Unified Splitting: Inference Utilizing Unified Memory Architecture

Ayushmaan Puri
UC Scholars Program
University of California, San Diego
Computer Science and Engineering

Dr. Patrick Pannuto
Associate Professor
University of California, San Diego
Computer Science and Engineering

Raymond Dueñas
PhD Student
University of California, San Diego
Computer Science and Engineering

***Abstract*—Mobile computers, like smartphones, now run ever-larger neural networks under tight power and cost budgets. While phones ship with multiple CPUs, these processors are ignored for AI processing in favor of the "always superior" GPU. Our work finds that the CPU can and should be part of the AI inference story: Convolutions (which act like powerful image filters) are very parallelizable and run 370–740x faster on the GPU, while fully connected (less parallelizable) layers see only 37–38x gains. Identifying this gap, we assign convolutions to the GPU and the remaining layers to the CPU. Unified memory lets both processors share data without copies, and a queue-based pipeline overlaps their work across batches, keeping each busy. Tested on ResNet-18 and ResNet-50, our scheme delivers an 8% and 17% throughput boost over GPU-only execution, without any changes to the model. The approach generalizes to diverse networks and offers a practical path to faster, energy-aware inference on resource-constrained platforms, achieving consistent performance improvements across ResNet-18 (20.75%), ResNet-50 (16.84%), MobileNetV2 (13.87%), EfficientNet-B2 (8.49%), and ViT-Tiny (24.39%).**

I. Introduction

The proliferation of artificial intelligence applications at the network edge has driven significant advances in heterogeneous computing platforms. Modern embedded systems, particularly those designed for AI workloads, commonly integrate powerful multi-core ARM processors alongside GPU accelerators within unified memory architectures [1]. Despite this computational diversity, contemporary inference frameworks predominantly channel all neural network operations through GPU pipelines, effectively relegating CPU resources to auxiliary tasks such as data preprocessing and system orchestration.

This GPU-centric approach, while leveraging parallel processing capabilities for computationally intensive operations, creates a fundamental resource utilization imbalance. Empirical observations across multiple edge platforms reveal suboptimal CPU utilization rates during intensive inference workloads, representing substantial untapped computational capacity. Meanwhile, GPU resources may encounter throughput limitations when processing operations that do not inherently benefit from massive parallelization.

II. Related Work

A. DNN Partitioning and Offloading

The idea of splitting neural network computation across processors originated in the mobile-cloud partitioning literature. Kang et al. introduced Neurosurgeon [4], a layer-granularity scheduler that profiles per-layer latency and data size to find an optimal split point between a mobile device and a cloud server. Neurosurgeon demonstrated that different layer types exhibit vastly different computation-to-communication ratios, making blanket offloading suboptimal. Our work adopts a similar layer-level profiling philosophy but applies it to *on-device* CPU–GPU partitioning rather than device–cloud offloading, eliminating network latency from the equation entirely.

Lane et al. proposed DeepX [5], a software accelerator that decomposes DNN architectures into unit-blocks and maps them onto heterogeneous mobile processors (CPUs, GPUs, DSPs). DeepX further applies runtime layer compression to reduce per-block computational cost. While DeepX targets general heterogeneous scheduling with model modification, our approach preserves the original model architecture and focuses specifically on exploiting the asymmetric GPU speedup across layer types.

B. Heterogeneous Inference on Edge Platforms

More recent work has explored CPU–GPU co-execution specifically on embedded platforms with unified memory. Park et al. [6] proposed a layer-wise processor selection method on unified memory for on-device *training*, demonstrating up to 28.4% latency reduction on an ODROID-XU4 by selecting between CPU and GPU for each layer's forward and backward passes. Their work validates that unified memory can enable efficient inter-processor data sharing, but targets training rather than inference and does not exploit pipelined execution across batches.

Jeong et al. developed the JEDI framework [7], a TensorRT-based system for Jetson platforms that pipelines DNN inference across GPUs and DLAs (Deep Learning Accelerators) with multi-threading, buffer management, and network duplication. JEDI achieves 101–680% throughput improvement over GPU-only baselines by leveraging all available accelerators. However, JEDI relies on TensorRT's graph-level optimizations and DLA hardware, whereas our approach operates at the PyTorch level without specialized compiler support and targets the CPU as the secondary processor, which is available on a far wider range of devices.

