important trends in ML models

Sparsity

Adaptive computation

Dynamically-changing neural networks

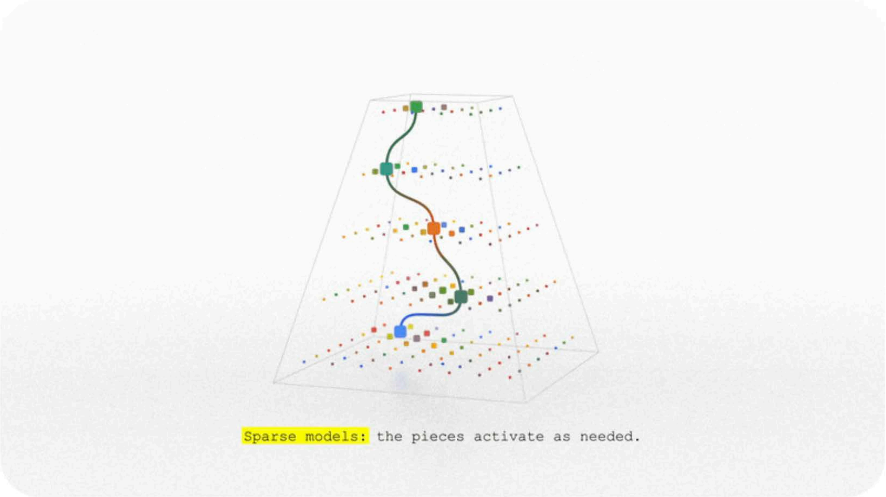
Dense models

Focus of vast majority of ML community

Whole model activated for each input example or token

Sparse computation

Sparse models have different pathways that are adaptively called upon as needed

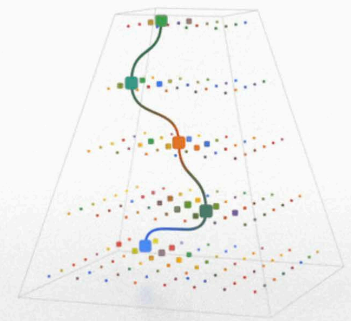


Sparse models: the pieces activate as needed.

Sparse computation - Why?

By activating tiny part of overall model for each example:

- Can be **much more efficient** (just call upon right pieces of overall model)
- **Different parts of model are specialized** for different kinds of inputs
- **Touch just the right 1% or 10% of large model:** improved responsiveness



`Sparse models:` the pieces activate as needed.

Coarse-grained vs. Fine-grained sparsity

Coarse-grained sparsity

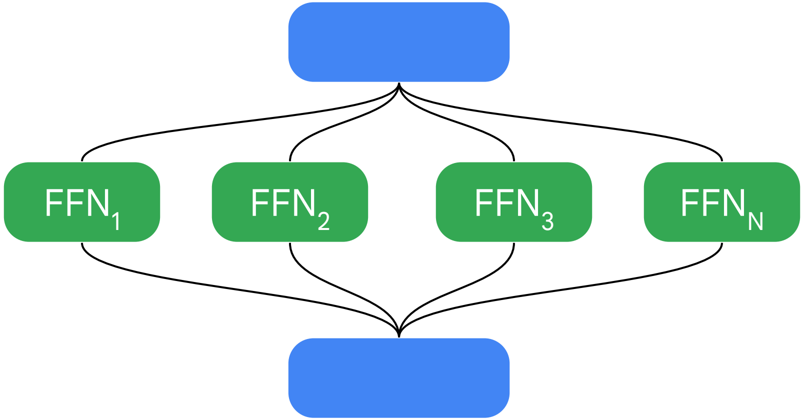
Large modules that are either activated or not

Fine-grained sparsity

Sparsity within a single vector or tensor (e.g. where 1 or 2 of every 4 values are 0). Modern hardware starting to support this

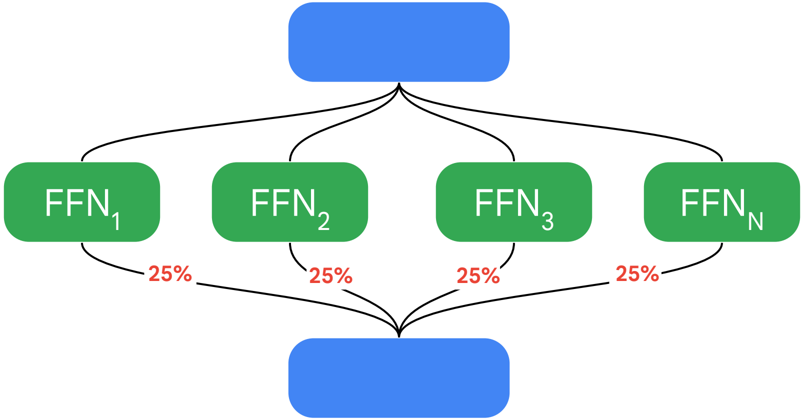
Fine-grained & coarse-grained sparsity are complementary

Most sparsity work today uses same size and structure for each expert



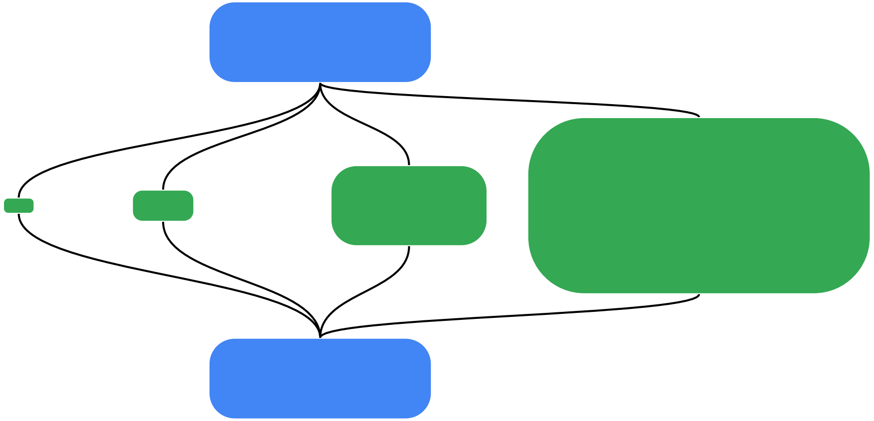
Most sparsity work today uses same size and structure for each expert

Computational balance achieved by equal size computation per expert and equal flow of # of examples to each expert

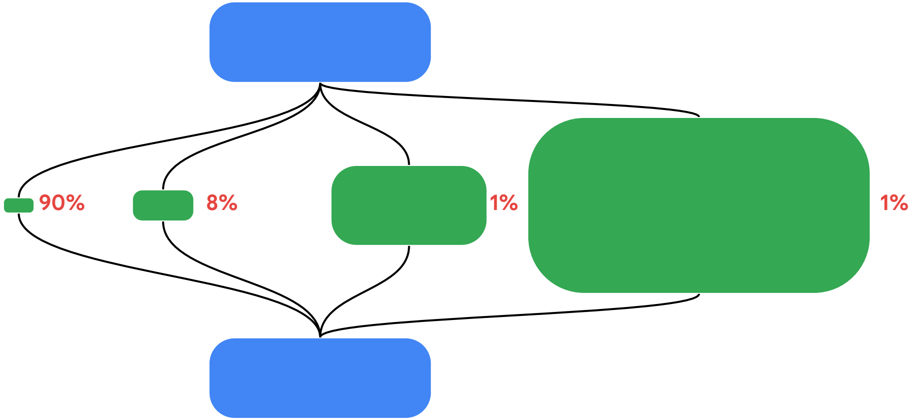


All-to-all shuffle performance across accelerators important

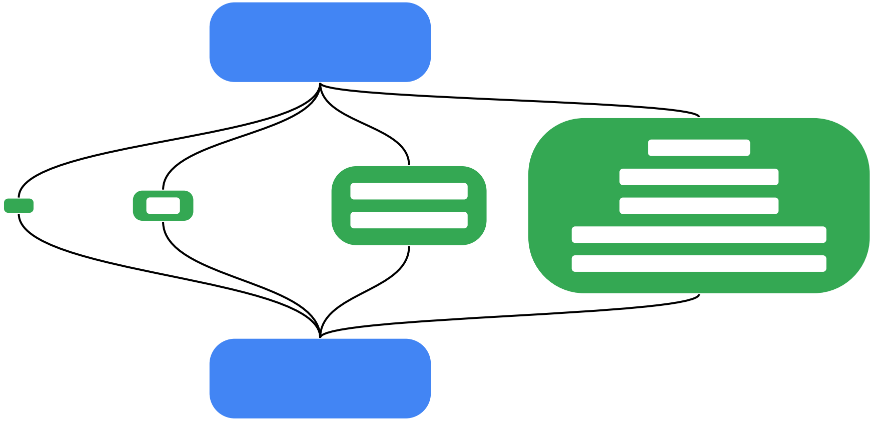
Varying computational costs?



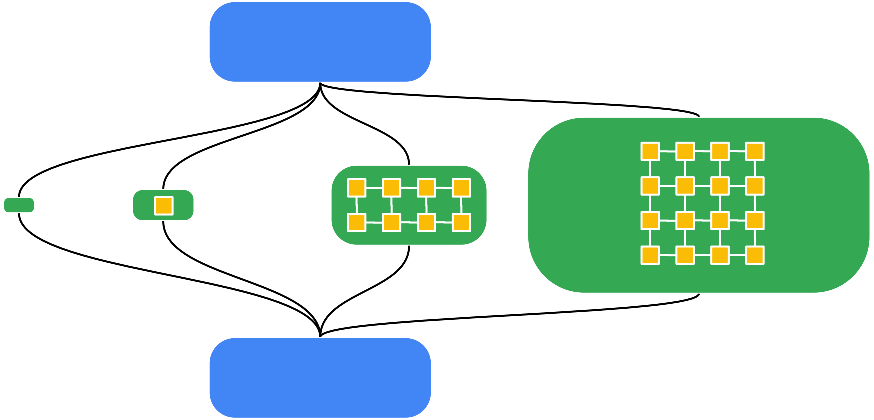
Varying computational costs?



Varying expert structure?

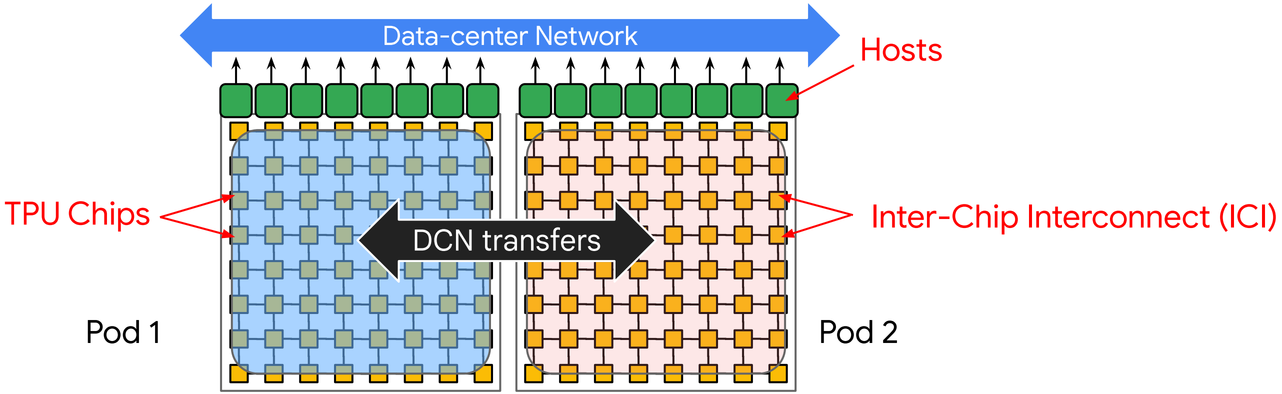


Varying computational costs?



