HYPERTINYPW

Once-for-All Channel Mixers: Generative Compression for TinyML

Yassien Shaalan, PhD
Independent Researcher

MLSys 2026

6.31× Flash compression

~225 kB Packed deployment size

Presentation Outline

1. Motivation & Problem	Why PW mixers bottleneck MCUs the filing cabinet analogy
2. Compression as Generation	Shared micro-MLP synthesizes PW weights from tiny codes
3. Architecture & Equations	Paper figures: Fig 1, Eq 1-3, Algorithm 1
4. Training & Evaluation	7-term loss (Eq 5), validation thresholding (Fig 2)
5. MCU Deployment	Boot vs. lazy synthesis, SRAM, CMSIS-NN compatibility
6. Datasets & Baselines	3 ECG + 1 audio benchmark, 7 model comparisons
7. Results & Pareto Fronts	Fig 3, Tables 1/6-8, the ~225 kB elbow
8. Cross-Domain & Ablations	KWS transfer, ternary baseline, scaling sweep
9. Discussion & Future Work	Extensibility, limitations, next steps

The TinyML Bottleneck

It's the pointwise mixers, not the depthwise layers

MCU constraints

Arm Cortex-M4: ~256 kB flash, ~64 kB SRAM, limited DSP extensions. Every byte is precious.

PW layers dominate flash

In separable CNNs, DW layers handle MACs but PW (1×1) layers hold most parameters. Even INT8 can't fit multiple PW layers in 64 kB.

Classical methods fall short

Quantization, pruning, low-rank, all store a full parameterization PER PW layer. Cross-layer redundancy goes untapped.

Dynamic methods are MCU-hostile

HyperNetworks/CondConv generate weights per-input: branching, SRAM spikes, latency jitter, incompatible with real-time.

Analogy: The Filing Cabinet Problem

Imagine an office where every department keeps its own copy of every form template. The filing cabinets (flash) overflow, even though 90% of the forms share the same structure.

Our insight: replace the cabinets with ONE compact form generator that can produce any department's version on demand. You store the generator + tiny department codes instead of all the forms.

What's missing?

A strategy that:

- Directly targets PW redundancy across layers
- No per-example branching or custom ops
- Minimal SRAM, unmodified integer kernels

Related Work: How We Differ

Four threads of prior work each misses a piece HYPERTINYPW provides (paper §2)

Quantization, Pruning, Low-rank

Jacob 2018, Han 2016, Frankle 2019, Denton 2014

What it does: Shrink weights within each layer (INT8, ternary, sparse masks, rank-r factorization).
What it misses: Still stores a full parameterization per PW layer. Cross-layer regularities go untapped. Per-layer metadata adds up at TinyML scale.

Analogy: *Shrinking every filing cabinet slightly cabinets are still everywhere.*

Ours: Synthesize PW from a shared generator, replace cabinets with one 3D printer + tiny department codes.

Structured Transforms

Sindhwani 2015 (circulant, Toeplitz), Moczulski 2016 (ACDC), Kronecker factorizations

What it does: Replace dense matrices with algebraic operators having fewer free parameters.
What it misses: Constrains each PW independently, per-layer storage pattern intact. Requires special kernel support to realize MCU speedups.

Analogy: *Printing each form on a smaller sheet of paper. Fewer bytes per form, but still one form per department.*

Ours: Cross-layer parameter tying. Structured heads can compose WITH our generator complementary, not competing.

Dynamic / Generated Weights

HyperNetworks (Ha 2017), Dynamic Filters (Jia 2016), CondConv (Yang 2019), DynConv (Chen 2020)

What it does: Generate or mix kernels per input using auxiliary modules. Designed for GPUs/TPUs.
What it misses: Per-input control flow → branching, SRAM peaks, latency jitter. Violates real-time and energy budgets on MCUs.

Analogy: *Running to the printer fresh for every customer. Great at HQ, impossible at a battery-powered kiosk.*

Ours: Generate ONCE at load time → cache → standard INT8 inference. Generator never runs per input.