C. Model Architectures

We evaluate our splitting strategy across five architectures that span the design space of modern vision models. ResNet-18 and ResNet-50 [8] introduced residual connections that enabled training of very deep networks and remain standard benchmarks for inference optimization. MobileNetV2 [9] uses depthwise separable convolutions and inverted residual blocks to minimize computation while preserving accuracy, representing mobile-optimized CNN design. EfficientNet-B2 [10] applies compound scaling across depth, width, and resolution to achieve strong accuracy-efficiency trade-offs. ViT-Tiny [11] adapts the transformer architecture to vision tasks using patch embeddings and self-attention, representing a fundamentally different computational pattern from CNNs. Testing across this range validates that our approach generalizes beyond any single architectural family.

D. Positioning of This Work

Prior partitioning work has largely focused on either device–cloud splits (Neurosurgeon) or required model modifications (DeepX) or specialized accelerator hardware (JEDI). Our contribution is a lightweight, model-transparent CPU–GPU splitting strategy that exploits unified memory and CPU cache residency for inference on commodity edge hardware. The closest prior work is Park et al. [6], but their method targets training, operates on weaker hardware, and does not pipeline across batches. Our pipelined queue-based design enables concurrent CPU and GPU execution across successive batches, a mechanism not explored in prior unified-memory partitioning work.

III. Methods

A. Core Architecture

- **GPU Execution:** Convolutional layers and operations requiring high parallelism remain on GPU.
- **CPU Execution:** Fully connected layers, batch normalization, and sequential operations move to CPU.
- **Unified Memory:** Shared memory between the CPU and the GPU that eliminates explicit data copying between processors.
- **Queue Coordination:** Implements lightweight queue system for inter-processor communication.

The implementation maintains original model architecture while redistributing computational workload. GPU processors handle parallel-intensive operations like convolutions, while CPU cores process operations with limited parallelization benefits. This approach utilizes previously idle CPU resources without compromising GPU performance for suitable operations.

B. Layer Performance Characterization

To determine optimal splitting points, we systematically measured execution time for individual neural network layers across both CPU and GPU processors.

Profiling Protocol:

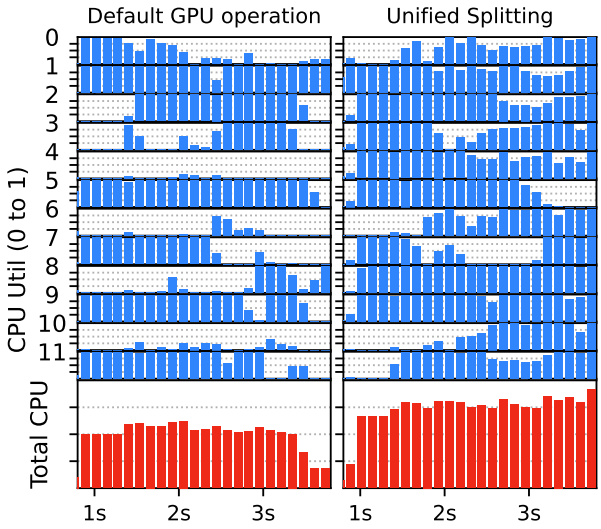

Fig. 1. A view of utilization across all CPU cores with Full GPU (left) and UnifiedSplitting implementation (right).

- Extract individual layer operations from complete network architectures.
- Execute multiple timing measurements per layer on each processor type.
- Calculate statistical mean and variance for execution times.
- Compute CPU-to-GPU performance ratios for each operation class.
- Identify layers where GPU advantage is minimal (ratio < 50x).

C. Performance Categories Identified

High Benefit: Convolutional layers (300–500x faster on GPU).
Moderate Benefit: Activation functions (100–200x faster on GPU).
Low Benefit: Fully connected layers (30–40x faster on GPU).
Minimal Benefit: Batch normalization (25–35x faster on GPU).

Layers in the "Low" and "Minimal" categories became candidates for CPU execution in our split implementation.

