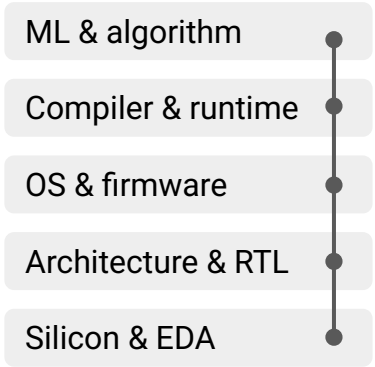
Yen-Chieh Huang 黃彥傑

Building efficient computing systems across the stack

Computer architecture · ML systems · HW/SW co-design

National Taiwan University (NTU)
M.S. in Electrical Engineering · 2024-2026

National Yang Ming Chiao Tung University (NYCU)
B.S. in Computer Science · 2020-2024



3 papers
ISCA'25, EMNLP'26 · first author
TCAD'26 · co-author

2 / 201
NYCU graduation rank
GPA 4.28 / 4.30 (top 1%)

7x Recipient
Academic excellence award
(for top 5% of the CS department)

✉ r13921038@ntu.edu.tw · 🌐 lashhw.github.io

Building efficient systems across the stack

High-level → Low-level

ML & algorithm | Compiler & runtime | OS & firmware | Architecture & RTL | Silicon & EDA

Ray-tracing accelerator
AQB8 · ISCA 2025 · first author

Long-context LLM inference
WG-KV · EMNLP 2026 · first author

RISC CPU: from RTL to GDSII
Final project @ NYCU ICLab
· overall A+ (top ~18%)

TinyML on microcontrollers
DupNAS · TCAD 2026 · toolchain development

VLSI routers
Course project @ NYCU

VLM for autonomous driving
Final project @ NTU DLCV
· 1st place · team leader

Linux kernel module & page-table manipulation
Course project @ NTU SSDI

MLP accelerator on FPGA
Course project @ NYCU

Mini-Pascal compiler
Course project @ NYCU

Linux sandbox & debugger
Course project @ NYCU APUE

LLVM dataflow graph extraction for high-level synthesis
Course project @ NTU

Additional projects, with code and details, at lashhw.github.io/projects

Academic foundations

Computer architecture & embedded

- 微處理機系統原理與實作 A+
- 高等計算機結構 A-
- 平行程式設計 A+
- 計算機組織 A+
- 嵌入式即時系統 A+
- 嵌入式系統總整與實作 A+

OS & compilers & system software

- 系統軟體設計與實作特論 A+
- 高等UNIX程式設計 A+
- 計算機系統管理 A+
- 計算機網路概論 A+
- 作業系統概論 A+
- 高等編譯器設計 A+

Digital IC design & EDA

- 積體電路設計實驗 A+
- 電子學(一) A+
- 超大型積體電路導論 A+
- 演算法在繞線問題的實務應用 A+
- 數位系統設計 A+
- 數位電路實驗 A+

AI & machine learning

- 深度學習於電腦視覺 A+
- 圖形識別 A+
- 機器學習 A+
- 人工智慧概論 A+
- 資料科學 A
- 深度學習概論與應用 A+

Algorithms & theory

- 正規語言與計算理論 A+
- 離散數學 A+
- 演算法概論 A+
- 物理 A+ / A+
- 資料結構與物件導向程式設計 A+

Mathematics & signal processing

- 訊號與系統 A+
- 數位訊號處理概論 A+
- 機率 A+
- 統計學 A+
- 線性代數 A
- 微積分甲 A / A+
- 微分方程 A+

Research & projects

- 資訊工程專題 A+ / A+
- 碩士論文 A+

2 / 201
NYCU graduation rank
GPA 4.28 / 4.30 (top 1%)

7x Recipient
Academic excellence award
(for top 5% of the CS department)

Peer-reviewed research across architecture, systems, and ML

AQB8 · ISCA'25
(first author)

Source: ICORE2026
Rank: A*
Field Of Research: CSE - Computer Systems Engineering (h-index) (citation)
Field Of Research: 4606 - Distributed computing and systems software (h-index) (citation)
Field Of Research: 4612 - Software engineering (h-index) (citation)

WG-KV · EMNLP'26
(first author)

Source: ICORE2026
Rank: A*
Field Of Research: 4602 - Artificial intelligence (h-index) (citation)

DupNAS · TCAD'26
(toolchain development)



ICORE Conference Rankings: A* (Flagship) > A (Excellent) > B (Good) > C (Satisfactory).
Scimago Journal Rank: Q1 (Best quartile) > Q2–Q4.