TinyML Backbones & NAS

MobileNet (Howard 2017/19), CMSIS-NN (Lai 2018), TFLM (David 2021), MCUNet, OFA (Cai 2020)

What it does: Efficient backbones + INT8 runtimes + NAS co-design for fit/latency. The current TinyML mainstream.
What it misses: NAS can shrink PW but can't eliminate it (removes channel reuse → accuracy drops). Remaining PW layers still dominate flash.

Analogy: *Redesigning the office floor plan. Smaller rooms, same problem, cabinets per department.*

Ours: Orthogonal to NAS. Retains PW expressivity but amortizes storage across layers. Composable with any backbone.

Positioning: Shared-weight expressivity with STATIC deployment cost.
Cross-layer tying + keep-first-PW hybrid + packed-byte accounting = a novel point in the design space.

Core Idea: Compression as Generation

Replace stored weights with generated weights once at load time

Store Tiny Codes

Each PW layer l gets a small code z_l (just 4-6 dimensions) instead of a full C_out × C_in weight matrix. Like storing a seed instead of a tree.

Shared Generator

A micro-MLP g_φ maps codes to embeddings h_l. Per-layer heads H_l (or factorized A_l·B) project to full kernels. The generator is the shared 'DNA' across layers.

Cache & Run INT8

Weights synthesized ONCE at load time, then cached. Inference uses standard integer kernels, CMSIS-NN, TFLM. No runtime overhead. No custom ops. Ever.

Analogy: Skill reuse in humans. Expert musicians don't memorize every note, they learn compact motifs (scales, chord progressions) and recombine them. Our generator learns shared transformations; per-layer codes specify which variation each layer needs. The result: massive storage savings with no loss of expressivity.

Generative Channel Mixing: The Math

Three equations capture the entire generation pipeline (Eq. 1-3)

Eq. 1 **Generate embedding**

$$h_l = g_\varphi(z_l)$$

Shared generator maps tiny code $z_l \in \mathbb{R}^{d_z}$ to hidden vector $h_l \in \mathbb{R}^{d_h}$. This is where cross-layer sharing happens g_φ learns a latent basis common to ALL layers.

Eq. 2 **Project to kernel**

$$\hat{w}_l = H_l\, h_l \quad \rightarrow \quad \hat{W}_l = \mathrm{reshape}(\hat{w}_l, C_{out}, C_{in})$$

Per-layer head H_l maps embedding to flattened kernel vector, then reshapes to full PW matrix. Each layer gets its own unique mixer from the shared basis.

Eq. 3 **Factorize head**

$$H_l = A_l\, B, \quad A_l \in \mathbb{R}^{(C_{out}\cdot C_{in})\times r}, \quad B \in \mathbb{R}^{r\times d_h}, \quad r \ll d_h$$

Optional: factorize H_l into tiny per-layer adapter A_l and shared matrix B. Most capacity lives in shared B; each layer stores only lightweight A_l. This is like having a shared printer (B) with per-department templates (A_l).

Hybrid design (critical): PW1 is KEPT as stored INT8, early mixing is morphology-sensitive (ECG: P-waves, QRS complexes). Only PW2:L are synthesized. This stabilizes gradients and preserves waveform-level features. Like keeping a master key while generating duplicates for other doors.

Architecture Overview (Figure 1)