D. Experimental Platform

- NVIDIA Jetson AGX Orin developer kit.
- 12-core ARM Cortex-A78AE CPU.
- 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores.
- 64GB unified LPDDR5 memory.
- PyTorch framework with CUDA support [2].

E. Test Configurations

Based on our layer performance characterization, we designed test configurations to validate our hypothesis that strategic layer partitioning improves performance by utilizing idle

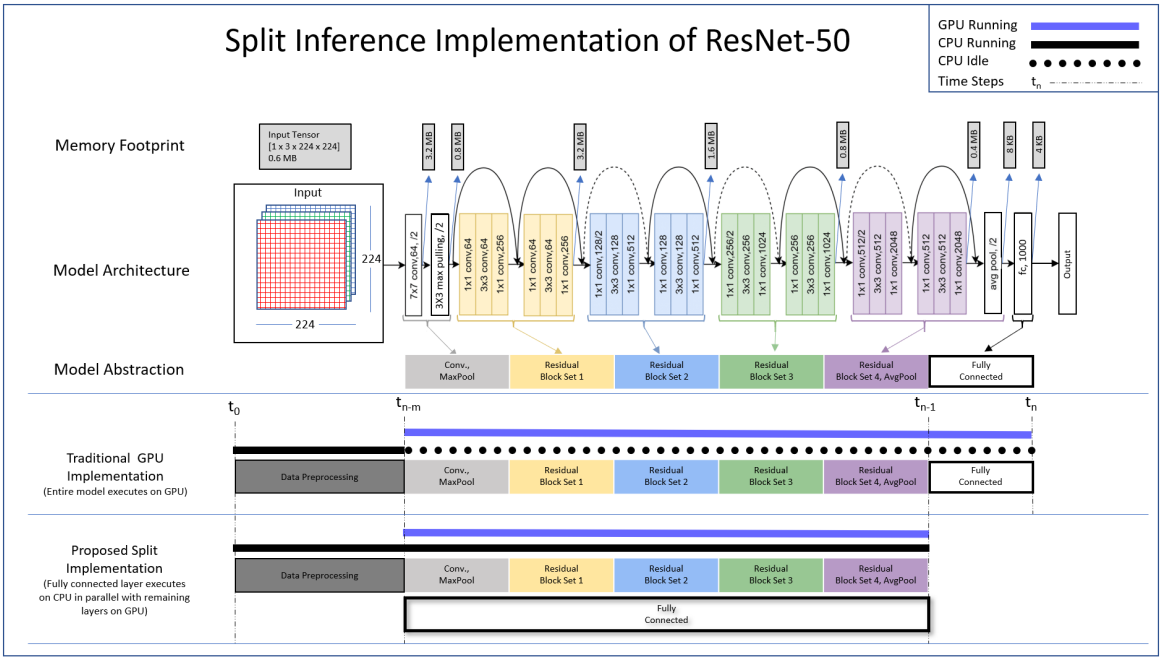

Fig. 2. Overview of the ResNet 50 model, and a before-and-after view of the UnifiedSplitting implementation highlighting throughput gains.

CPU resources for operations with minimal GPU acceleration benefits.

1) Baseline Configuration: Complete model execution on GPU only.

- *Purpose:* Establish performance ceiling for GPU-exclusive execution.
- *Expected limitation:* CPU cores remain idle (∼20% utilization) while GPU handles all operations including those with low parallelization benefits.

2) Split Implementation: Convolutional layers on GPU, fully connected layers on CPU.

- *Rationale:* Our profiling identified fully connected layers show only 30–40x GPU acceleration (vs 300–500x for convolutions).
- *Hypothesis:* Moving FC layers to idle CPU cores should improve throughput by enabling concurrent processing.
- *Expected benefit:* GPU freed from sequential operations, CPU cores activated for meaningful computation.

3) Model Selection: ResNet-18, ResNet-50 [8], MobileNetV2 [9], EfficientNet-B2 [10], ViT-Tiny [11].

- *ResNet-18/50:* Standard CNN architectures with clear conv/FC separation test basic splitting effectiveness.
- *MobileNetV2:* Depthwise separable convolutions test splitting benefits on mobile-optimized architectures.
- *EfficientNet-B2:* Compound scaling model validates approach across different efficiency-accuracy trade-offs.
- *ViT-Tiny:* Vision transformer tests splitting strategy on attention-based architectures with different computational patterns.
- *Architectural diversity:* Range from pure CNNs to hybrid attention models evaluates generalizability of splitting approach.