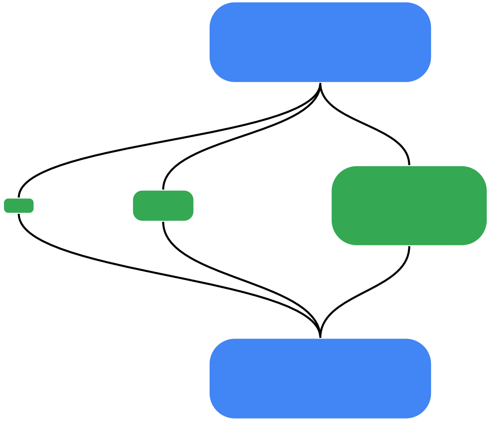
Pathways: Scalable system for flexible ML models

- Flexible mapping of components (pieces of ML computation) onto collection of physical computational devices
- Can dynamically add or remove resources to running system
- Manages communication across multiple kinds of network transports (ICI, DCN, ...)
- Highly scalable: PaLM language model trained across multiple TPUv4 Pods using Pathways

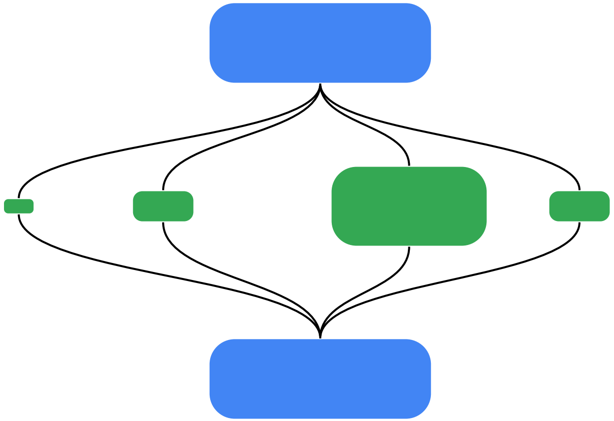


Pathways, MLSys 2022: arxiv.org/abs/2203.12533 PaLM language model, JMLR: https://arxiv.org/abs/2204.02311

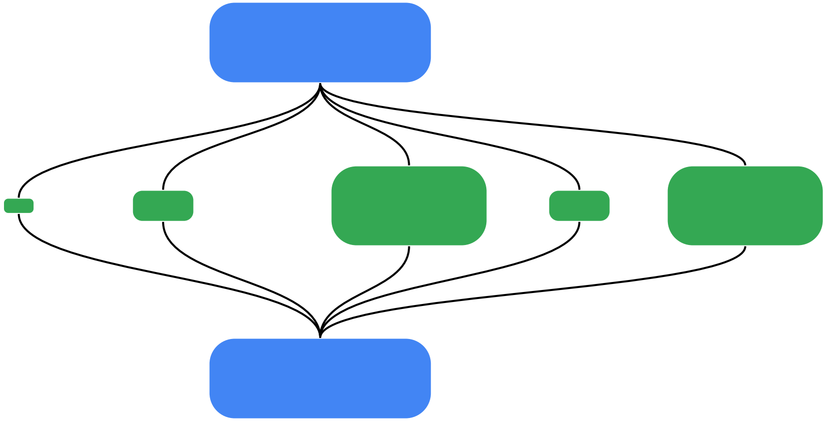
Dynamic introduction of new model capacity



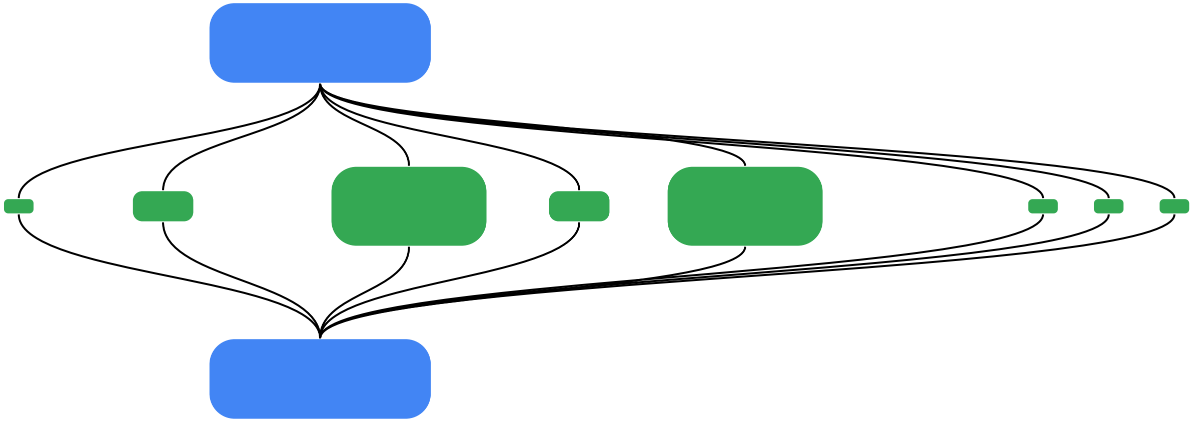
Dynamic introduction of new model capacity



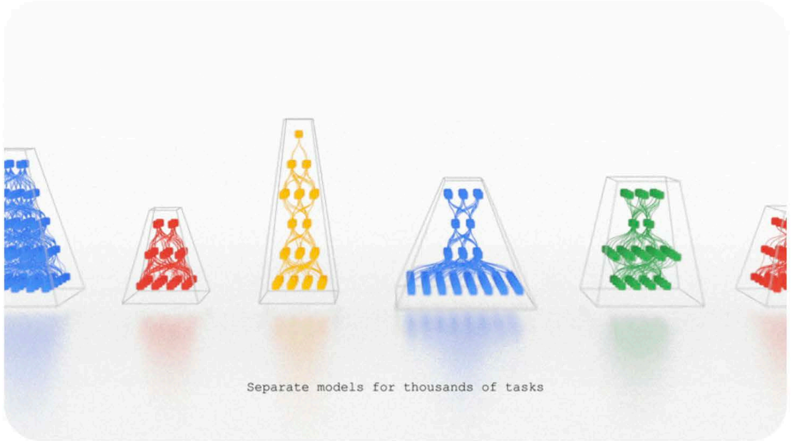
Dynamic introduction of new model capacity



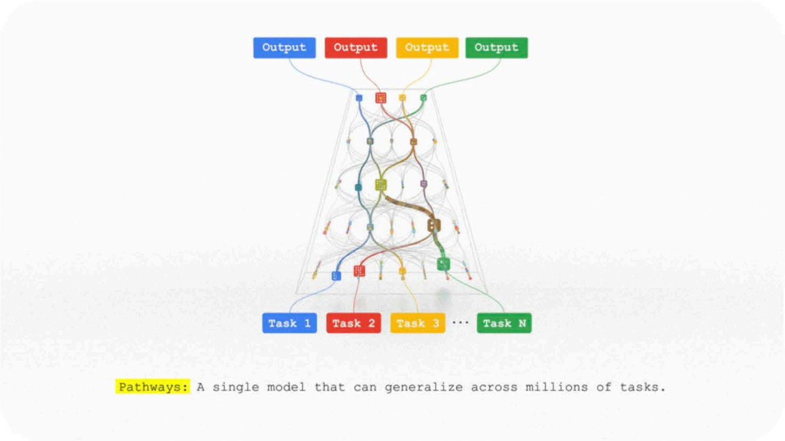
Dynamic introduction of new model capacity



Where are we headed?



Separate models for different tasks

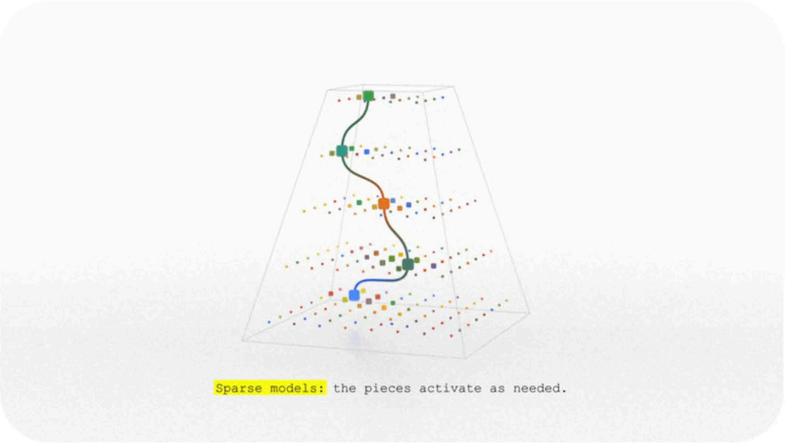

Output
Output
Output
Output
Task 1
Task 2
Task 3
...
Task N
Pathways: A single model that can generalize across millions of tasks.

Single model that can generalize across millions of tasks

Where are we headed?

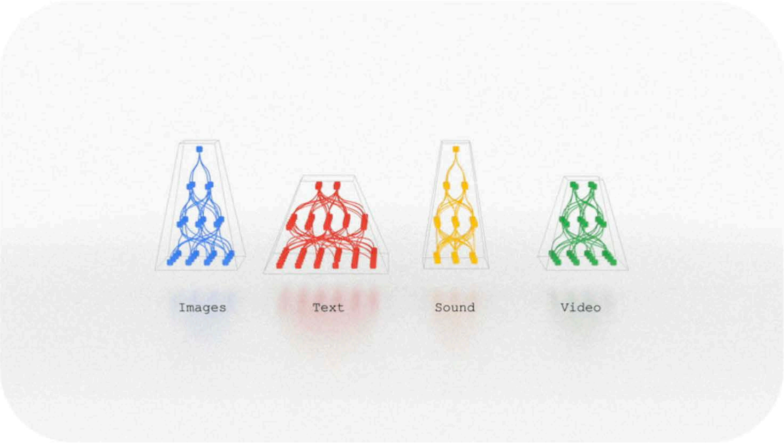
Dense models: the whole model activates.

Dense models

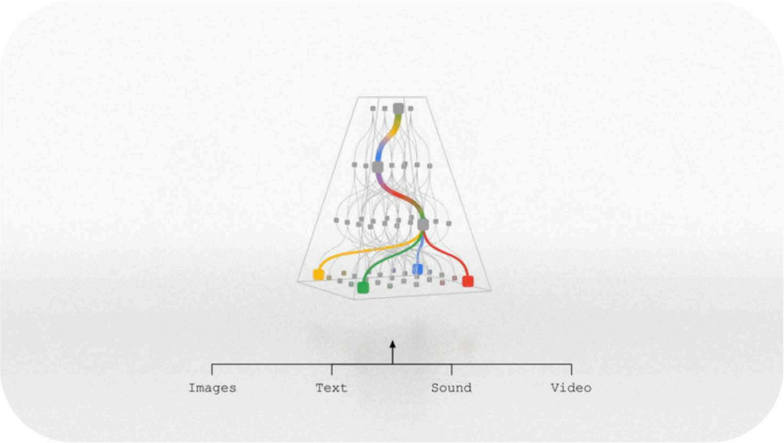


Efficient sparse models

Where are we headed?



Single modality models



Models that deal with many modalities

Key takeaways for computer architects & system builders

- → Connectivity of accelerators (BW and latency) matters
- → Scale matters for both training and inference
- → Sparse models put pressure on memory capacity and efficient routing
- → ML software must make it easy to express interesting models
- → Power, sustainability and reliability really matter

CO_2e* emissions of machine learning training

- Lots of attention and dramatic headlines
- Lots of misinformation
- Important topic: critical to use actual data to focus on the right things

*CO_2e = Carbon Dioxide Equivalents

Example of misinformation in this space

Energy and Policy Considerations for Deep Learning in NLP by Strubell *et al.* published in 2019 (cited >2300 times) attempted to estimate CO$_2$e* emissions of Evolved Transformer neural architecture search (NAS) run by So *et al.*

Unlike other data in their paper, this was estimated not measured

1. Modeled P100 not TPU v2 (where computation was actually run), and US average DC not Google DC: **actual NAS was ~5X lower**
2. Assumed use of full size model, not small proxy size model for search (despite description in So *et al.*): **actual NAS was ~19X less compute/emissions due to this error**
3. Misunderstood that **NAS is a one-time cost, not an every-problem cost**

● **Arrived at flawed estimate of 284t of CO$_2$e for the Evolved Transformer NAS**

The Evolved Transformer, So *et al.*, https://arxiv.org/abs/1901.11117
Energy and Policy Considerations for Deep Learning in NLP, Strubell *et al.*, ACL 2019, https://arxiv.org/abs/1906.02243

*CO$_2$e = Carbon Dioxide Equivalents

MIT Technology Review

Training a single AI model can emit as much carbon as five cars in their lifetimes

Deep learning has a terrible carbon footprint.