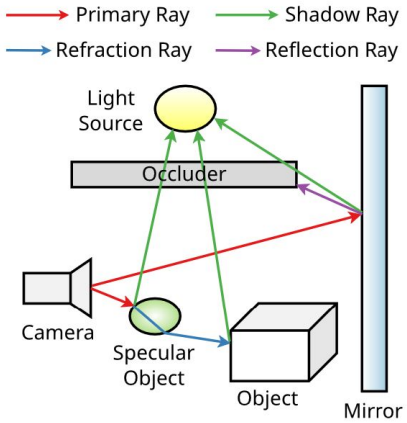
DEEP DIVE 1 / 3

Ray tracing accelerator

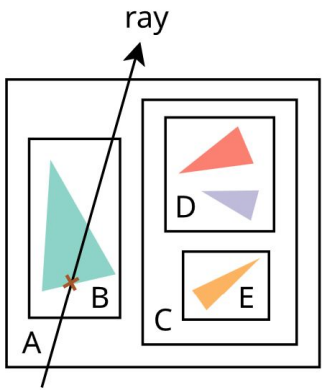
Efficient ray tracing via quantization, from algorithm to silicon

AQB8 · ISCA 2025 · first author

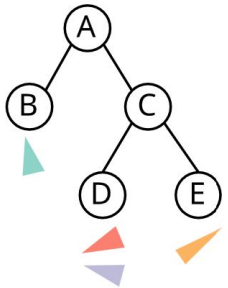
How GPUs render 3D images by tracing rays



Ray tracing



Bounding boxes



BVH tree

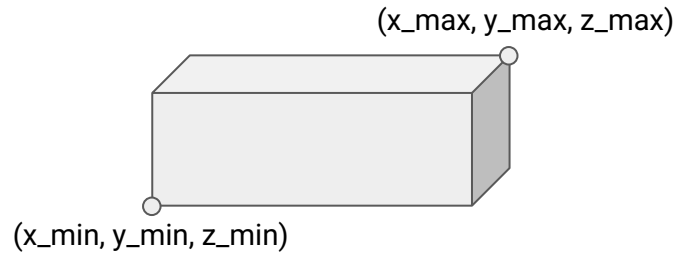
1. To render a scene into a realistic image, the GPU traces rays of light and finds the object each ray hits.
2. Scenes hold millions of triangles, so they are grouped into a tree of nested bounding boxes, called a BVH.
3. A ray opens only the boxes it passes through; one box test can skip thousands of triangles.
4. Modern GPUs have dedicated hardware accelerators that walk this tree.

Huang, Yang, Yeh - AQB8: Energy-Efficient Ray Tracing Accelerator through Multi-Level Quantization [ISCA 2025]

Bounding boxes dominate ray-tracing (RT) memory traffic

74%

of the RT accelerator's memory traffic is box data — more than the triangles themselves.



So RT accelerators spend much of their time waiting on memory — and every box test needs large, power-hungry FP32 arithmetic.

Obvious fix 1: just use reduced-precision floating points (e.g., FP16)

Rounding makes bounding boxes bigger, leading to increased traversal steps:

2.7× box tests **19.6×** triangle tests



Obvious fix 2: store boxes in low-bits, and convert back to FP32 on-the-fly

8-bit boxes in memory → convert back → FP32 test

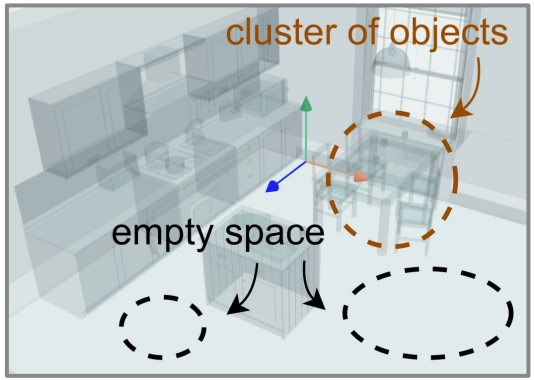
Saves memory traffic, but every test still runs on 32-bit FP units.

Can we test bounding boxes *directly* in low-bits without exploding the number of traversal steps?

Huang, Yang, Yeh - AQB8: Energy-Efficient Ray Tracing Accelerator through Multi-Level Quantization [ISCA 2025]

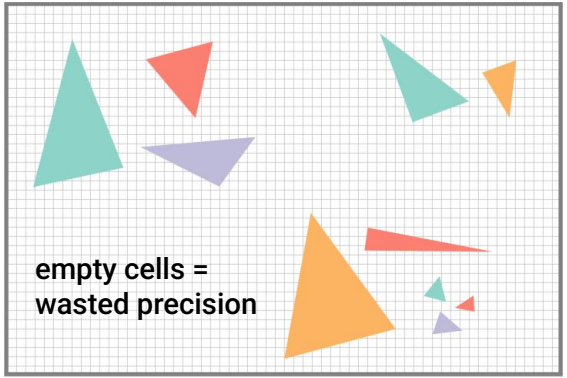
Scenes are sparse, so spend precision only where geometry is

(a) Scenes are sparse
Geometry sits in a few clusters; most of the scene is empty space



