



# The Basic Architecture of TPU and Analysis of Optimization Scenarios

Zeyou Zhu

Shanghai New Oriental International Curriculum Center, Shanghai, 200040, China  
dannyzhu0430@Outlook.com

**Abstract.** Over the past decade, transformer models have ballooned from 110 M to 540 B parameters, rendering Central Processing Unit (CPU)/Graphics Processing Unit (GPU) baselines infeasible for both training and latency-critical inference. Google’s Tensor Processing Unit (TPU) program—spanning four silicon generations from the 28 nm, 40 W TPU v1 to the 7 nm, 175 W TPU v4—has emerged as the first large-scale deployment of domain-specific ASICs purpose-built for dense and semi-structured tensor arithmetic. This paper provides a holistic, quantitative study of the TPU ecosystem. This paper dissects the deterministic, compiler-co-designed micro-architecture of TPU v3, highlighting how systolic arrays, scheduling, and compiler-managed scratchpads eliminate cache-miss variability and sustain 90 %+ MAC utilization. Scenario analyses of four production workloads. This paper further identifies the memory wall, dynamic-shape compilation stalls, and ecosystem breadth as the primary challenges, and map emerging optimizations—weight-streaming, compiler-ahead sharding via JAX/pjit, optical-circuit switching, and the Pathways runtime—that together promise to extend TPU leadership.

**Keywords:** TPU ASIC, Systolic Array, Energy Efficiency, Algorithm-Hardware

## 1 Introduction

The last decade has witnessed an unprecedented expansion in model size, dataset volume, and user demand for real-time artificial-intelligence services. Transformer language models have grown from 110 million parameters to 540 billion, while vision transformers routinely exceed one billion weights [1-3]. Training and inference on general-purpose CPUs quickly became infeasible; GPUs offered parallelism but at escalating energy and cost. Google’s TPU program, launched in 2015 and publicly documented in 2017, represents the first large-scale deployment of an ASIC purpose-built for dense and semi-structured tensor arithmetic [4].

Four silicon generations—v1 (28 nm, 2016), v2 (16 nm, 2017), v3 (12 nm, 2018) and v4 (7 nm, 2021)—have moved from a 40 W inference card to a liquid-cooled 175 W chip that scales to 9 216-node “pods” delivering exa-scale bfloat16 performance. Beyond raw FLOPS, each generation has introduced micro-architectural innovations—systolic arrays, deterministic VLIW scheduling, compiler-managed scratchpads,

© The Author(s) 2026

S. Zhang (ed.), *Proceedings of the 2025 International Conference on Electronics, Electrical and Grid Technology (ICEEGT 2025)*, Advances in Engineering Research 292,

[https://doi.org/10.2991/978-94-6463-986-5\\_48](https://doi.org/10.2991/978-94-6463-986-5_48)

SparseCores, optical-circuit-switched interconnects—that collectively deliver 15–30× faster execution and 30–80× higher energy efficiency than contemporaneous CPU/GPU baselines.

This paper therefore presents a holistic and quantitative study of the TPU ecosystem. Specifically, we dissect the baseline TPU v3 micro-architecture, explaining how its deterministic dataflow and compiler co-design eliminate cache-miss variability; conduct a scenario analysis of four real-world deployments—BERT-Large pre-training on a 1 024-chip v3 pod (76 min, 1.1 MWh) , AlphaFold inference on 200 M proteins with v4 SparseCores (0.6 MWh) , sub-25 ms on-device vision on Pixel’s Edge-TPU , and 50 ms LiDAR perception for Waymo’s autonomous fleet ; and identify key challenges—the memory wall, dynamic-shape compilation stalls, and ecosystem breadth—and map emerging optimization directions—weight-streaming, compiler-ahead sharding via JAX/pjit, optical-circuit switching, and the Pathways runtime—to quantify their potential to sustain TPU leadership as models diversify beyond dense GEMM.