NewScientist

Creating an AI can be five times worse for the planet than a car

---

Environmental cost to improve ML task (2024)?*

*"The answers are grim: Training such a model would cost **US $100 billion** and would **produce as much carbon emissions as New York City does in a month**. And if we estimate the computational burden of a 1 percent error rate, the results are considerably worse."*

Thompson *et al.*, [Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable](), Oct 2021, *IEEE Spectrum*

Fortunately, with correct data, things are not so dire!

1. The actual **one-time** Evolved Transformer NAS search done by So et al. on TPU v2 hardware in a Google datacenter in Georgia generated 3.2t of CO_2e*, not 284t of CO_2e* **(~88X less)**

Carbon Emissions and Large Neural Network Training, David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, https://arxiv.org/abs/2104.10350

The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink, David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, IEEE Computer, https://www.techrxiv.org/ndownloader/files/34128165

*CO_2e = Carbon Dioxide Equivalents

Fortunately, with correct data, things are not so dire!

1. The actual **one-time** Evolved Transformer NAS search done by So et al. on TPU v2 hardware in a Google datacenter in Georgia generated 3.2t of CO_2e*, not 284t of CO_2e* **(~88X less)**

2. The discovered Evolved Transformer model is a **drop-in replacement for the plain Transformer** and **uses 16-25% less energy to reach same accuracy.** It is **open sourced** for all to use:
   - ➡ github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/evolved_transformer.py

Carbon Emissions and Large Neural Network Training, David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, https://arxiv.org/abs/2104.10350

The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink, David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, IEEE Computer, https://www.techrxiv.org/ndownloader/files/34128165

*CO_2e = Carbon Dioxide Equivalents

Fortunately, with correct data, things are not so dire!

1. The actual **one-time** Evolved Transformer NAS search done by So et al. on TPU v2 hardware in a Google datacenter in Georgia generated 3.2t of CO_2e*, not 284t of CO_2e* **(~88X less)**

2. The discovered Evolved Transformer model is a **drop-in replacement for the plain Transformer** and **uses 16-25% less energy to reach same accuracy.** It is **open sourced** for all to use:
   - ➡ github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/evolved_transformer.py

3. Training an NLP model of the scale examined by Strubell *et al.* using the discovered Evolved Transformer on ML efficient hardware in a Google datacenter in Iowa takes 120 TPUv2 hours, costs \$40, and generates 0.0024t of CO_2e*, not 284t of CO_2e*
(2.4 kg, ~118,000X less)

Carbon Emissions and Large Neural Network Training, David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, https://arxiv.org/abs/2104.10350

The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink, David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, IEEE Computer, https://www.techrxiv.org/ndownloader/files/34128165

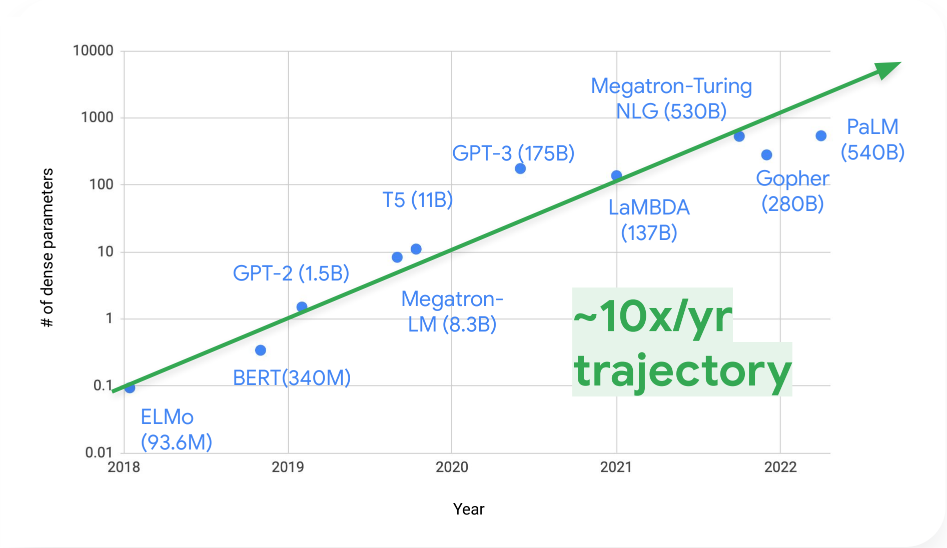
*CO_2e = Carbon Dioxide Equivalents

System goodput, power, reliability and CO_2e* should be primary benchmarking and design targets

Many contributions to this section, including: Houle Gan, Sebastian Lobo, Xiaoyu Ma, Ram Padmanabhan, Dave Patterson, Mukarram Tariq, Parthasarathy Ranganathan

*CO_2e = Carbon Dioxide Equivalents

The demand for ML compute is growing exponentially



Quality continues to improve with number of dense parameters for foundational models

- Required computing power growing super linearly with dense model size
- Synchronous, parallel computation requires collectives→high speed, low latency network interconnect

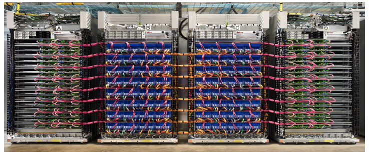
Accelerated computing with
TPU Supercomputing

judicious specialization + application codesign

10-100x
system efficiency
(perf, power, cost)

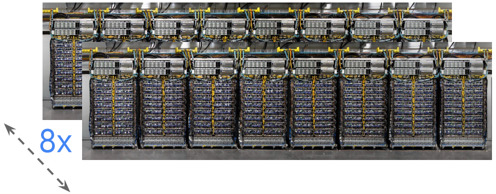
- Synchronous, **high bandwidth interconnect, parameter distribution**
- **Liquid cooling** for maximum system efficiency
- **Specialized data representation** (e.g., int4 to bf16)
- **Optical Circuit Switching** for fault tolerance and efficient scheduling
- **Specialized hardware** for **dense matrix** multiplication
- **DRAM**→ High Bandwidth Memory stacked up to 10x bandwidth
- **Specialized hardware** for **scatter/gather** in sparse operations

Rapid innovation with Cloud TPU AI Supercomputers



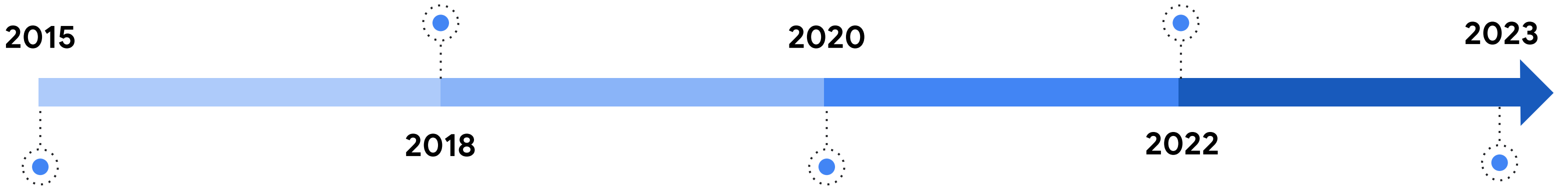
Cloud TPU v2

Domain-specific AI supercomputing
256 chips distributed shared memory



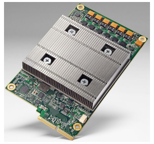
Cloud TPU v4

Optically reconfigurable 3D Torus
4k chips with distributed shared memory



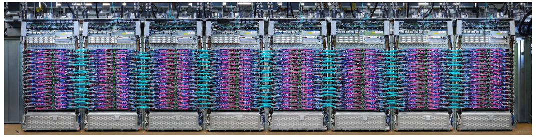
TPU v1

Internal Inference accelerator



Cloud TPU v3

Liquid cooled
1k chips distributed shared memory



Google Cloud
Next '23

Accelerated computing with specialized hardware has gotten us a massive factor but this is **no longer enough…**

Implications (or how to deliver the next 100x):

- → Optimize for **systems goodput, power, reliability, and CO_2e***
- → Next generation of horizontal scaling
- → Algorithmic innovation and software/hardware codesign

*CO_2e = Carbon Dioxide Equivalents

The problem with current metrics

Traditionally, hardware evaluated in terms of **“Chip Perf”/$** within a fixed power budget

- Higher power ok as long as it meets reliability and heat dissipation requirements and can be air cooled within a fixed space
- Chip perf is often simplistic view of headline numbers (e.g., max FLOPS, SpecInt), does not account for systems cost
- ML perf reports absolute performance at a given system size
  It does not yet account for systems cost, CO_2e*, or efficiency (and power is optional)

*CO_2e = Carbon Dioxide Equivalents

Defining systems Perf/TCO

“Perf/TCO”

Reflecting infrastructure cost-performance is today’s primary metric to evaluate **new architectural designs** to be deployed in **3-5 years**

Performance normalized to *Total Cost of Ownership* (*TCO*), or *Perf/TCO*, has been the primary criteria for architectural evaluation

TCO = CapEx + OpEx

CapEx (Capital Expenditure)= one-time investment to build compute HW and physical infrastructure
OpEx (Operational Expenditure) = recurring cost paid during the life cycle of the servers:

OpEx = **DC Provisioning Cost** + Electricity Cost

DC Provisioning Cost = #years × **TDP** × $/Provisioned Watt
Electricity Cost = #years × **consumed power** × $/Consumed Watt

Perf/TCO hidden assumptions

1. There is enough DC capacity to house new compute and it is ok not to idle some provisioned power capacity
2. Consumed power can be accurately attributed back to individual workloads
3. Performance accurately captures the characteristics of both present and future workloads and accounts for reliability

Perf/TCO is no longer sufficient

Change in assumptions driving need to evolve metrics

We cannot assume power capacity is infinite

- Power bound
- Location bound
- Environmental considerations

We need metrics to match community's commitment to the carbon-free future

Google has publicly committed to operate 24/7 carbon-free energy by 2030

We need metrics to match Google's commitment to the carbon-free future

Google has publicly committed to operate 24/7 carbon-free energy by 2030

1. **Systems Perf/Average Watt**
   A metric that represents systems performance capacity with fixed power capacity

2. **Systems Perf/CO2e***
   CO_2e = DC construction CO_2e +
   Compute Infra build/delivery CO_2e +
   Compute Infra Operational CO_2e

   We must account for the cost of building, shipping, and deploying our infrastructure.

*CO_2e = Carbon Dioxide Equivalents

Sample server CO_2e

Server build CO_2e: 1-4t/server *according to public sources*

CO_2e offsets: \$1000/ton *according to public sources*

1000W server with 50% average utilization of TDP:

500W → 4380kWh/year * 6 year lifetime = **26000kWh** → **12.5 metric tons of CO_2 per IEA**
using average 2019 power emissions [1]
475 gCO_2e*/kWh*

[1] International Energy Agency, Global Energy and CO_2 Status Report, 2019.