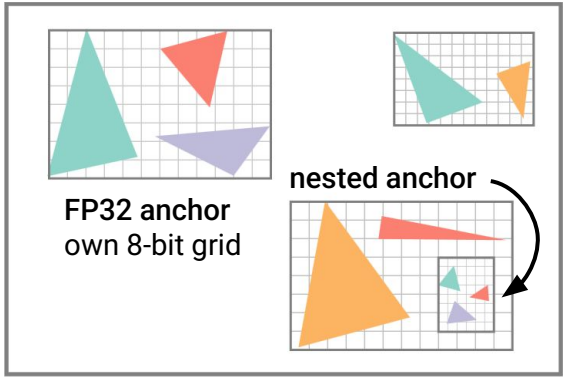
Country Kitchen, a real test scene

(b) One global frame
Small objects can be anywhere, so the grid must be fine everywhere



only a few grid cells ever hold geometry

(c) Multi-level quantization
Fine grids only inside clusters, each small enough for 8 bits

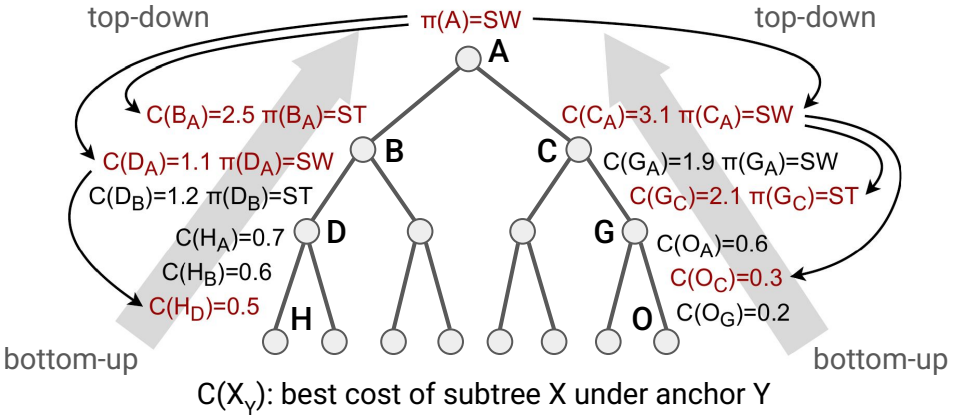


fine 8-bit grids only where geometry is

***Sparsity* is why 8 bits suffice: each anchor spans its cluster, not the empty space around it.**
Across our 7 scenes, 4.9M BVH nodes share just 77K FP32 anchors, about one per 64 nodes.

Start from any BVH, then let a cost model choose the anchors

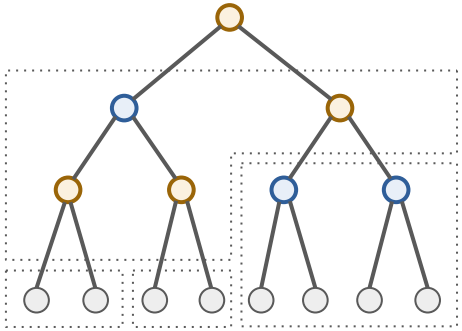
How anchors are chosen optimal via DP in $O(n \log^2 n)$



$C(X_Y)$: best cost of subtree X under anchor Y

SWITCH Its FP32 box anchors a new cluster

STAY Keep the current anchor's 8-bit grid



Result: 4 clusters, one FP32 anchor each

Cost model S = surface area; **staying** inflates S, **switching** adds c_s

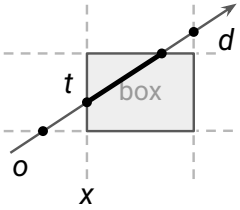
$$C(N) = \frac{1}{S(N)}\left[\sum_{N_i} T(N_i)S(N_i) + c_i \sum_{N_l} S(N_l)|N_l|\right], \text{where } T(N_i) = \begin{cases} c_t + c_s, & \text{if } \pi(N_i) = \text{SWITCH}, \\ c_t, & \text{if } \pi(N_i) = \text{STAY}. \end{cases}$$

Quantize the ray once per cluster: integer-only box tests

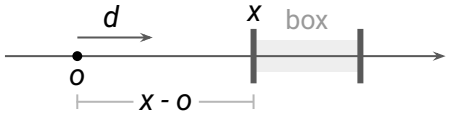
When a ray enters a cluster → FP32 anchor test → quantize the ray → ~1 in 8 box tests

For each box in the cluster → 8-bit box → integer test → ~7 in 8 box tests

How a box test works



A ray crosses a face at $t = (x - o) / d$. It hits the box if every entry comes before every exit.