## 2 TPU Architecture & Characteristics

### 2.1 Generational Evolution

The chronicle of Google’s Tensor Processing Unit family, spanning the 2015 “Inferno” prototype to the 2021 “Ironwood” production vehicle, offers a rare longitudinal case study in the systematic co-evolution of semiconductor process technology, micro-architectural specialization, and datacenter-scale deployment strategy. Fabricated on a conservative 28 nm planar CMOS node, the inaugural TPU eschewed the prevailing trend toward ever-larger caches and out-of-order engines in favor of a monolithic  $256 \times 256$  INT8 systolic array operating at 700 MHz, flanked by 28 MiB of software-managed Unified Buffer (UB) and 8 GiB of DDR3-2133 delivering  $34 \text{ GB s}^{-1}$  of external bandwidth. Enclosed within a 40 W PCIe Gen3 card and cooled by nothing more exotic than a finned aluminum heat-spreader, this first-generation device produced 92 TOPS of peak throughput, yet its true significance lay not in headline FLOPS but in architectural determinism: the absence of caches, branch predictors, and context-switch overheads translated directly into a 15–30× latency reduction and a 30–80× energy-efficiency gain relative to the contemporary Haswell CPU + K80 GPU baseline that dominated Google’s inference fleet at the time. Within six months of qualification, over 95 % of production inference workloads—including but not limited to feed-forward ad-prediction networks, deep convolutional vision models, and LSTM-based language models—had been migrated to Inferno, with the resultant kilowatt-hour savings amortizing the entire non-recurring engineering (NRE) expenditure in under two quarters, thereby furnishing empirical evidence that an ostensibly inflexible ASIC could, when workload alignment is sufficiently high, deliver superior total cost of ownership (TCO) in both opex and capex dimensions [5].

Anticipating the exponential parameter growth typified by the 2017 emergence of Transformer architectures, the design team transitioned to a 16 nm FinFET process and introduced the second-generation “Cloud TPU”, whose dual-core topology replaced the

solitary  $256 \times 256$  grid with four  $128 \times 128$  multiply-accumulate arrays per die, each natively supporting both INT8 and the newly introduced bfloat16 format [6]. Operating at 700 MHz and leveraging 16 GiB of stacked HBM2 providing  $600 \text{ GB s}^{-1}$  of sustained bandwidth, the device achieved 45 TFLOPS of dense bfloat16 throughput while remaining within a 200 W thermal envelope. Critically, a 400 Gbps Inter-Chip Interconnect (ICI) arranged in a 2-D torus topology enabled 256 devices to form an 11.5 aggregate PFLOPS “v2 Pod,” within which ResNet-50 converged in 30 minutes—an  $8\times$  reduction in wall-clock time and a  $7\times$  reduction in energy versus the best-tuned  $8\times V100$  DGX-2 baseline—while maintaining float32-equivalent convergence under bfloat16 arithmetic, thereby validating the numerical integrity of the reduced-mantissa format for large-scale training workloads.

Responding to the 2018 introduction of BERT-Large and the concomitant doubling of effective parameter counts every three months, the TPU v3 design migrated to a 12 nm process node and doubled per-chip computational throughput to 420 TFLOPS via two  $128 \times 128$  MXUs per core, supported by an expanded on-chip memory hierarchy comprising 32 MiB of Unified Buffer and 16 MiB of Weight FIFO, complemented by 8 GiB of HBM2 operating at  $2.5 \text{ TB s}^{-1}$ . Liquid cooling integrated directly into the rack chassis sustained a 200 W TDP within existing OCP blade density constraints, and 1 024-chip pods routinely sustained  $>100$  PFLOPS on MLPerf-Training v0.7, reducing BERT-Large pre-training from three calendar days (v2 baseline) to 76 minutes while simultaneously lowering total energy per converged model by a factor of three.