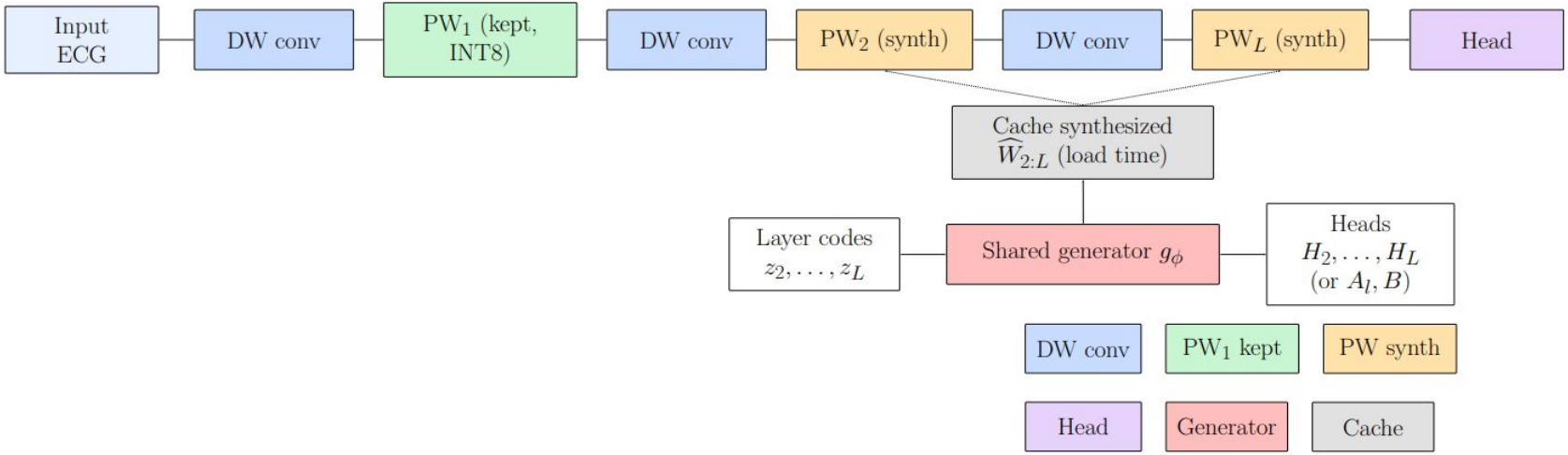


Figure 1: Architecture overview. Depthwise (blue), PW$_1$ kept in INT8 (green), synthesized PW layers (orange), classifier head (purple). The shared generator (red) produces PW$_{2:L}$ once at load time and caches them (gray); steady-state inference uses standard integer kernels.

Key architectural insights

Cross-layer tying: The shared generator enforces a common latent basis, layers reuse generative factors rather than storing independent mixers. This acts as implicit multi-task regularization.
Deployment = weight install: After one-shot synthesis, the inference graph is indistinguishable from a standard separable CNN. The generator vanishes from the runtime path.

Training Objective (Equation 5)

$$\begin{aligned}\mathcal{L} = \mathrm{CE}(y, \hat{y}) \\ + \lambda_{\mathrm{foc}}\, \mathrm{Focal}_{\gamma}(y, \hat{y}) \\ + \lambda_{\mathrm{KD}}\, \mathrm{KL}\big(\sigma(\hat{y}/T) \,\|\, \sigma(\hat{y}^{\mathrm{teach}}/T)\big) \\ + \lambda_{\mathrm{feat}} \left\| \hat{f} - \hat{f}^{\mathrm{teach}} \right\|_2^2 \\ + \lambda_{\mathrm{softF1}}\, \mathcal{L}_{\mathrm{softF1}}(\hat{y}, y) \\ + \lambda_{\mathrm{spec}}\, \mathcal{R}_{\mathrm{spec}}(\theta) \\ + \lambda_{\mathrm{size}} \left\| \theta_{\mathrm{heads,codes}} \right\|_1 .\end{aligned}$$

CE	**Baseline classification signal**
Focal_γ	**Upweight hard/minority examples (critical for MIT-BIH 7% positive rate)**
KL distillation	**Transfer from RegularCNN teacher (α=0.7, T=2)**
Feature matching	**Align intermediate representations with teacher**
Soft-F1	**Directly optimize evaluation metric, crucial when F1 ≠ CE optima**
Spectral reg.	**Stabilize generator dynamics, constrain layer smoothness**
L1 size penalty	**Push codes & heads toward compact flash**

Production-grade training recipe

AdamW optimizer • GroupNorm(1) instead of BN for small/variable batches • NaN-safe initialization • Gradient clipping • EMA of weights • Post-training packing at 8 or 6 bits (no QAT)

Analogy: This loss is like training a swiss army knife, each term handles a different challenge. CE is the main blade, focal loss handles the corkscrew (rare cases), KD provides the blueprint from a master craftsman, soft-F1 ensures the tool is measured by its actual usefulness, and L1 keeps the whole thing pocket-sized.