4) Simulated Device Configurations: To validate our approach across diverse edge hardware capabilities, we simulated multiple device profiles on the Jetson AGX Orin by controlling CPU core activation states, CPU/GPU clock frequencies, and EMC (External Memory Controller) frequencies:

- *Jetson Xavier NX:* Reduced CPU cores and GPU compute units to match Xavier NX specifications.
- *Jetson Nano:* Limited CPU cores (4-core) and significantly reduced GPU clock rates.
- *Jetson TX2:* Configured dual-core Denver + quad-core ARM cluster with corresponding GPU limitations.
- *Coral Dev Board:* ARM Cortex-A53 configuration with minimal GPU simulation (CPU-dominant testing).
- *CPU Bottleneck Test:* Maximum GPU clocks with artificially constrained CPU performance to isolate CPU limitations.
- *GPU Bottleneck Test:* Maximum CPU performance with constrained GPU clocks to isolate GPU limitations.

5) Batch Size Range: 16, 32, 64, 128, 256 samples.

- *Small batches (16–32):* Test whether coordination overhead dominates in low-parallelism scenarios.
- *Medium batches (64–128):* Expected optimal range where pipeline benefits outweigh overhead.
- *Large batches (256+):* Validate approach under GPU memory pressure conditions.
- *Hypothesis:* Performance gains should peak in medium range where both processors can maintain high utilization.

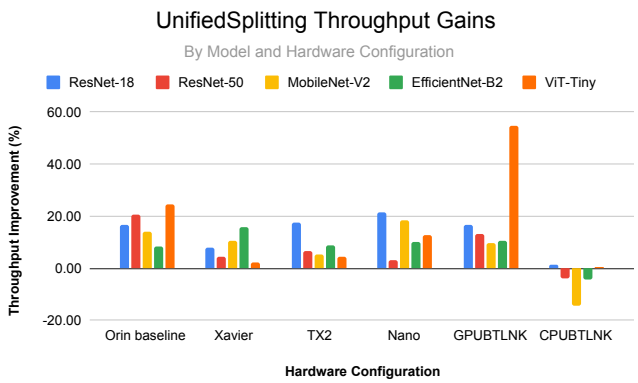

Fig. 3. Average percentage throughput gains vs. the full GPU implementation for each model and hardware configuration.

6) Key Metrics: Throughput (images/second), resource utilization, inference latency.

- *Throughput:* Primary metric validating our concurrent execution hypothesis.
- *Resource utilization:* Confirms CPU activation and sustained GPU performance.
- *Latency:* Ensures coordination overhead doesn't introduce unacceptable delays.

F. Data Movement Cost Optimization

The performance benefits stem from two key factors that minimize data movement overhead:

1) **Unified Memory Advantage:** The unified memory architecture eliminates explicit memory copies between CPU and GPU address spaces [1]. Rather than requiring expensive `cudaMemcpy()` operations, both processors access shared memory regions directly.
2) **CPU Cache Utilization:** A critical performance factor involves leveraging CPU cache capacity for small layer weights. For final layers like fully connected operations, the weight matrices are small enough to fit entirely in CPU cache memory. After the first batch loads weights into cache, subsequent batches benefit from a "load weights from memory + compute" to "cache-resident compute only" transition. This cache residency provides substantial speedup for repeated inference operations, particularly benefiting edge layers where weight matrices are relatively small compared to earlier convolutional layer filters.

G. Measurements and Controls

To ensure reliable and reproducible results, we implemented several control measures to eliminate confounding variables:

- **Temperature Regulation:** All experiments conducted with the Orin's fan always at 100% to prevent dynamic cooling during and between tests.
- **Cache Warm-up Protocol:** Each test configuration included a 5-batch warm-up phase before, as well as a 5-batch cool-down phase after measurement collection to ensure consistent cache states across CPU and GPU memory hierarchies. This eliminates first-run cache loading effects that could skew initial performance measurements.
- **Process Isolation:** Background processes minimized and system load monitored to ensure consistent resource availability. Tests repeated if system utilization exceeded baseline thresholds during measurement periods.
- **Power State Consistency:** System configured to maximum performance power profiles with disabled dynamic frequency scaling to eliminate variable clock rate effects on timing measurements.
- **Memory State Reset:** System memory cleared between different configuration tests to prevent memory allocation patterns from influencing subsequent measurements.