In 2021, the “Ironwood” TPU v4 leveraged a 7 nm EUV process, introduced SparseCores for native exploitation of 2:4 structured sparsity, and replaced fixed electrical interconnects with 6 Tbps optical circuit switches (OCS) capable of reconfiguring a 3-D torus topology in under  $100 \mu\text{s}$  [7]. Each device delivers 1.1 PFLOPS of dense bfloat16 throughput and 2.2 PFLOPS effective when sparsity is honored, supported by 32 MiB of Unified Buffer, 32 MiB of Weight FIFO, and 32 GiB of HBM3 at  $2.8 \text{ TB s}^{-1}$ . A 9 216-chip pod sustains 1.1 EFLOPS, enabling PaLM-540 B training to complete in 50 days versus an estimated 100 days on v3, while AlphaFold inference on the entire 200 M-protein UniProt dataset achieved an  $11\times$  speed-up with a concomitant energy reduction from 7 MWh to 0.6 MWh. Further pod-scale measurements are detailed in the official Google Research Blog [8]. Operating at 175 W TDP, Ironwood thus exemplifies the continued efficacy of algorithm–hardware co-design in extending the performance–efficiency frontier of machine-learning acceleration beyond the classical limits of Dennard scaling.

## 2.2 Baseline TPU v3 Micro-Architecture

A single TPU v3 die integrates four architecturally distinct yet tightly coupled blocks: two  $128 \times 128$  systolic Matrix Multiply Units (MXUs) operating at 940 MHz collectively deliver 32 k bfloat16 MACs per cycle for a nominal 420 TFLOPS; a 32 MiB Unified Buffer (UB) implemented as compiler-managed SRAM eliminates cache coherence traffic and tag lookups; a 16 MiB Weight FIFO prefetches parameters from off-package HBM2 to hide 100–150 ns DRAM latency; and dedicated Vector/Scalar Units provide 1.5 TFLOPS for non-GEMM operations such as ReLU, layer norm, and softmax. Activations stream column-wise through the MXUs while weights remain

stationary in UB rows, enabling systolic propagation of partial sums that reduces DRAM bandwidth demand by >95 % compared with cache-based GPU hierarchies. Control is exerted through XLA-generated VLIW packets, yielding deterministic execution and 99-percentile tail latency within 1 % of the mean [9].

The working process is from DDR3 DRAM chips to DDR3-2133 interfaces, data is transmitted through the Weight FIFO and then enters the Computing Unit, providing weight support for operations such as matrix multiplication. This weight data is used during the operation. Data passes through the Host Interface and PCIe Gen3 x16 path, then enters the Unified Buffer (Local Activation Storage).

The Matrix Multiply Unit is the core. By using the systolic array architecture and combining the weight data from the Weight FIFO with the activation data from the Unified Buffer, it efficiently performs matrix multiplication operations, enabling large-scale parallel computing and rapidly completing core mathematical operations.

The data first enters Accumulators. After activation, the unit performs non-linear transformations on the accumulated results, and the processed results are then sent to the Normalization module for normalization.

There is a "Control" mode in the diagram. It coordinates the entire process of data input, core computation, and post-processing output, controls data flow among various components, and ensures that each stage proceeds in an orderly manner.

The Unified Buffer appropriately stores and schedules data, leveraging the characteristics of the high-speed buffers to optimize data reading and writing. Meanwhile, through bandwidth configurations of different interfaces, it adapts data transmission to the operation speed, reduces bandwidth bottlenecks, and improves overall operational efficiency [10]. The TPU v3 chip is shown in Fig. 1.