*CO_2e = Carbon Dioxide Equivalents

Meeting the demand requires innovation

Rethinking our system and infrastructure designs

Fleet infrastructure and optimizing deployment strategies → system TDP

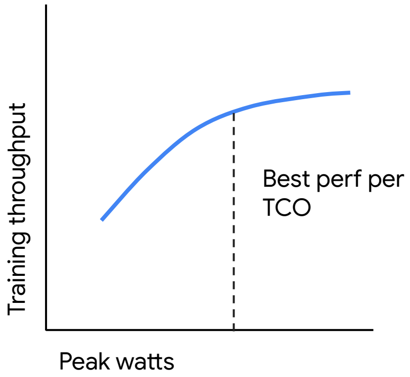
Optimizing software/hardware to manage dynamic power consumption range for average power

Optimizing software/hardware to manage dynamic power consumption range for average power

Get more out of each Watt

Optimize the power parameters
for each job

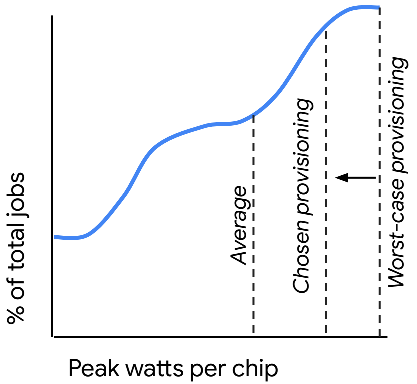
ML job



Provision the right amount of Watts

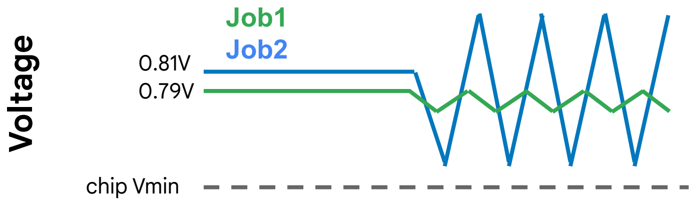
Load balance jobs to avoid worst-case,
concentrated power peaks

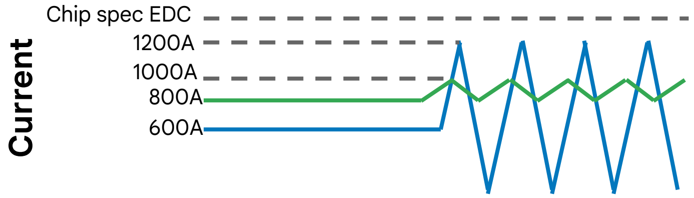
Power demand CDF



Before optimization

	TDC	Vmin margin	WL bound by	Core freq	Vset	Load line
Job1	800A	80mV	compute	1x	0.87V	0.1m
Job2	600A	10mV	memory	1x	0.87V	0.1m



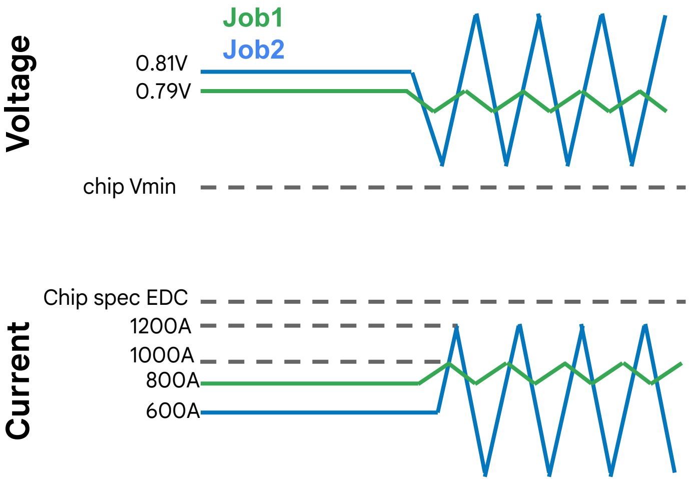


Before optimization

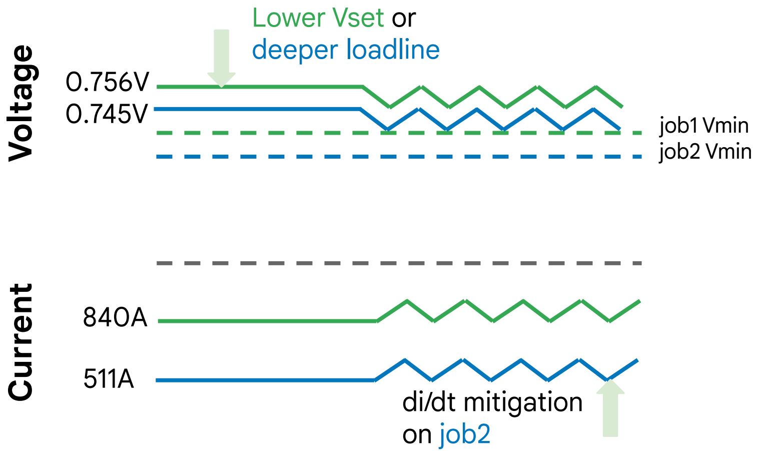
	TDC	Vmin margin	WL bound by	Core freq	Vset	Load line
Job1	800A	80mV	compute	1x	0.87V	0.1m
Job2	600A	10mV	memory	1x	0.87V	0.1m

After optimization

	TDC	Vmin margin	WL bound by	Core freq	Vset	Load line
Job1	840A	25mV	balanced	**1.1x**	**0.84V**	0.1m
Job2	500A	25mV	balanced	**0.9x**	0.87V	**0.25m**



Workload-aware optimization

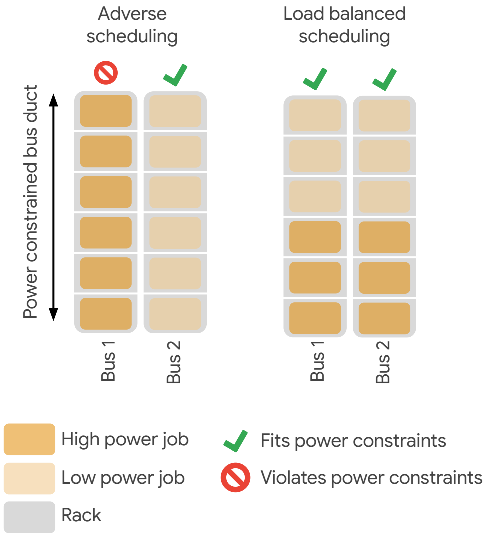


1.1x Performance
0.8x Power

Optimizing software/hardware to manage dynamic power consumption range for average power

Power aware scheduling

Use power demand & supply in job scheduling

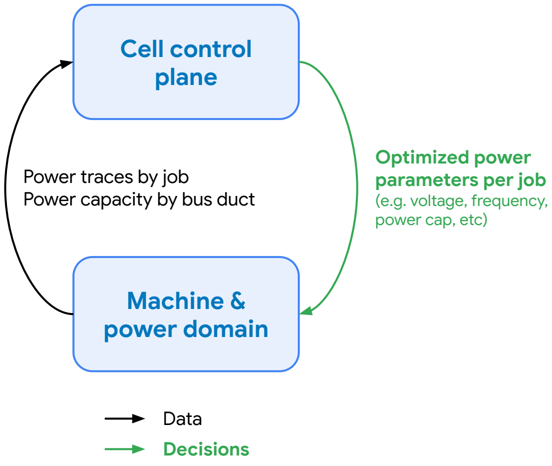


Job power parameter fine tuning

Add live feedback loops between power domain and cell control plane

Given job placements, optimize each job’s power params

Given the power profile of jobs, (re)optimize their placement



Power traces by job
Power capacity by bus duct

Optimized power parameters per job
(e.g. voltage, frequency, power cap, etc)

The control plane combines both capabilities to **maximize cell throughput per unit of power**, while **respecting job SLOs** and reacting to power domain failure events

End to end power/perf management

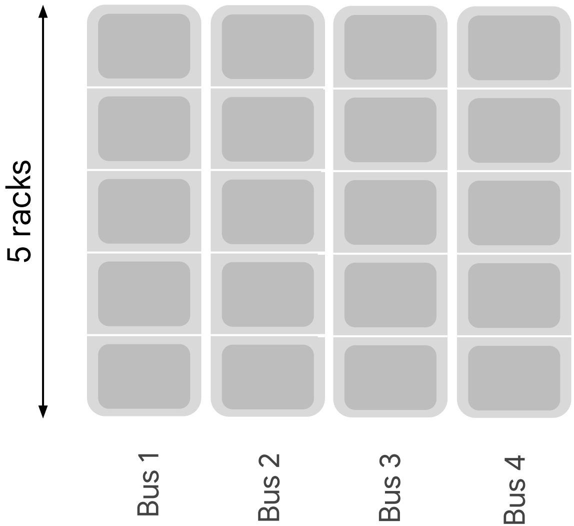
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



End to end power/perf management

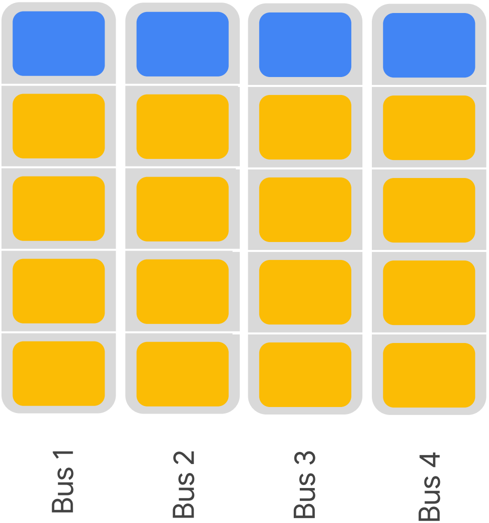
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Initial placement load balances power demand across all bus ducts

End to end power/perf management

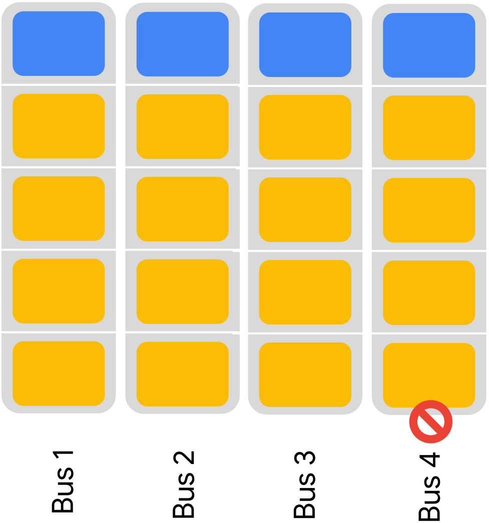
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Power failure event: Bus duct 4 has 40% less power available

End to end power/perf management

Schedule:

- 4x 1-rack training jobs
- 1x 16-rack training job

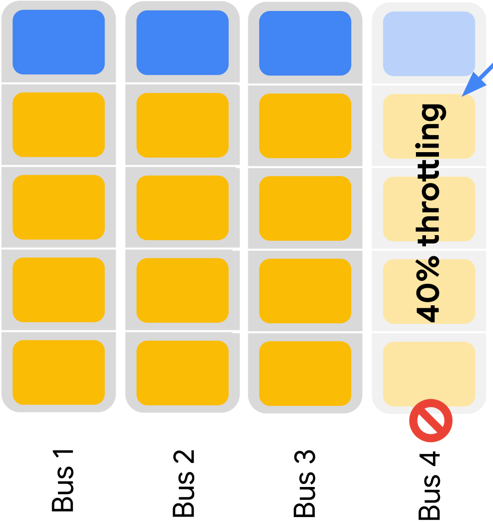
Mini datacenter:

- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 1: Throttle all jobs in bus duct 4 by 40%