Let $w = 1 / d$ and $b = -o / d$,

Original $t = w \cdot x + b$ — one FP32 multiply-add

Quantized $q_t = \pm 2^r (m \cdot q_x) + q_b$ — 8-bit multiply, shift, 32-bit add

What it buys in hardware

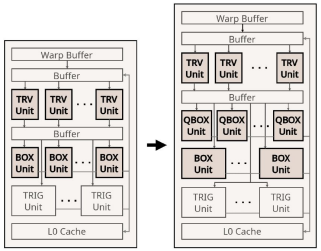
BOX unit: FP32 × ▸ FP32 + ▸ min/max ▸ compare

QBOX unit: INT8 × ▸ shift ▸ ± ▸ INT32 + ▸ min/max ▸ compare

5.1×
smaller area*

5.7×
less energy per test*

* QBOX vs. BOX unit, TSMC 40nm



From algorithm to silicon

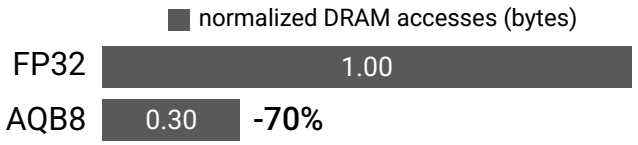
1 ALGORITHM	Quantized BVH tree construction and traversal algorithms	C++ · Python
2 ENCODING	Compact BVH layout and quantized ray representations	C++
3 ARCHITECTURE	RT accelerator microarchitecture and implementation	HLS C++ · SystemVerilog
4 FPGA PROTO	Accelerator prototyping on Zynq UltraScale+ MPSoC	Vivado · Vitis · PetaLinux
5 ASIC EVAL	TSMC 40nm synthesis, simulation & power/area analysis	Catapult HLS · Design Compiler · QuestaSim · PrimePower

Less memory traffic, energy, and area at a small traversal cost

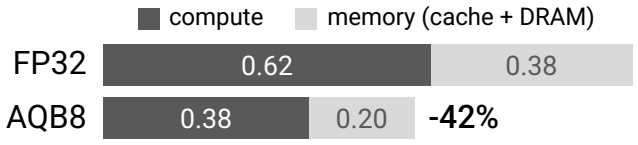


evaluated scenes

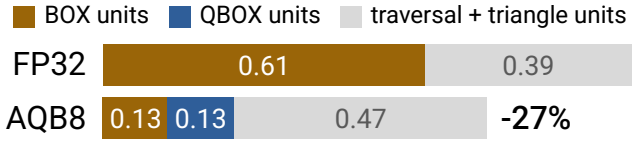
Memory traffic average of 7 scenes



Energy average of 7 scenes



Area at equal throughput



Extra traversal steps compared to FP32

	FP16*	AQB8
Box tests	1.3–2.7×	**1.03–1.06×**
Triangle tests	2.4–19.6×	**1.06–1.31×**

* Baseline with every box stored and tested in FP16.

For more details, see our paper and code.

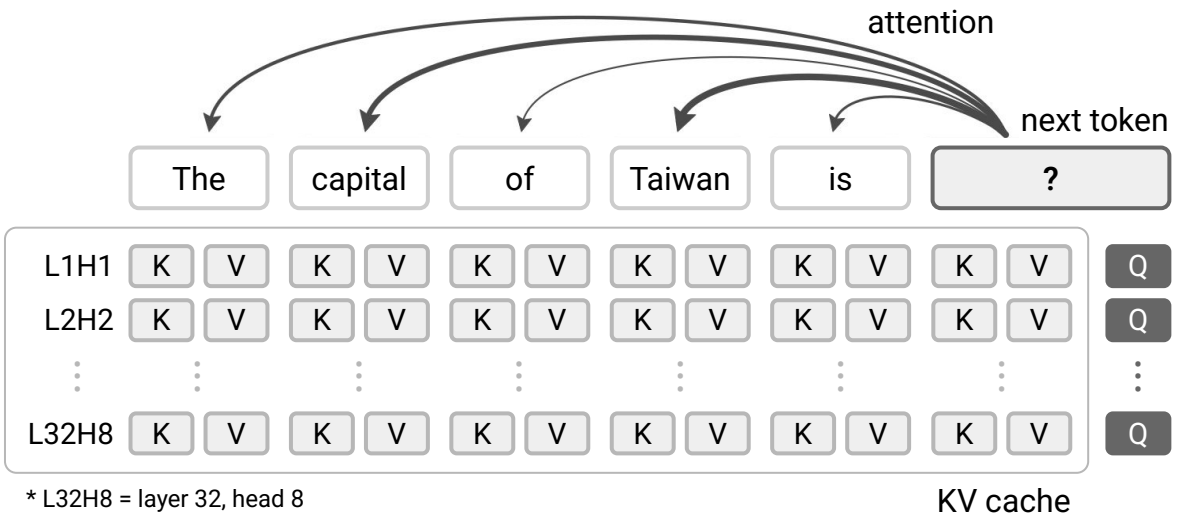
DEEP DIVE 2 / 3

Long-context LLM inference

Learning which tokens are worth caching, for faster and leaner LLM inference

WG-KV · EMNLP 2026 · first author

How an LLM remembers its context



1. An LLM generates one token at a time.
2. Attention works like a search: each new token's query interacts with previous keys and values to gather relevant information.
3. Every layer and head runs its own "search", so each keeps its own key and value for every token: the KV cache.
4. With full attention, as in Llama-3.1-8B, the cache grows with the text and each new token reads all of it.