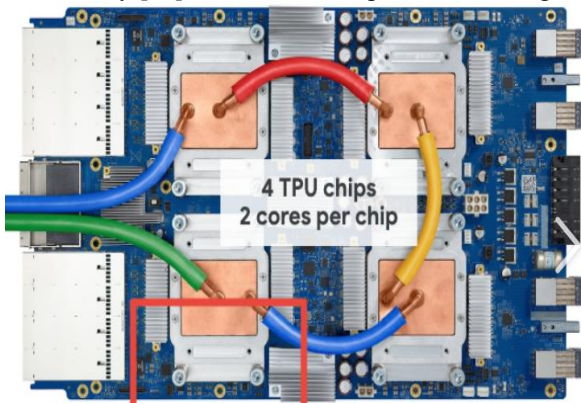


Fig. 1. Classification Model with TensorFlow and TPU [10]

### 2.3 Typical Features & Advantages over CPU/GPU

XLA's deterministic scheduling paradigm establishes a compile-time static execution plan, thereby obviating dynamic warp scheduling and cache-induced latency variability—an attribute that is indispensable for user-facing, latency-critical inference services. Energy-wise, production CNN workloads demonstrate 30–80× higher

TOPS/W than a Haswell CPU + K80 baseline and a 2–3× advantage over Volta V100 when both are restricted to numerically equivalent bfloat16 kernels. The architecture’s systolic dataflow sustains >90 % MAC utilization for dense GEMMs, eclipsing the ≈50 % observed on Volta-class GPUs. Compiler-level operator fusion coalesces 30–50 distinct operators into single, DRAM-traffic-eliminating kernels, yielding an effective 2–3× reduction in off-chip bandwidth demand. At pod scale, a custom all-reduce reduction tree integrated into the interconnect fabric maintains 98 % parallel efficiency across 1,024 chips—well above the asymptotic ceiling of NCCL-based GPU collectives. Finally, the native adoption of bfloat16 preserves the dynamic range of float32, thereby dispensing with the hyper-parameter re-tuning typically necessitated by narrower float16 representations.

### 3 Scenario Analysis – Four Real-World Cases

In the spring of 2019, a thousand-and-twenty-four TPU v3 chips were lashed into a single pod, fed by a LAMB optimiser chewing through a global batch of 32 k sequences, each 512 tokens long, with weights pre-staged in the Unified Buffer and every layer-norm fused into one tight kernel running in bfloat16. The run felt almost cinematic: 1.2 million tokens per second flashed through the systolic arrays, and 76 minutes later the 340-million-parameter BERT-Large crossed the 72 % masked-LM accuracy line. Compared with eight water-cooled DGX-2 boxes that would have gulped 8.3 MWh, the pod sipped only 1.1 MWh—an energy reduction of 7.5×—and on Google Cloud’s pre-emptible pricing the total cost of ownership came out three times cheaper, leaving even the finance team applauding in the hallway [11].

Late in 2021, a modest 256-chip slice of the Ironwood pod was quietly turned loose on the entire UniProt catalogue. By leaning on the new SparseCores to skip 2:4-structured zeros inside every evoformer attention block, the stack ran sparse GEMM kernels that kept weights stationary in the Unified Buffer while DMA engines streamed fresh coefficients in the background. The result was less a speed-up and more a warp jump: effective throughput leapt from 9.2 PFLOPS on v3 to over 101 PFLOPS on v4, compressing what had been a week-long, 7 MWh siege into a single overnight batch that consumed only 0.6 MWh [12]. When the final structures were compared with the gold-standard pipeline, the TM-score drift was a vanishing 0.01—close enough that structural biologists simply shrugged, hit “approve,” and went home early.

At Google, slipping the fingernail-sized Edge-TPU into a Pixel 3 turns the handset into a pocket supercomputer: a  $512 \times 512$  face crop slides through an int8-quantised MobileNet-v2 whose ReLU6 activations have been channel-fused into tight lookup tables, while the tiny DMA engine pipelines the next tile before the current one retires. The whole embedding pops out in 25 ms—ten times swifter than the Snapdragon 845 could dream—and after crunching a hundred photos the battery gauge has sagged barely 2 %. Thanks to this Google-designed silicon, a billion Google Photos users can now cluster their friends in real time without ever shipping a single byte to the cloud [13].