Power failure event: Bus duct 4 has 40% less power available

End to end power/perf management

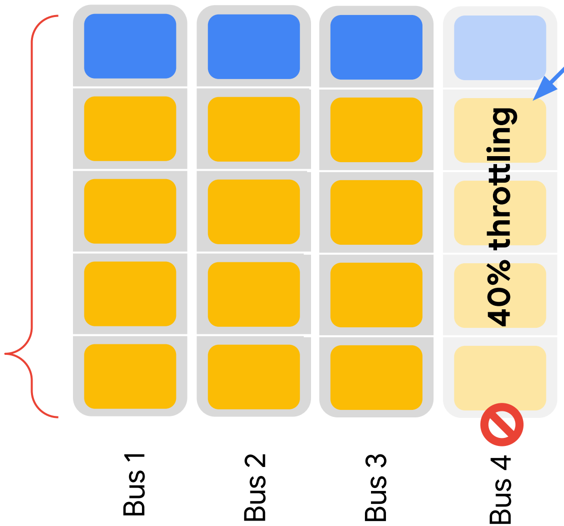
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 1: Throttle all jobs in bus duct 4 by 40%

Power failure event: Bus duct 4 has 40% less power available

Throttling affects throughput of entire large job, severely impacting cell-wide throughput

End to end power/perf management

Schedule:

- 4x 1-rack training jobs
- 1x 16-rack training job

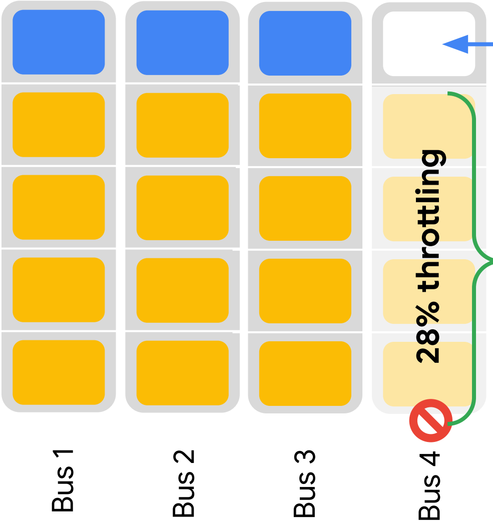
Mini datacenter:

- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 2: Evict the small job in bus 4 & give more power budget to the Large job

Effect: Throttling is reduced from 40% to 28%

Power failure event: Bus duct 4 has 40% less power available

End to end power/perf management

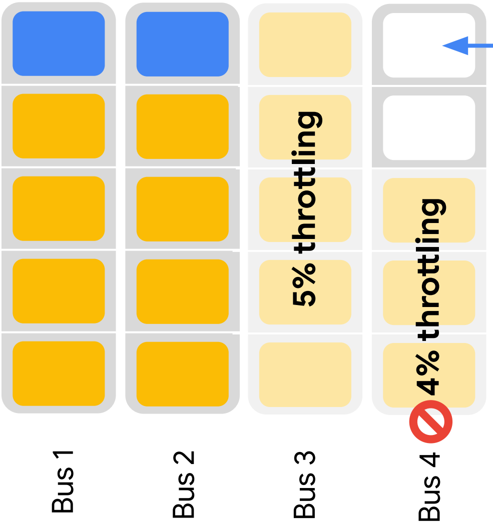
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 3: Shift workers from the large job to bus 3 (evicting a small job along the way). Throttle Bus 3 to avoid violating power constraint

Effect: Throttling is reduced from 28% to 5%. This is the minimum possible throttling for the cell.

Power failure event: Bus duct 4 has 40% less power available

End to end power/perf management

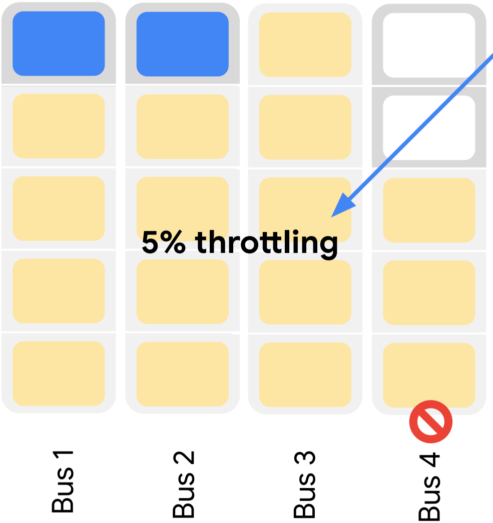
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 4: Throttle the entire large job by 5% to even out performance

Power failure event: Bus duct 4 has 40% less power available

End to end power/perf management

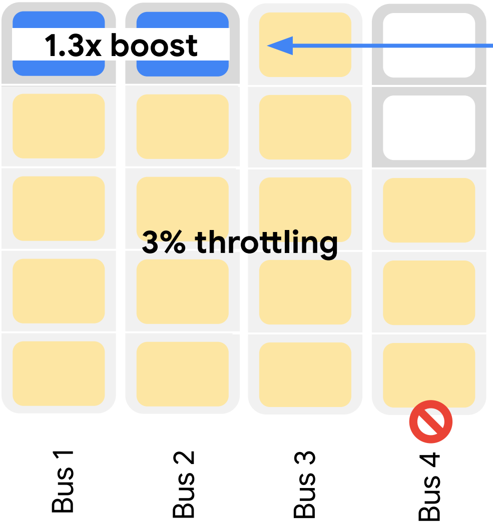
Schedule:
- 4x 1-rack training jobs
- 1x 16-rack training job

Mini datacenter:
- 4 bus ducts
- 20 racks (5 racks per bus duct)

⬜ Rack

🟦 Small, 1 rack synchronous training job
(Up to 16% peak bus duct power per rack)

🟨 Large, 16 rack synchronous training job
(Up to 21% peak bus duct power per rack)



Action 5: Use the incremental power budget to boost the remaining jobs

Effect: Maximized throughput for the cell

Power failure event: Bus duct 4 has 40% less power available

Given the scale of ML systems and the size of ML training jobs any reliability issues become massively impactful, often times infecting the entire system

Silent data corruption

- Non-deterministically produce incorrect results, silently
- Challenging problem when running largely independent computation
- Multiplicatively worse at scale with synchronous stochastic gradient descent
- Can quickly spread results across thousands of components across ML supercomputer

Cores that don't count

Peter H. Hochschild
Paul Turner
Jeffrey C. Mogul
Google
Sunnyvale, CA, US

Rama Govindaraju
Parthasarathy
Ranganathan
Google
Sunnyvale, CA, US

David E. Culler
Amin Vahdat
Google
Sunnyvale, CA, US

Abstract

We are accustomed to thinking of computers as fail-stop, especially the cores that execute instructions, and most system software implicitly relies on that assumption. During most of the VLSI era, processors that passed manufacturing tests and were operated within specifications have insulated us from this fiction. As fabrication pushes towards smaller feature sizes and more elaborate computational structures, and as increasingly specialized instruction-silicon pairings are introduced to improve performance, we have observed ephemeral computational errors that were not detected during manufacturing tests. These defects cannot always be mitigated by techniques such as microcode updates, and may be correlated to specific components within the processor, allowing small code changes to effect large shifts in reliability. Worse, these failures are often "silent" – the only symptom is an erroneous computation.

We refer to a core that develops such behavior as "mercurial." Mercurial cores are extremely rare, but in a large fleet of servers we can observe the disruption they cause, often enough to see them as a distinct problem – one that will require collaboration between hardware designers, processor vendors, and systems software architects.

This paper is a call-to-action for a new focus in systems research; we speculate about several software-based approaches to mercurial cores, ranging from better detection and isolating mechanisms, to methods for tolerating the silent data corruption they cause.

ACM Reference Format:
Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Workshop on Hot Topics in Operating Systems (HotOS '21), May 31–June 2, 2021, Ann Arbor, MI, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3458336.3465297

1 Introduction

Imagine you are running a massive-scale data-analysis pipeline in production, and one day it starts to give you wrong answers – somewhere in the pipeline, a class of computations are yielding corrupt results. Investigation fingers a surprising cause: an innocuous change to a low-level library. The change itself was correct, but it caused servers to make heavier use of otherwise rarely-used instructions. Moreover, only a small subset of the server machines are repeatedly responsible for the errors.

This happened to us at Google. Deeper investigation revealed that these instructions malfunctioned due to manufacturing defects, in a way that could only be detected by checking the results of these instructions against the expected results; these are "silent" *corrupt execution errors*, or CEEs. Wider investigation found multiple different kinds of CEEs; that the detected incidence is much higher than software engineers expect; that they are not just incremental increases in the background rate of hardware errors; that these can manifest long after initial installation; and that they typically afflict specific cores on multi-core CPUs, rather than the entire chip. We refer to these cores as "mercurial."

Because CEEs may be correlated with specific execution units within a core, they expose us to large risks appearing suddenly and unpredictably for several reasons, including seemingly-minor software changes. Hyperscalers have a responsibility to customers to protect them against such risks. For business reasons, we are unable to reveal exact CEE rates, but we observe on the order of a few mercurial cores per several thousand machines – similar to the rate reported by Facebook [8]. The problem is serious enough for us to have applied many engineer-decades to it.

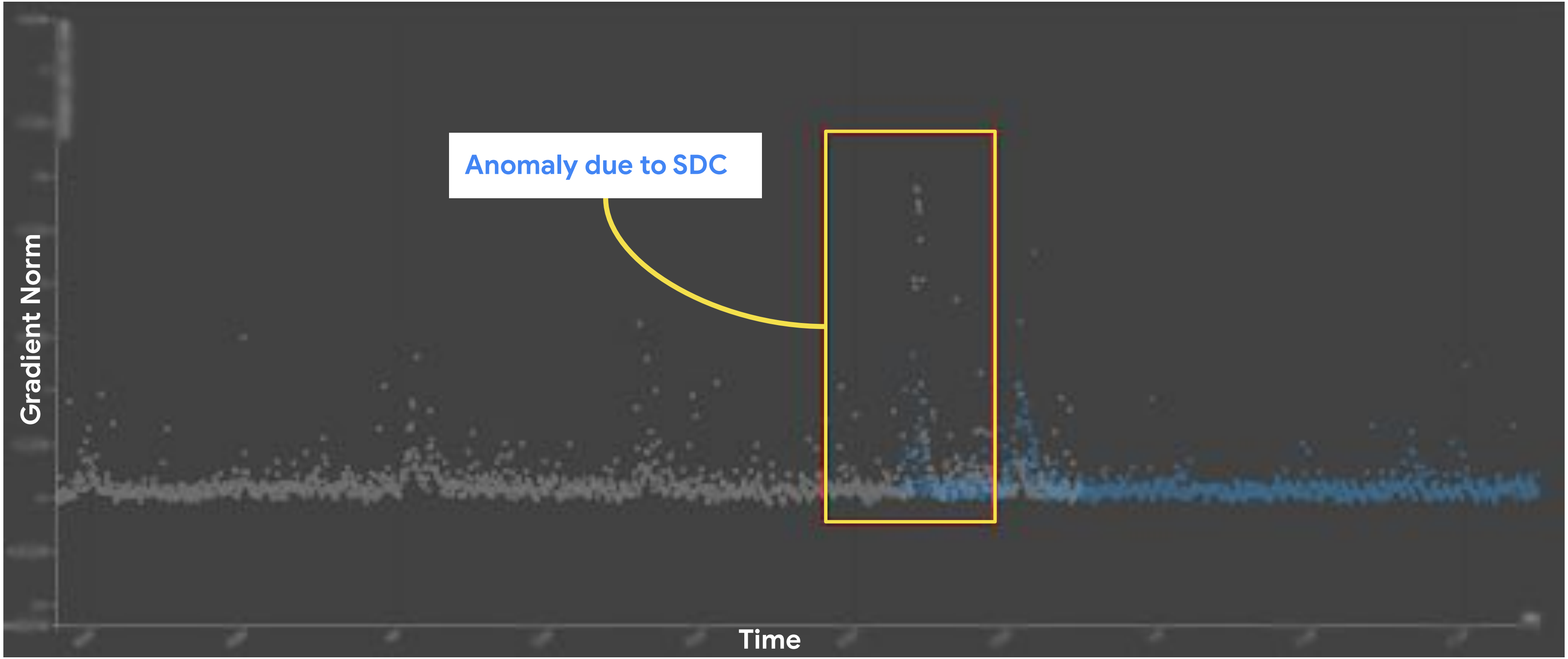
While we have long known that storage devices and networks can corrupt data at rest or in transit, we are accustomed to thinking of processors as fail-stop. VLSI has always depended on sophisticated manufacturing testing to detect defective chips. When defects escaped, or manifested with aging, they were assumed to become fail-stop or at least fail-noisy: triggering machine-checks or giving wrong answers for many kinds of instructions. When truly silent failures occurred, they