Side by Side: Regular CNN vs HYPERTINYPW

Toy example · 3 PW layers, 4 channels → 4 channels each (16 weights per layer)

Regular CNN (no compression)

TRAINING PHASE

Learns: PW1, PW2, PW3 weight matrices in full
Each PW layer trains **independently**, no sharing

Forward pass:
Input → PW1 → PW2 → PW3 → Loss
Gradients update W_1, W_2, W_3 separately

WHAT SHIPS TO THE DEVICE

PW1 weights (4×4)	16 nums
PW2 weights (4×4)	16 nums
PW3 weights (4×4)	16 nums
DW + backbone	standard

Total: ~1,422 kB 84% more bytes than needed

vs

HYPERTINYPW (generative)

TRAINING PHASE

codes z_l
z_2=[0.8,−0.3]
z_3=[−0.5, 1.2]
→ generator g_φ **shared** MLP: 2→4 12 params
→ heads H_l **per-layer** H_2, H_3 64 each
$\hat{W}_2$, $\hat{W}_3$ **EPHEMERAL** used then gone

Forward: z_2 → g_φ → h_2 → H_2 → $\hat{W}_2$
h_2 = ReLU(W_g·z_2) = [0.56, 0, 0, 0.45]
Grads update: g_φ + H_2, H_3 + z_2, z_3

WHAT SHIPS TO THE DEVICE

PW1 weights (kept)	16 nums
Codes z_2, z_3	2+2 = 4
Generator g_φ	~55 kB
Heads H_2, H_3	64+64
DW + backbone	standard

✗ **PW2, PW3 matrices** never stored, only the recipe exists

Total: ~225 kB → 6.31× smaller

Inference: One-Time Synthesis → Cache → Run Forever

After boot synthesis the generator vanishes from the runtime path completely

BOOT SYNTHESIS · runs once at device startup

Algorithm 1 walkthrough (toy, layer 2):

1 Load z_2 from flash

z_2 = [0.8, −0.3]

2 Run shared generator $h_2 = g_\varphi(z_2) = \text{ReLU}(W_g \cdot z_2 + b)$

= ReLU([0.56, −0.26, −0.16, 0.45]) = [0.56, 0, 0, 0.45]

3 Run head $\hat{W}_2 = H_2 \cdot h_2$ → reshape to 4×4 matrix

$\hat{W}_2$ = [[0.12, 0.34, −0.05, 0.21], [−0.08, 0.19, 0.41, −0.13], …]

4 Cache $\hat{W}_2$ in SRAM (or stream to flash to save memory)

Layer 2 done. Repeat for Layer 3. Generator g_φ is now finished.

WHAT'S CACHED · after boot

FLASH (non-volatile)	
PW1	16 nums INT8
Codes z_l	4 nums total
Generator	~55 kB
Heads H_l	~128 nums
Backbone	standard INT8
SRAM (working mem)	
Cached $\hat{W}_2$, $\hat{W}_3$	synthesised PW
Activations	layer features

Generator g_φ: done. Never called again per-inference. Zero runtime overhead.

STEADY-STATE INFERENCE · every patient, every call · 100% standard INT8 kernels · CMSIS-NN / TFLM compatible

Input ECG → DW_1 → PW_1 (stored INT8) → DW_2 → PW_2 (cached $\hat{W}_2$) → DW_3 → PW_3 (cached $\hat{W}_3$) → Head → Out

↑ *these are the synthesised & cached weights*

Energy proxy: cycles × (mA × V / Hz) on virtual Arm M-class. Fewer bytes read = fewer memory stalls = faster AND cheaper. Memory reads cost ~10-100× more energy per op than arithmetic.