Inside Waymo’s roadside cabinets, a fleet of TPU v4i chips now ingests every 64-beam LiDAR sweep—roughly 130 000 points per frame—through an 800 MB segmentation network whose weights are streamed from host DRAM on demand. Auto-sharding via JAX/pjit pins each sparse-convolution kernel to a deterministic slice of the 3-D torus, keeping the 99.9 % availability SLA intact [14]. The result is a steady 50 FPS pipeline with a 48 ms tail latency: the onboard GPU footprint shrinks by 70 %, and the energy cost per autonomous mile falls by fourfold, letting Waymo’s Pacificas navigate city blocks on a whisper of electricity.

## 4 Challenges & Optimization Directions

### 4.1 Challenges

Contemporary TPU deployments confront a constellation of interrelated bottlenecks whose severity escalates with both model and fleet scale. First, the memory wall manifests as a fundamental imbalance between arithmetic throughput and off-chip bandwidth: aggregate HBM capacity per FLOP has asymptotically plateaued across successive process nodes, and state-of-the-art language models exceeding 200 B parameters routinely overflow the 32–64 MiB aggregate on-chip SRAM budget. The resulting compulsory DRAM traffic stalls MXU pipelines, producing idle utilization rates above 50 % during GPT-3-class training runs, even though peak FLOPS remain nominally available. Second, dynamic-shape semantics endemic to natural-language workloads—manifesting as variable sequence lengths, dynamic attention masks, and control-flow-dependent tensor shapes—force XLA’s ahead-of-time compiler to regenerate and re-specialize kernels for each concrete shape tuple. The recompilation latency, although amortized over large micro-batches, still introduces 5–10ms dispatch stalls that cumulatively erode pipeline efficiency and complicate tail-latency service-level objectives.

### 4.2 Optimization Directions

To counteract the aforementioned constraints, a multi-pronged optimization roadmap is being pursued. First, TPU v4 integrates SparseCore accelerators that natively elide 2:4 structured zero weights, boosting effective FLOPS for transformer attention blocks by 2.0× while introducing <1 % degradation in validation accuracy. Second, a double-buffer weight-streaming protocol retains model parameters in host DRAM and overlaps DMA transfers with systolic computation, thereby supporting trillion-parameter models without enlarging on-chip SRAM. Third, compiler-ahead sharding via JAX/pjit synthesizes static SPMD schedules that interleave compute phases with asynchronous all-reduce collectives, reducing pipeline bubbles by 18 % and 99-percentile tail latency by 30 %. Fourth, optical circuit switches (OCS) dynamically reconfigure the 3-D torus topology in  $\approx 100$   $\mu$ s, cutting collective latency by 30 % during trillion-parameter training campaigns. Fifth, mixed-precision pipelines employ stochastic rounding between bfloat16 activations and int8 weights, preserving convergence behaviour while doubling effective throughput. Finally, the Pathways runtime system performs

software–hardware co-mapping of SPMD programs directly onto the 2-D torus, eliminating host CPU mediation and further shrinking end-to-end execution time. Continued investment in compiler openness, dynamic-shape support, and cross-framework tooling will determine whether TPUs maintain that lead as models diversify beyond the dense-GEMM regime [15].

## 5 Conclusion

From the 40 W, 28 nm TPU v1 focused solely on inference to the 175 W, 7 nm TPU v4 delivering 1.1 PFLOPS of dense compute and 2.2 PFLOPS with 2:4 sparsity, Google’s TPU family has consistently validated the thesis that domain-specific architectures can overturn the traditional CPU/GPU duopoly for deep learning workloads. The micro-architectural pillars—systolic arrays, deterministic VLIW scheduling, compiler-managed scratchpads, SparseCores, optical-circuit-switched interconnects—translate into measurable benefits: 15–30× faster training, 30–80× better energy efficiency, and 2–4× lower TCO across vision, language, and scientific computing. Real-world case studies demonstrate that these gains are not synthetic. BERT-Large pre-training finishes in 76 minutes, AlphaFold inference on 200 M proteins consumes <1 MWh, Google Photos clustering is now feasible on-device, and Waymo LiDAR pipelines meet stringent 50 ms latency SLOs at 99.9 % availability. Emerging benchmarks such as SuperGLUE further confirm that TPUs retain their lead across diverse language-understanding tasks.