The bottleneck: every token is cached, useful or not

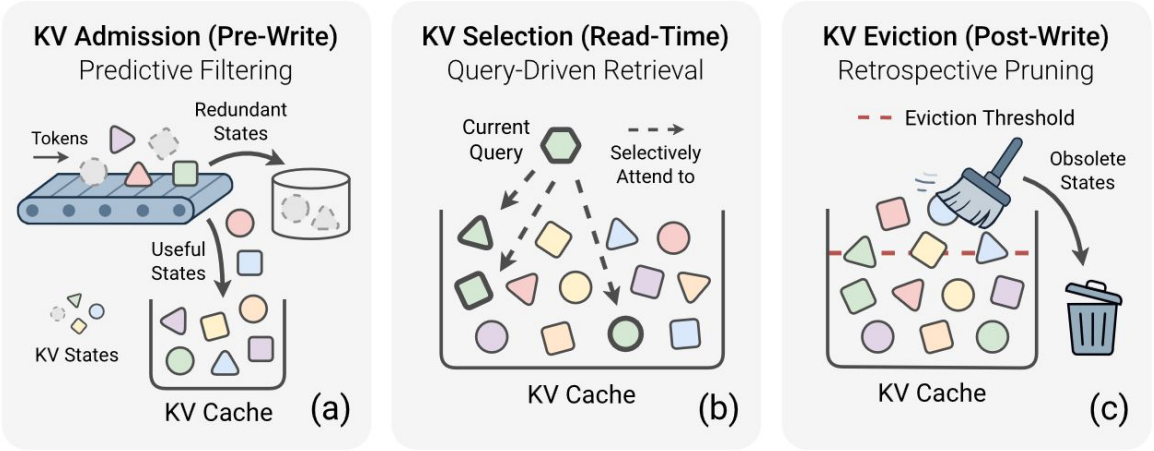
52 GB
of memory is spent on KV cache — over 3× the 16 GB model

94%
of prefill time is spent on attention

80%
of decode time is spent on attention

* measured on Llama-3.1-8B on H200, at 400K tokens

Today's methods write every token first, then decide what to read or retire. Can the model learn which tokens are worth writing at all?



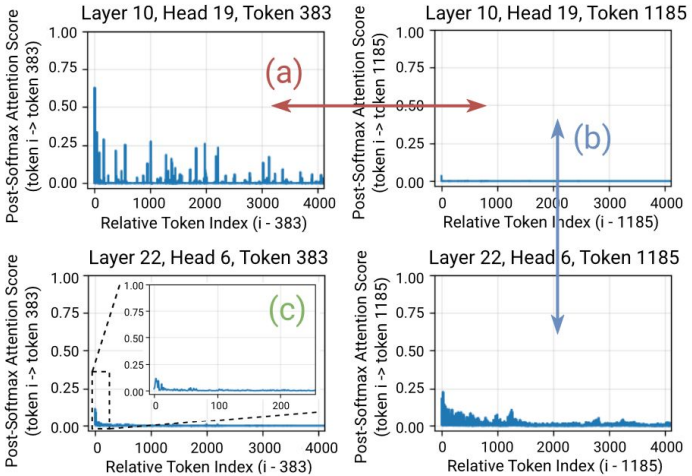
Our work — Prior work

Learning what to write

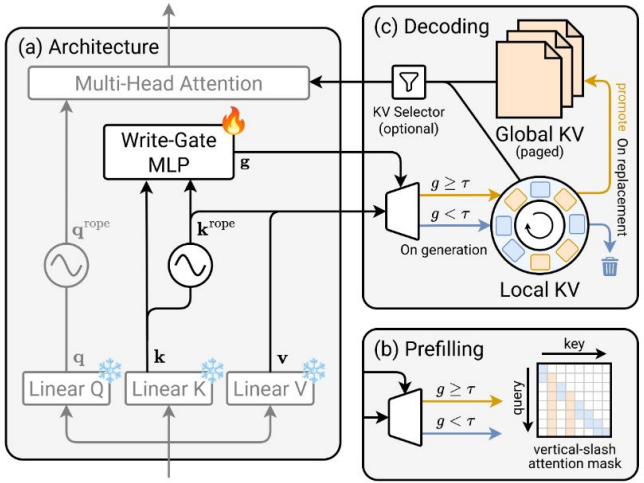
(a) Skewed utility
A few tokens draw most attention → filter the rest