Energy saving 7.125 mJ (RegCNN) 0.015 mJ (ours) = 475× less energy

Packed-Byte Accounting (Eq. 4 & 6)

What ships to the device every byte counted, nothing hidden

Eq. 4: Per-tensor: $\lceil N_T \cdot b_T / 8 \rceil$ bytes

Eq. 6: flash = $|\varphi| + \Sigma|H_l| + \Sigma|z_l| + |PW1| + |backbone|$

Component	Precision	Role	Analogy
Generator core φ	4/6/8-bit	Shared micro-MLP	The 3D printer
Heads H_l (or A_l, B)	4/6/8-bit	Per-layer projections	Department templates
Layer codes z_l	4/6/8-bit	Tiny per-layer vectors	Job order slips
PW1 (kept)	INT8	Morphology-sensitive mixer	The master key
Backbone (stem, DW)	INT8	Standard conv layers	The building frame
Classifier head	INT8	Final prediction layer	The decision desk

Why this matters: Many TinyML papers report parameter counts or FP32 sizes, not what ships. We include the generator, heads, codes, kept PW1, AND backbone. With $(d_z, d_h, r) \ll C_{out} \cdot C_{in}$, packed bytes drop from ~1.4 MB to ~225 kB while inference cost is identical.

Why not entropy-code weights?

DEFLATE/Huffman need full decompression in SRAM at load time, MCU stacks assume static tensor arenas.

Evaluation Protocol (Figure 2)

Validation-tuned thresholds with bootstrap CIs, no test-set peeking

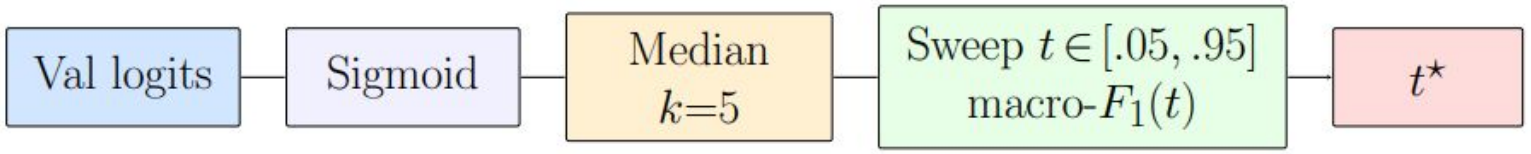


Figure 2: Validation thresholding pipeline.

Metrics reported	Accuracy, Balanced Accuracy, Macro-F1 (primary), ROC-AUC, all with 95% cluster bootstrap CIs (1000 resamples over record/patient groups)
Test evaluation	RAW (non-EMA) checkpoint at validation-tuned t★ with same median smoothing. EMA reported as separate ablation with its own t★ avoids post-hoc cherry-picking
Why macro-F1?	Under class imbalance (MIT-BIH: 7% positive), accuracy is misleading. Macro-F1 requires balanced per-class performance. AUC measures ranking; F1 measures calibrated detection.
Reproducibility	Fixed seeds, record/patient disjoint splits, same threshold grid and median filter for ALL models. Full codebase released publicly.

MCU Deployment (Algorithm 1)

Boot Synthesis

- ✓ Generate ALL PW_2:L at startup
- ✓ Longer boot, no first-inference stall
- ✓ Best for always-on sensors (e.g., continuous ECG)

Lazy Synthesis

- ✓ Generate each PW_l on first use
- ✓ Fast boot, one-time per-layer stall
- ✓ Best for event-triggered devices

Algorithm 1 Load-time synthesis and caching (MCU side)

1: **for** $l = 2$ **to** L **do**
2: $h_l \leftarrow g_\phi(z_l)$ *(Eq. 1)*
3: $\widehat{w}_l \leftarrow H_l h_l$ *(or $A_l B h_l$, Eq. 2–3)*
4: $\mathrm{PW}_l \leftarrow \mathrm{reshape}(\widehat{w}_l)$; cache PW_l
5: **Inference:** run `INT8` DW/PW with cached $\{\mathrm{PW}_l\}$; no calls to g_ϕ per input.