---

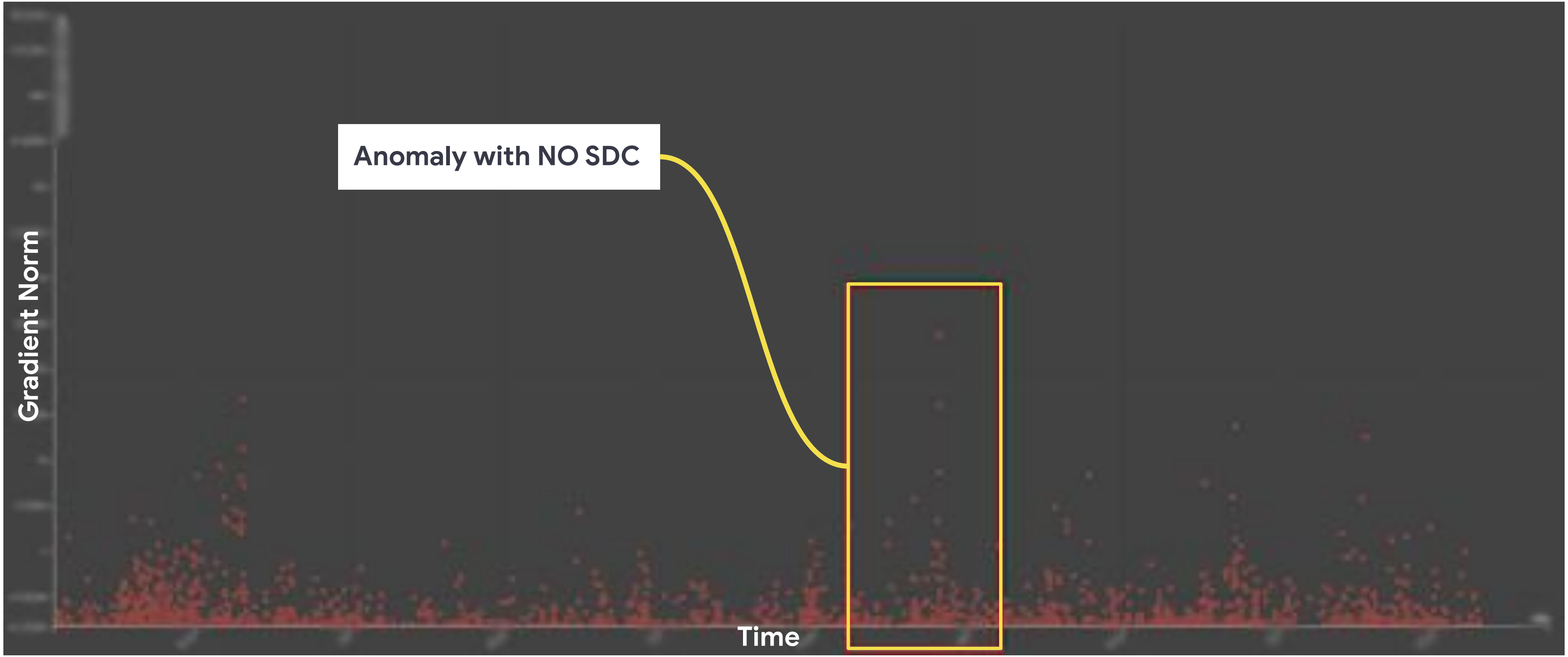
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

HotOS '21, May 31–June 2, 2021, Ann Arbor, MI, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8438-4/21/05.
https://doi.org/10.1145/3458336.3465297

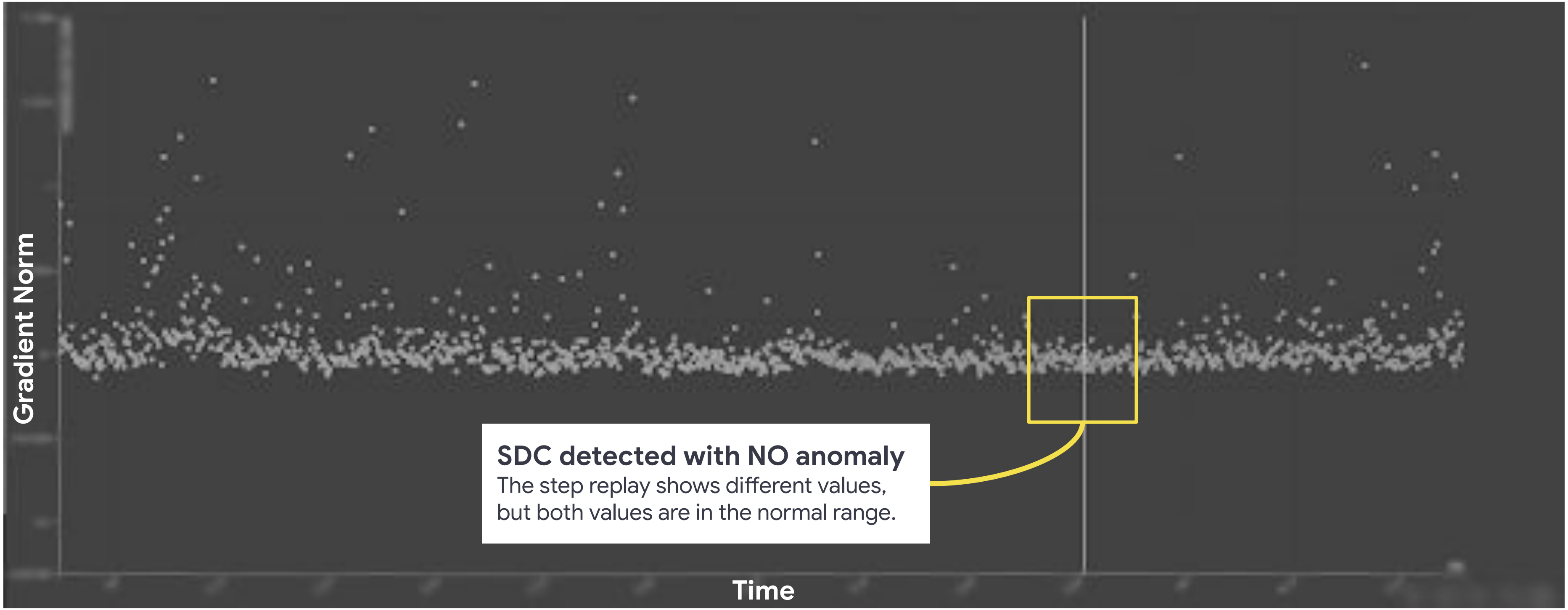
Metrics anomaly: anomaly due to SDC



Metrics anomaly: expected anomaly (no SDC)



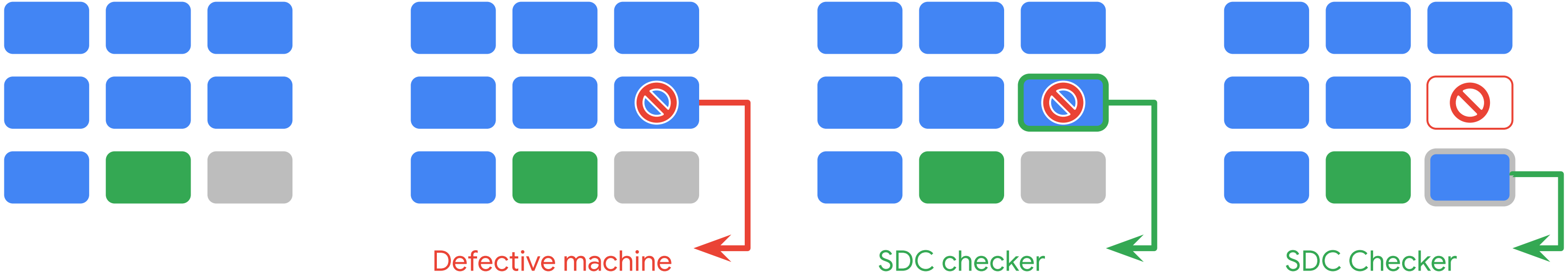
SDC with no metrics anomaly



SDC detected with NO anomaly
The step replay shows different values, but both values are in the normal range.

ML Controller transparently handles Silent Data Corruption (SDC)

🟦 Synchronous training worker 🟩 SDC checker ⬜ Hot spare



Normal training state

Defective machine causes SDC

SDC checker automatically identifies SDC

SDC Checker moves training to hot spare and sends defective machine for repair

Iterate much faster when delivering specialized hardware

We have to iterate much faster...

Current *idealized* timeline for chip delivery to production

6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

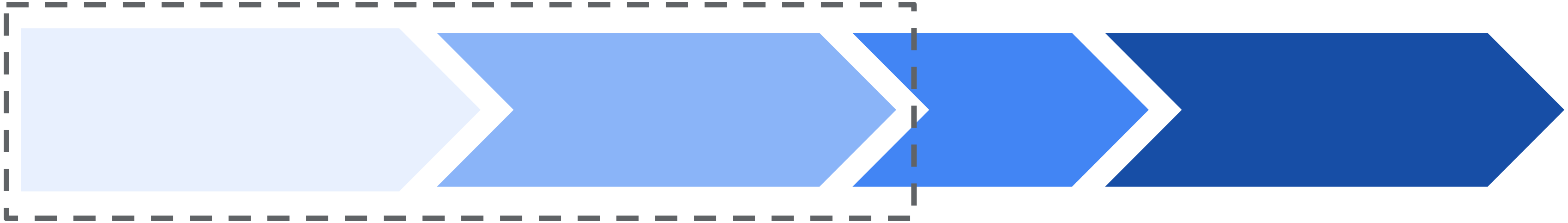
12 months

Chip to production

Right now, industry best practices from idea to production:

~3 years

Opportunity: Apply ML to chip design



6-12 months

Design and
exploration

12 months

Implement the
new design

6 months

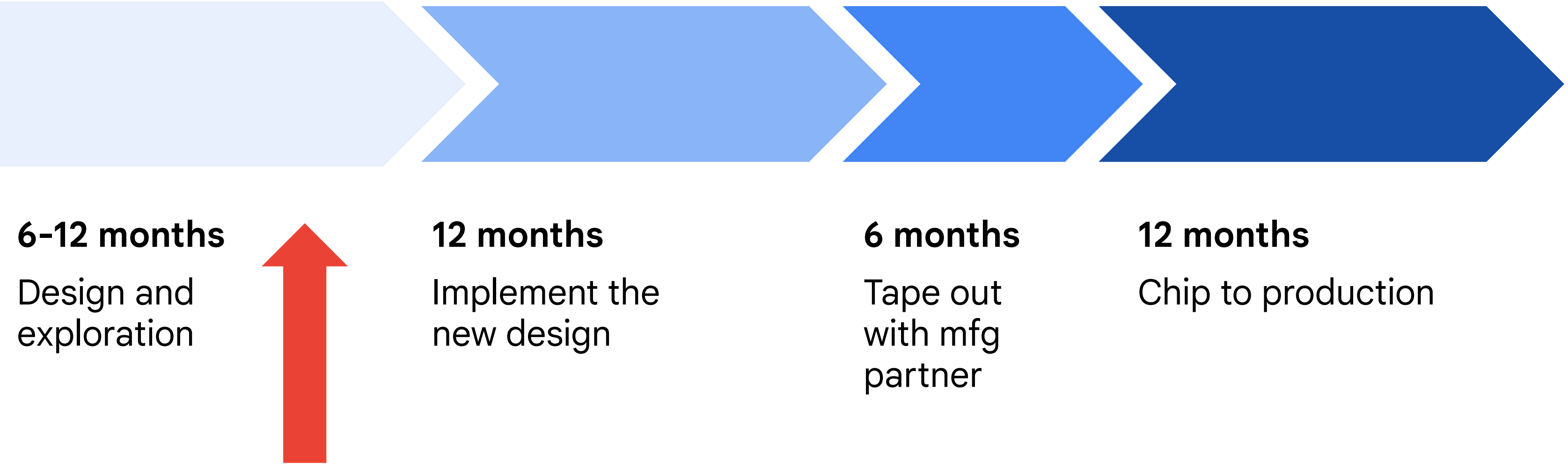
Tape out
with mfg
partner

12 months

Chip to production

What if designing a custom chip took a few people a few weeks?