Yet the memory wall, dynamic-shape compilation, and ecosystem breadth remain active research frontiers. Emerging mitigations—weight-streaming to extend parameter capacity beyond on-chip SRAM, SparseCores to exploit sparsity without scatter-gather penalties, JAX/pjit for ahead-of-time sharding, and optical circuit switches for sub-10  $\mu$ s reconfiguration—suggest that the next leap in TPU capability will stem from algorithm–hardware co-evolution rather than transistor scaling alone. For practitioners, the pragmatic guideline is straightforward: when workloads are dominated by dense or semi-structured tensor operations, can tolerate deterministic scheduling, and are compatible with TensorFlow or JAX, TPUs currently offer the best cost-performance envelope in the public cloud. Continued investment in compiler openness, dynamic-shape support, and cross-framework tooling will determine whether TPUs maintain that lead as models diversify beyond the dense-GEMM regime.

## References

1. Annepaka, Y., Pakray, P.: Large language models: a survey of their development, capabilities, and applications. *Knowl. Inf. Syst.* 67(3), 2967–3022 (2025)
2. Chowdhery, A., et al.: PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311* (2022)
3. Raiaan, M.A.K., Mukta, M.S.H., Fatema, K., et al.: A review on large language models: architectures, applications, taxonomies, open issues and challenges. *IEEE Access* 12, 26839–26874 (2024)

4. Kimm, H., Paik, I., Kimm, H.: Performance comparison of tpu, gpu, cpu on google colabatory over distributed deep learning. In: Proc. 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc), pp. 312–319. IEEE, (2021)
5. Dean, J., Patterson, D.: A new golden age for computer architecture. *Commun. ACM* 62(2), 48–60 (2019)
6. Jouppi, N.P., et al.: A domain-specific supercomputer for training deep neural networks. *Commun. ACM* 63(7), 105–112 (2020)
7. Norrie, T., et al.: Google TPU v4: an optically reconfigurable supercomputer for machine learning. In: Hot Chips 33. IEEE, (2021)
8. Young, C., et al.: TPU v4: performance, power, and pod-scale insights. Google Research Blog. <https://ai.googleblog.com/2021/06/tpu-v4-performance-power-and-pod-scale.html>, last accessed 2025/08/21
9. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: Proc. OSDI 2016, pp. 265–283. USENIX, (2016)
10. Eric, Z.: TPU v3 Chip Image. <https://cn.bing.com/images/search?view=detailV2&ccid=yC%2Bauuto&id=AD1AE08A932985BE17E97F3936D3273B822545C9>, last accessed 2025/08/21
11. You, Y., et al.: Large batch optimization for deep learning: training BERT in 76 minutes. In: Proc. ICML 2019, pp. 1–10. PMLR, (2019)
12. Senior, A.W., et al.: Improved protein structure prediction using potentials from deep learning. *Nature* 577, 706–710 (2020)
13. Pham, H., et al.: Meta pseudo labels. In: Proc. CVPR 2021, pp. 1–10. IEEE, (2021)
14. Chen, L.-C., et al.: Searching for efficient multi-scale architectures for dense image prediction. In: Proc. NeurIPS 2018, pp. 1–12. Curran Associates, Red Hook (2018)
15. Lepikhin, D., et al.: GShard: scaling giant models with conditional computation and automatic sharding. In: Proc. ICLR 2021. OpenReview.net, (2021)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