(b) Head-specific
Heads value different tokens → decide per KV head

(c) Local recency
Recent tokens matter briefly → keep them for a while

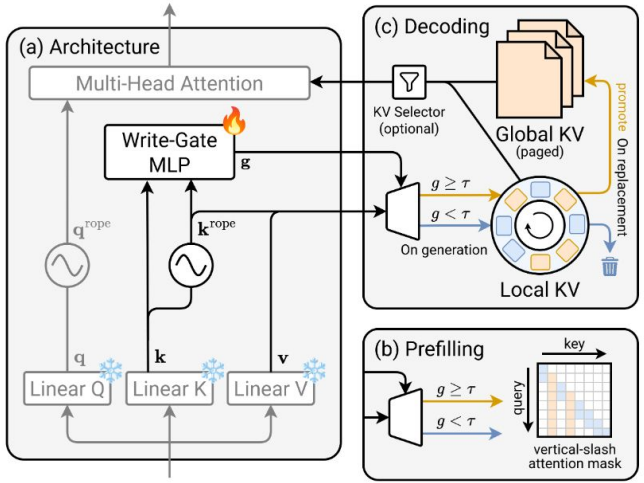


Heterogeneity of token utility



Overview of Write-Gated KV (WG-KV)

Learning what to write (details)



Overview of Write-Gated KV (WG-KV)

- Admission is a **hard 0/1 decision** at inference — hence non-differentiable. We relax it into a **continuous re-weighting of attention**, so gradients can flow:

$$\widetilde{\mathrm{Attn}}(q_i, K_{\le i}, V_{\le i}) = \sum_{j \le i} \frac{\exp(q_i \cdot k_j/\sqrt{d}) \cdot \boldsymbol{m_{ij}}}{\sum_{p \le i} \exp(q_i \cdot k_p/\sqrt{d}) \cdot \boldsymbol{m_{ip}}} v_j,$$
$$\text{where } m_{ij} = \max(\mathbb{1}(i - j < W_{\text{local}}), \boldsymbol{g_j})$$

- Fused attention kernels (FlashAttention) are optimized only for exp(*q*·*k*). To stay kernel-compatible, we fold the gate into an **additive log-space bias**:

$$\exp\left(q_i \cdot k_j/\sqrt{d}\right) \cdot \boldsymbol{m_{ij}} = \exp\left(q_i \cdot k_j/\sqrt{d} + \boldsymbol{\log(m_{ij})}\right)$$
$$\Rightarrow \widetilde{\mathrm{Attn}}(q_i, K_{\le i}, V_{\le i}) = \mathrm{Softmax}\left(q_i K_{\le i}^{\top}/\sqrt{d} + \boldsymbol{B_{\text{gate}}}\right) V_{\le i}$$

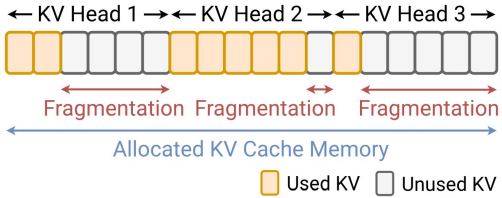
- We then train the gate to **match full attention** while keeping its values **sparse** and **near-binary**:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{distill}} + \lambda \mathcal{L}_{\text{sparsity}},$$
$$\text{where } \mathcal{L}_{\text{sparsity}} = \frac{1}{L \cdot H \cdot T} \sum_{l,h,t} \left(g_{l,h,t} + g_{l,h,t}(1 - g_{l,h,t})\right)$$

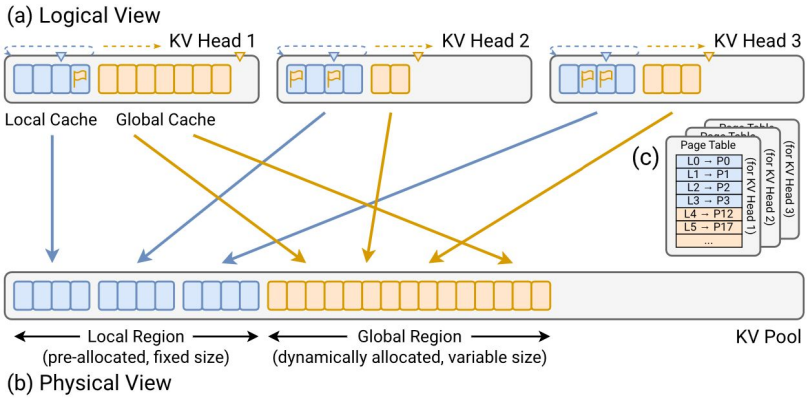
System design: turning sparsity into real speedups