Opportunity: Apply ML to chip design



6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

12 months

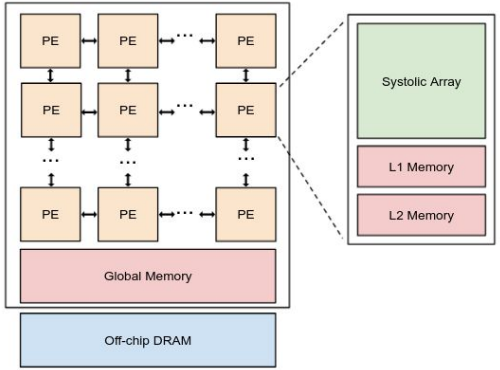
Chip to production

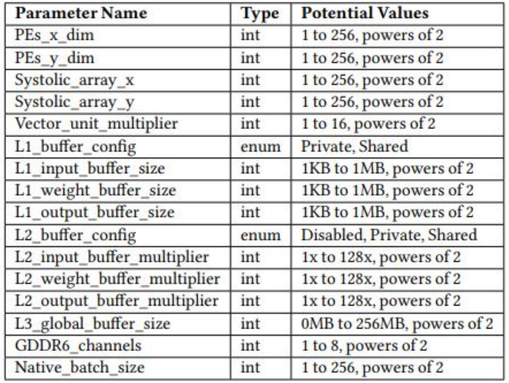
Use machine learning to automatically do architectural exploration and synthesis

Using ML to extend the design space exploration

Simultaneously optimize

- Hardware design space choices (~10^{13} search space)
- How workloads are mapped onto this hypothetical hardware (by compilers or other software): search space now much larger (~10^{2300})



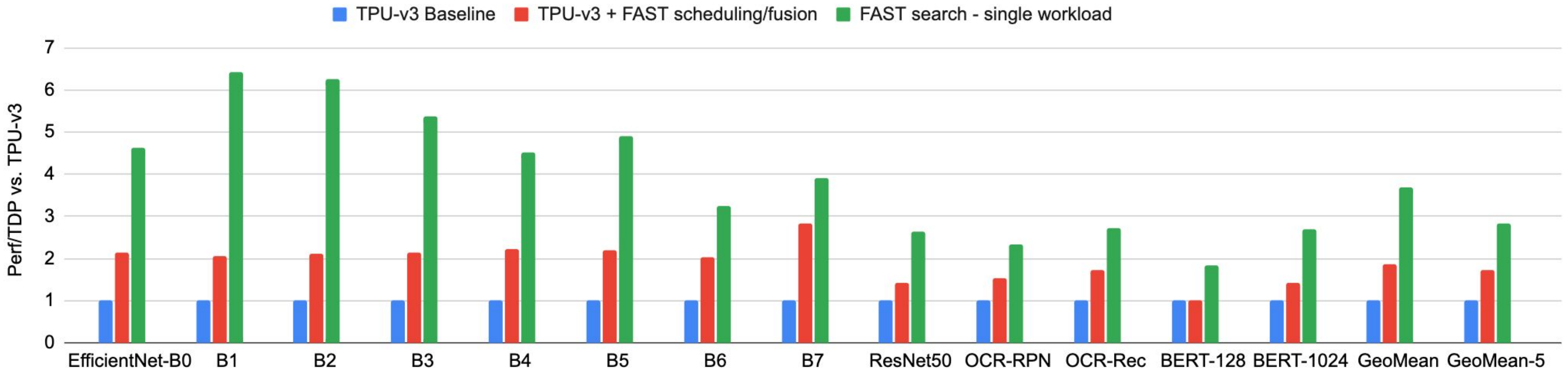


Parameter Name	Type	Potential Values
PEs_x_dim	int	1 to 256, powers of 2
PEs_y_dim	int	1 to 256, powers of 2
Systolic_array_x	int	1 to 256, powers of 2
Systolic_array_y	int	1 to 256, powers of 2
Vector_unit_multiplier	int	1 to 16, powers of 2
L1_buffer_config	enum	Private, Shared
L1_input_buffer_size	int	1KB to 1MB, powers of 2
L1_weight_buffer_size	int	1KB to 1MB, powers of 2
L1_output_buffer_size	int	1KB to 1MB, powers of 2
L2_buffer_config	enum	Disabled, Private, Shared
L2_input_buffer_multiplier	int	1x to 128x, powers of 2
L2_weight_buffer_multiplier	int	1x to 128x, powers of 2
L2_output_buffer_multiplier	int	1x to 128x, powers of 2
L3_global_buffer_size	int	0MB to 256MB, powers of 2
GDDR6_channels	int	1 to 8, powers of 2
Native_batch_size	int	1 to 256, powers of 2

A Full-stack Search Technique for Domain Optimized Deep Learning Accelerators,
Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. ASPLOS, 2022, arxiv.org/abs/2105.12842

Using ML to extend the design space exploration

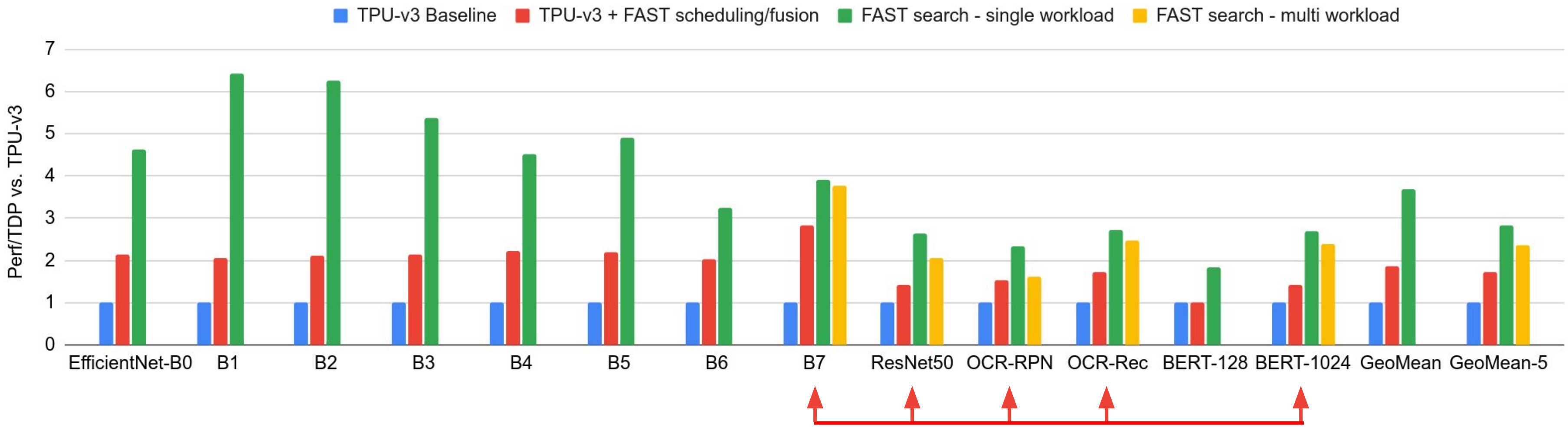
Blue: baseline TPUv3-like system but simulated on more modern process
Red: compiler space exploration only with no HW changes vs. baseline
Green: customizing accelerator+compiler for a particular single model



A Full-stack Search Technique for Domain Optimized Deep Learning Accelerators,
Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. ASPLOS, 2022, arxiv.org/abs/2105.12842

Using ML to extend the design space exploration

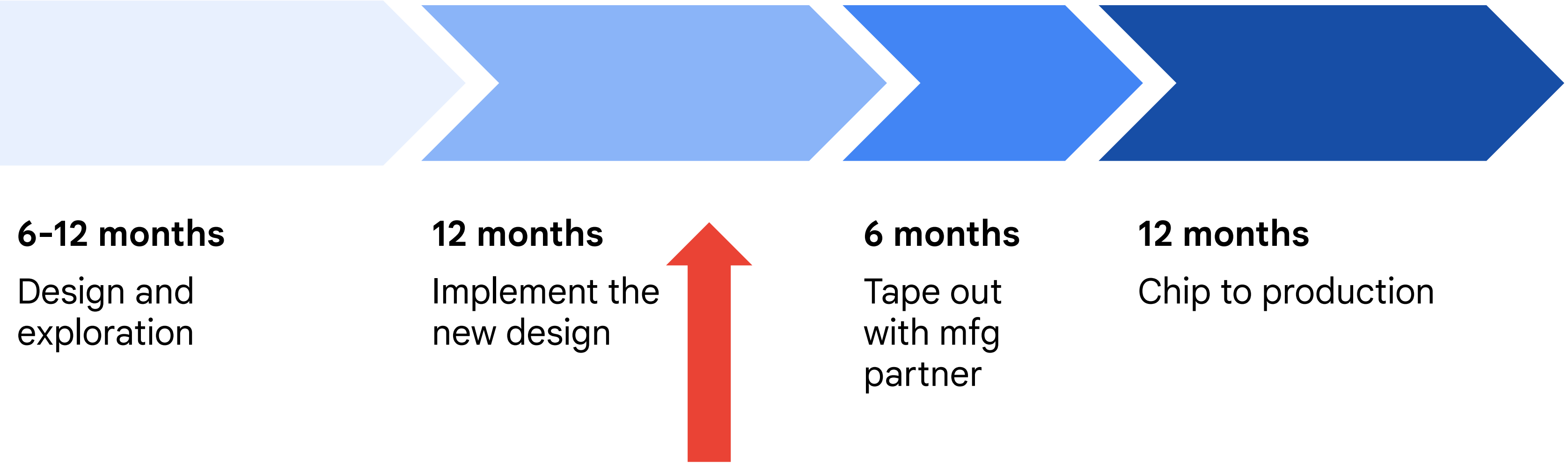
Blue: baseline TPUv3-like system but simulated on more modern process
Red: compiler space exploration only with no HW changes vs. baseline
Green: customizing accelerator+compiler for a particular single model
Yellow: customizing accelerator+compiler to mix of five model workload



Customize for these five

A Full-stack Search Technique for Domain Optimized Deep Learning Accelerators,
Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. ASPLOS, 2022, arxiv.org/abs/2105.12842

Opportunity: Apply ML to chip design



6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

12 months

Chip to production

Speed up verification by learning to automatically generate test coverage with small set of tests