IV. Results

The split implementation consistently outperformed GPU-only baselines across all tested configurations, achieving 8–20% throughput improvements depending on model architecture and batch size. The one exception was the CPU bottleneck configuration, where performance declined as expected, since the split implementation depends on CPU capacity, which is the constrained resource in that test.

Performance improvements were most pronounced for larger models (ResNet-50, ResNet-18) where fully connected operations represent a larger portion of total computation time. The gains remained consistent across multiple test runs, demonstrating reliable performance benefits.

V. Conclusions

This work demonstrates that the conventional approach of routing all neural network operations exclusively through GPU resources leaves significant performance gains on the table. Our split implementation achieves consistent 8–20% throughput improvements across diverse architectures by strategically utilizing previously idle CPU cores for operations with minimal GPU acceleration benefits.

The key insight driving these improvements is that not all neural network operations are created equal. While convolutional layers benefit enormously from GPU parallelization (300–500x speedup), operations like fully connected layers show much smaller GPU advantages (30–40x speedup) and can be effectively relocated to CPU cores. The additional benefit of CPU cache residency for small weight matrices transforms these operations from memory-bound to compute-bound, further improving their CPU execution efficiency.

VI. Future Work

Weight Distribution Analysis: Our results reveal that performance gains correlate strongly with model weight distribution patterns. ResNet architectures achieve higher gains due to their characteristic weight "ballooning" in middle layers followed by compact final classifiers that fit entirely in CPU cache. Future work should systematically analyze weight distribution patterns across broader architecture families to predict optimal splitting strategies.

Transformer Architecture Scaling: While ViT-Tiny showed surprisingly strong performance, larger transformer models present different challenges. Full-scale transformers have more uniform weight distributions across layers, with attention heads and feedforward networks maintaining consistent parameter counts. Research should investigate whether larger transformer final projections exceed CPU cache capacity, potentially limiting splitting benefits. Additionally, attention mechanism computation patterns may reveal new CPU-suitable operations beyond traditional fully-connected layers.

Cache-Aware Model Design: The cache residency insight suggests opportunities for co-designing neural architectures with deployment hardware. Future research could explore architectures that intentionally concentrate parameters in cache-friendly distributions – small edge layers for CPU processing, large middle layers optimized for GPU parallelism.

REFERENCES

[1] M. Harris, "Unified Memory in CUDA 6," NVIDIA Technical Blog, Nov. 2013. [Online]. Available: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/. [Accessed: Aug. 19, 2025].

[2] "Jetson AGX Orin for Next-Gen Robotics," NVIDIA. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. [Accessed: Aug. 15, 2025].

[3] "Unified memory — HIP 6.2.41133 Documentation," ROCm Documentation. [Online]. Available: https://rocm.docs.amd.com/projects/HIP/en/docs-6.2.0/how-to/unified_memory.html. [Accessed: Aug. 19, 2025].

[4] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *Proc. 22nd Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Xi'an, China, Apr. 2017, pp. 615–629.

[5] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," in *Proc. 15th ACM/IEEE Int. Conf. Information Processing in Sensor Networks (IPSN)*, Vienna, Austria, Apr. 2016, pp. 1–12.

[6] S. Park, J. Lee, and H. Bahn, "Accelerating On-Device Learning with Layer-Wise Processor Selection Method on Unified Memory," *Sensors*, vol. 21, no. 7, p. 2364, Mar. 2021.

[7] E. Jeong, J. Kim, and S. Ha, "TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards," *ACM Trans. Embedded Computing Systems*, vol. 21, no. 5, pp. 1–26, Oct. 2022.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 4510–4520.

[10] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proc. 36th Int. Conf. Machine Learning (ICML)*, Long Beach, CA, USA, Jun. 2019, pp. 6105–6114.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. Int. Conf. Learning Representations (ICLR)*, May 2021.