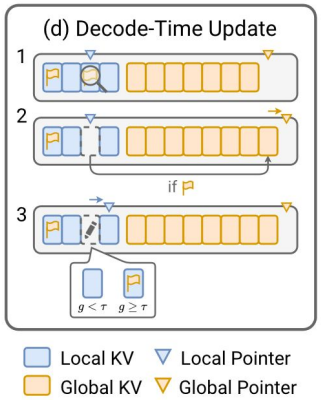
CHALLENGE

Per-head admission makes the cache ragged



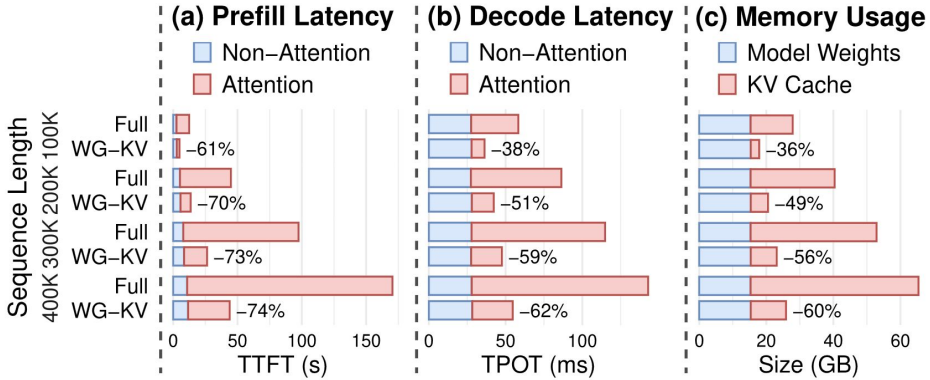
Contiguous buffers would fragment memory or force costly reallocation





1	**MEMORY**	Paged dual cache: **local** + **global** caches of all heads share one KV pool
2	**PREFILL**	Vertical-slash sparse attention skips **non-admitted** tokens
3	**DECODE**	Triton-based variable-length paged attention kernels for **ragged** heads

End-to-end speedups and memory savings



End-to-end latency and memory usage on Llama-3.1-8B

	Llama-3.1-8B	Qwen3-4B-2507
Prefill	2.56–3.85×	3.23–4.17×
Decode	1.61–2.63×	1.59–2.50×
Memory	-36–60%	-53–69%

≈80% sparsity (20% of KVs admitted); 100K–400K-token PG-19 inputs on a single H200. Timings include all runtime overheads introduced by WG-KV.

For more details, see our paper and code.

DEEP DIVE 3 / 3

TinyML on microcontrollers

Fitting accurate multi-branch networks into kilobytes of SRAM, proven on real MCUs

DupNAS · TCAD 2026 · toolchain development

On microcontrollers, peak memory decides which networks can run

Microcontrollers (MCUs) run **bare-metal**: no OS, no virtual memory to fall back on.
So a network runs only if all tensors still in use, plus the engine, fit in **on-chip SRAM**: 320 KB on an STM32F746.

1 For accuracy, networks went multi-branch

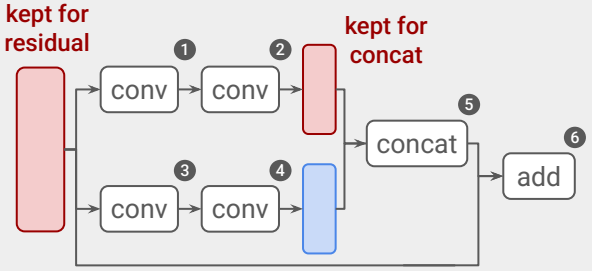
+2–4%

accuracy at similar compute

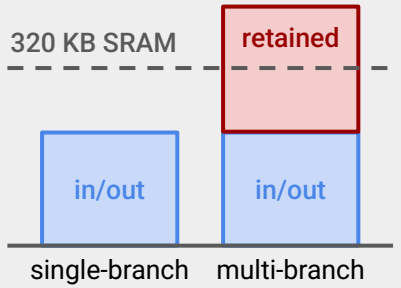
ShuffleNetV2 (multi-branch) vs.
MobileNetV2 (single-branch)

2 But branches inherently raise peak memory

After ❹, before ❺:



3 So the peak overflows the SRAM

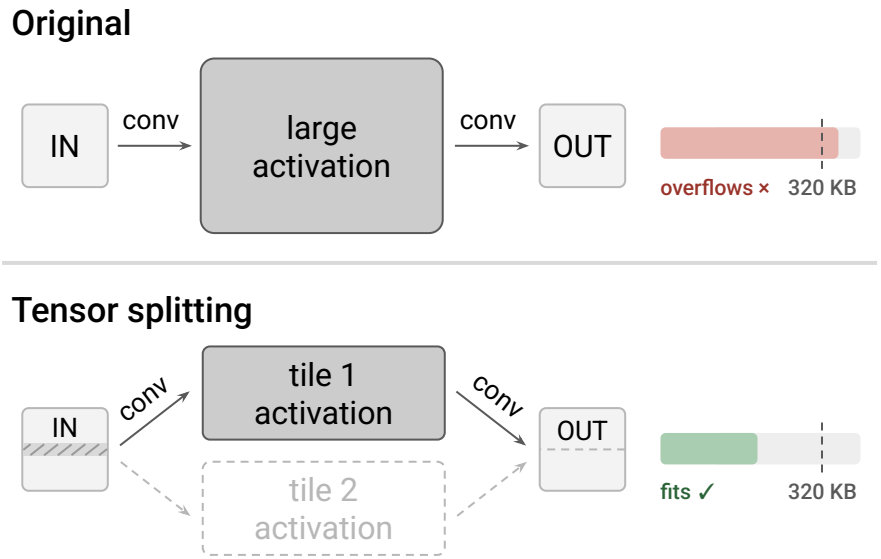


→ the accurate network can't run

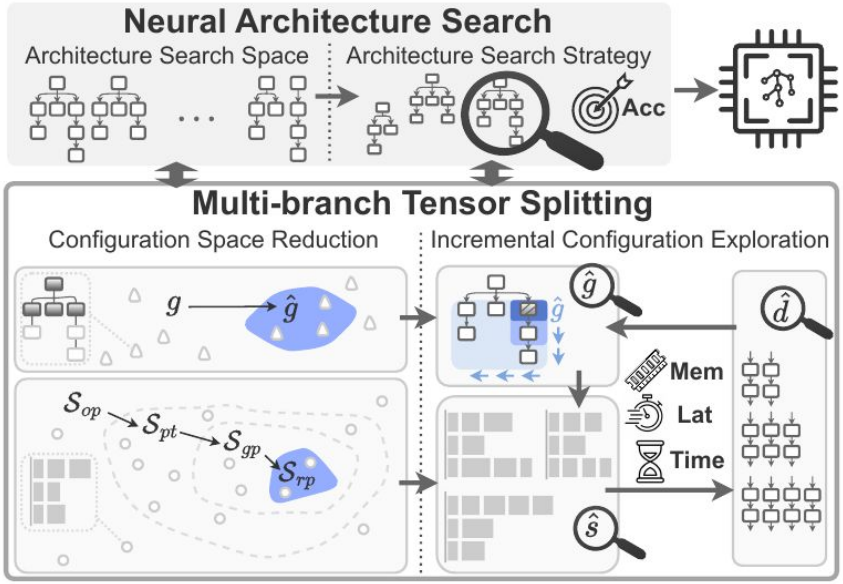
DupNAS: reduce peak memory while keeping the branches, and prove it on real MCUs

Tensor splitting, searched together with the network

algorithm by co-authors



Tensor splitting: one tile in memory at a time



DupNAS: find the most accurate network that fits

Liu, Mendis, Huang, Chen, Hsiu - Splitting Bottlenecks: Memory-Aware Neural Architecture Search for Multi-Branch TinyML [TCAD 2026]

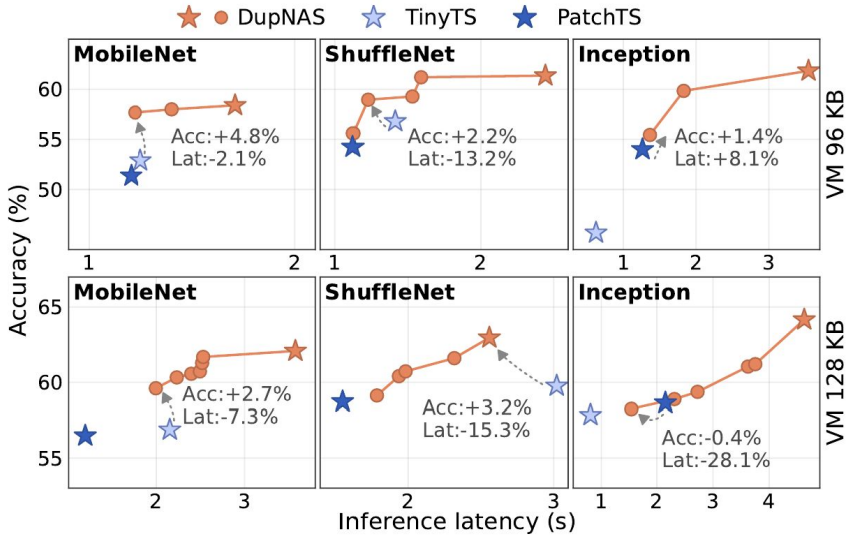
From split config to on-device deployment

my contribution

Split config which region, how many tiles, what order (from co-authors)

1	**REWRITE**	Turn the split config into a tile-by-tile ONNX graph	PyTorch · ONNX GraphSurgeon
2	**VERIFY**	Check the outputs match the original model	ONNX Runtime
3	**QUANTIZE**	Convert to INT8-quantized model	onnx2tf · TFLite
4	**BUILD**	Compile the model for 2 engines × 2 MCUs	TFLite Micro · STM32Cube.AI · make
5	**MEASURE**	Scripted build → flash → report latency and memory	STM32CubeIDE · Bash

Measured on STM32: better trade-offs, portable across engines



Accuracy vs. latency measured on STM32F746 using TFLite Micro

	vs. best baseline*
Accuracy	-0.4 to +4.8 pp
Latency	-28% to +8% (avg ≈ 10%)
Portability	29 networks on 2 engines × 2 MCUs

* per network family & memory budget

For more details, see our paper and code.

Ready to build next-generation compute systems

Bridging research-grade rigor with proven execution across architecture modeling, FPGA/ASIC flows, and ML serving systems.

Thank you – questions welcome

✉ r13921038@ntu.edu.tw · 🌐 lashhw.github.io