Opportunity: Apply ML to chip design

6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

12 months

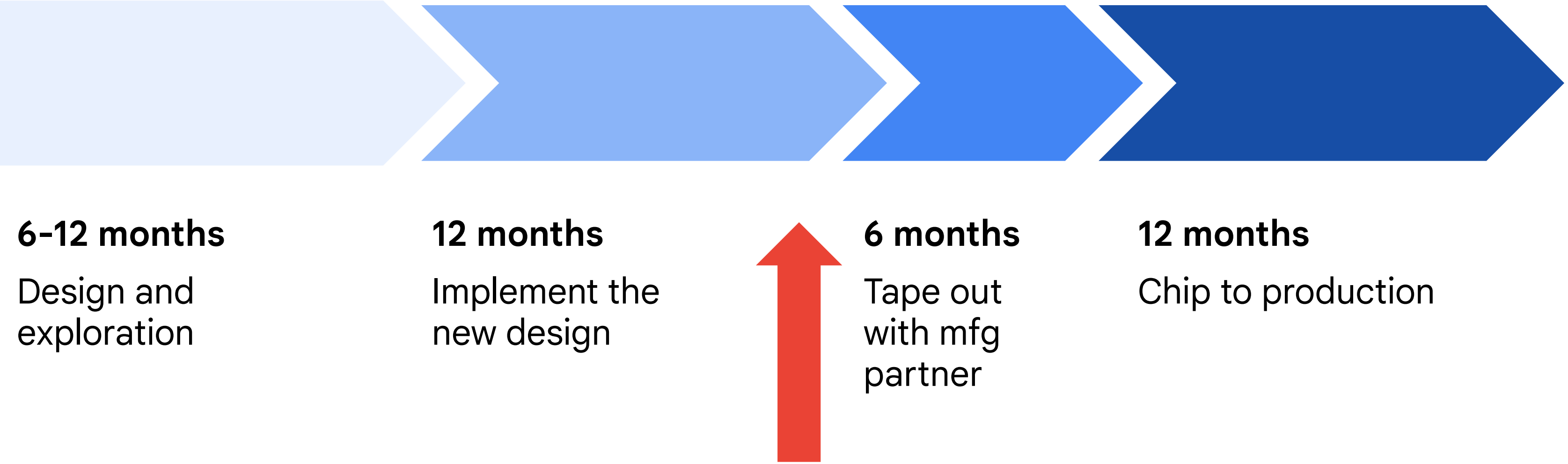
Chip to production

Speed up verification by learning to automatically generate test coverage with small set of tests

Promising results not discussed in this talk. See paper below!

Learning Semantic Representations to Verify Hardware Designs, Shobha Vasudevan, Wenjie Jiang, David Bieber, Rishabh Singh, Hamid Shojaei, C. Richard Ho, Charles Sutton. NeurIPS 2021, openreview.net/pdf?id=oIhzg4GJeOf

Opportunity: Apply ML to chip design



6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

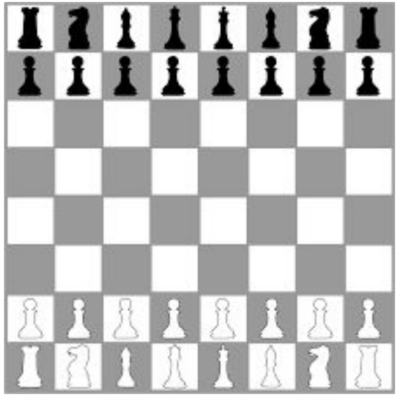
12 months

Chip to production

Learn to quickly generate high quality placement and routing solutions

Can we get an RL Agent to successfully play the “Game of ASIC Chip Layout”?

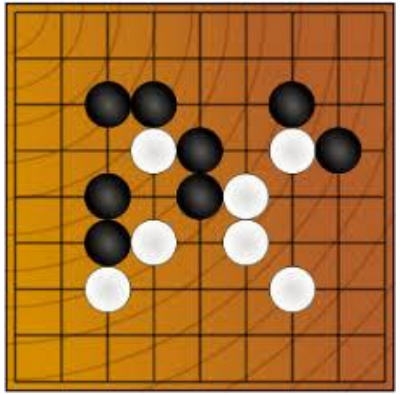
Chess



Number of states

~10^{123}

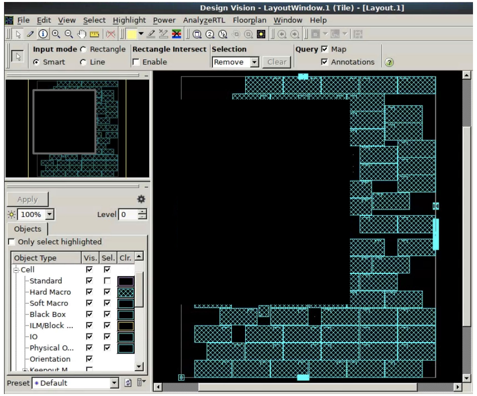
Go



Number of states

~10^{360}

Chip Placement


Design Vision - LayoutWindow.1 (Tile) - [Layout.1]
File Edit View Select Highlight Power AnalyzeRTL Floorplan Window Help
Apply
Objects
Preset
Remove
Clear

Input mode ○ Rectangle ● Smart ○ Line

Rectangle Intersect ☐ Enable

Selection Remove

Query ☑ Map ☑ Annotations

100% Level 0

☐ Only select highlighted

Object Type	Vis.	Sel.	Clr.
Cell	☑	☑	
Standard	☑	☐	
Hard Macro	☑	☑	
Soft Macro	☑	☑	
Black Box	☑	☑	
ILM/Block ...	☑	☑	
IO	☑	☑	
Physical O...	☑	☑	
Orientation	☑		
Keepout M...	☐		

Preset Default

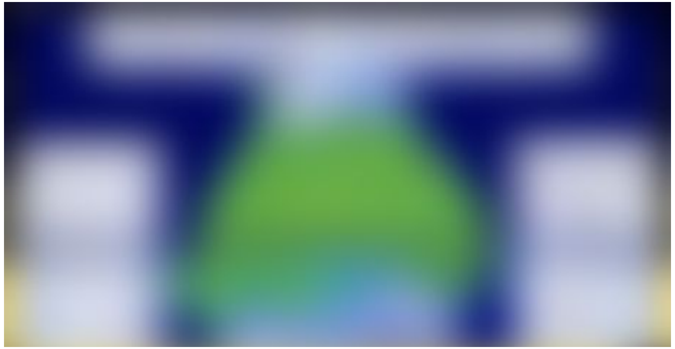
Number of states

~10^{9000}

Results on a TPU design block

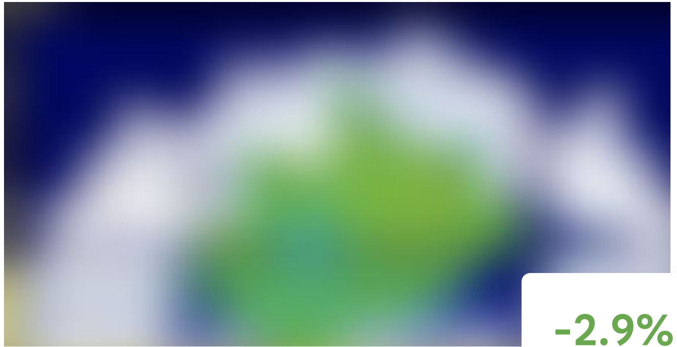
White blurred area are macros (memory); green blurred area are standard cell clusters (logic)
ML placer finds smoother, rounder macro placements to reduce the wirelength

Human Expert



Time taken: **~6-8 person weeks**
Total wirelength: 57.07m

ML Placer



-2.9% shorter

Time taken: **24 hours**
Total wirelength: 55.42m

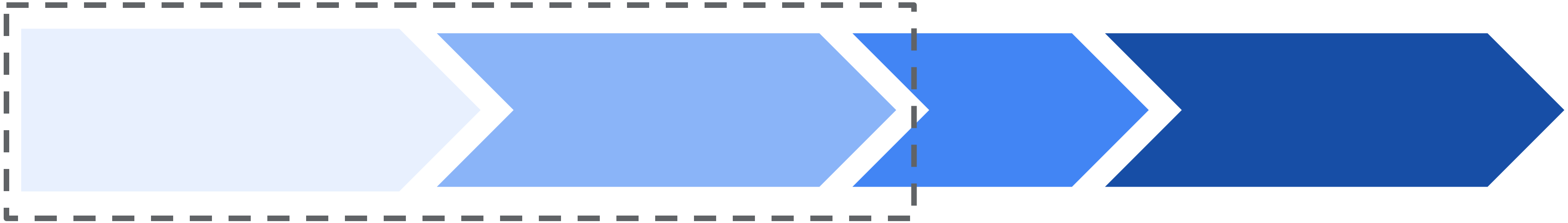
Results on recent full chip TPU design

RL tool used for placement and routing of 37 blocks of recent TPU chip design

Compared to placements by human experts:

- **26 of 37** blocks better quality of result
- **7 of 37** blocks equal quality of result
- **4 of 37** blocks worse quality of result

Opportunity: Apply ML to chip design



6-12 months

Design and exploration

12 months

Implement the new design

6 months

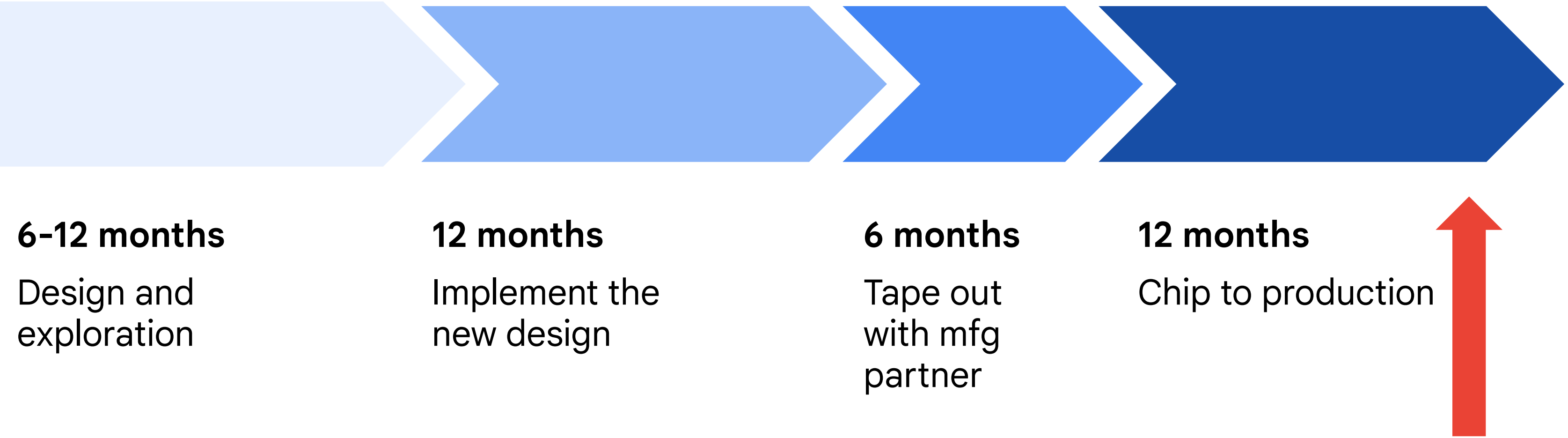
Tape out with mfg partner

12 months

Chip to production

Note that even if we just get much more rapid turnaround with less human effort required for the whole design and implementation phase of a chip, we can then run it on system emulators, get much higher quality feedback, etc., even if we don't send every automated design+implementation to a fab

Opportunity: Apply ML to chip design



6-12 months

Design and exploration

12 months

Implement the new design

6 months

Tape out with mfg partner

12 months

Chip to production

Software running on emulation at tapeout
Assuming silicon is correct, converge quickly to target DPPM
Running workloads across thousands of chips in one month
Vendor IP with debugging and visibility support, SDC isolated with OOB test

Conclusions

- → ML capabilities (image creation, audio, coding assistance, etc) improving rapidly and will bring fundamental changes to the way we do things
- → ML is an increasingly large portion of global computation
- → ML models increasingly dynamic and evolving structures, not static, dense models
- → Focus on systems goodput not chip headline performance
- → Power, CO_2e efficiency, SDC important to accurately measure and improve
- → Shorter timelines for designing and deploying new hardware essential to rapidly adapt to changing ML landscape (ML automation of design process can help!)

Thank you!

Google