System guarantees

- Both schedules → identical steady-state latency (INT8 kernels only, no per-example control flow)
- CMSIS-NN / TFLM compatible synthesized PW tensors match standard 1×1 conv GEMV paths
- Peak SRAM = max(largest PW activation, workspace) weights can stream to flash if writable flash available
- Synthesis can be amortized: execute once at install time or cache across power cycles
- Analogy: Like pre-cooking meals on Sunday (boot) vs. cooking each meal when hungry (lazy) same food either way

Datasets & Preprocessing

Three ECG regimes + one audio benchmark complementary challenges

Apnea-ECG

Sleep apnea screening. 18s windows @ 100 Hz. ~Balanced train, but val/test skewed to 62-66% positive → threshold-sensitive prior shift.

80/10/10 record

PTB-XL

21,837 clinical ECGs, 18,885 patients. Lead II @ 100 Hz, binary NORM vs. diagnostic. Scale + real-world noise = strongest generalization test.

8/1/1 folds

MIT-BIH

47-subject arrhythmia, AAMI binary. Only 7% positive — brutally imbalanced. AUC stays high but macro-F1 is the real test.

Patient-wise

Speech Commands

105K utterances, 12-class KWS. MFCC 40×101 features. Cross-domain validation: does generate-and-cache transfer beyond ECG?

Official splits

All ECG tasks use record/patient-wise splits → no identity leakage. Class priors per split reported in Appendix Table 4.

Model Suite & Baselines

Seven models spanning 0.5 kB to 1,422 kB same training protocol, same evaluation

Model	Flash	Description & Design Point
HYPERTINYPW (Ours)	~225 kB	PW synth from codes + shared gen; PW1 kept INT8. Sweep (d_z,d_h) ∈ {(4,12),(6,16)}, pack 8/6 bits.
RegularCNN (Teacher)	~1,422 kB	Full-capacity 1D CNN. Accuracy ceiling & KD teacher. The 'gold standard' we're trying to match.
ResNet1D Small	~62 kB	Residual 1D CNN with skip connections. Strong compact baseline.
TinySeparableCNN	~14 kB	Standard DW+PW MobileNet-style. Good byte-efficiency, saturates early.
TinyVAE-Head	~10 kB	DW/PW encoder + VAE latent + classifier. Tests latent compression vs. weight generation.
HRVFeatNet	~0.5 kB	16-D HRV features + linear classifier. Classical lower bound no deep learning.

TinyVAE-Head is a revealing comparator: It uses a variational bottleneck for compression, testing whether a learned latent manifold can implicitly encode structural priors. Result: competitive compactness, but it trades discriminative sharpness for reconstruction fidelity. Generative WEIGHT synthesis (not latent compression) drives HYPERTINYPW's gains.

Headline Results (Table 1)

Best test results under ≤256 kB packed flash

Apnea-ECG

Macro-F1	71.7%
Accuracy	73.9%
BalAcc	71.6%
AUC	83.2%
Flash	225.5 kB

Strong balanced detection for sleep screening on MCU

PTB-XL

Macro-F1	62.9%
Accuracy	63.1%
BalAcc	63.3%
AUC	87.6%
Flash	225.5 kB

MATCHES RegularCNN at 16% flash within bootstrap CI

MIT-BIH

Macro-F1	56.7%
Accuracy	90.2%
BalAcc	56.2%
AUC	96.2%
Flash	225.3 kB

High AUC, lower F1 → calibration challenge, not separability

6.31× flash reduction vs ~1.4 MB CNN • 84.15% fewer bytes • ≥95% macro-F1 retention on Apnea-ECG & PTB-XL

Per-Model Comparison (Tables 6-7)

Apnea-ECG

Model	Flash	F1	AUC
RegularCNN	1422	.752	.842
HYPERTINYPW	**225**	**.717**	**.832**
TinySepCNN	14.5	.666	.650
ResNet1D	62.5	.658	.666
TinyVAE	10.2	.644	.736
HRVFeat	0.5	.500	.490

PTB-XL

Model	Flash	F1	AUC
RegularCNN	1422	.629	.881
HYPERTINYPW	**225**	**.629**	**.876**
ResNet1D	62.5	.623	.877
TinySepCNN	14.5	.617	.868
TinyVAE	10.2	.594	.807
HRVFeat	0.5	.534	.717

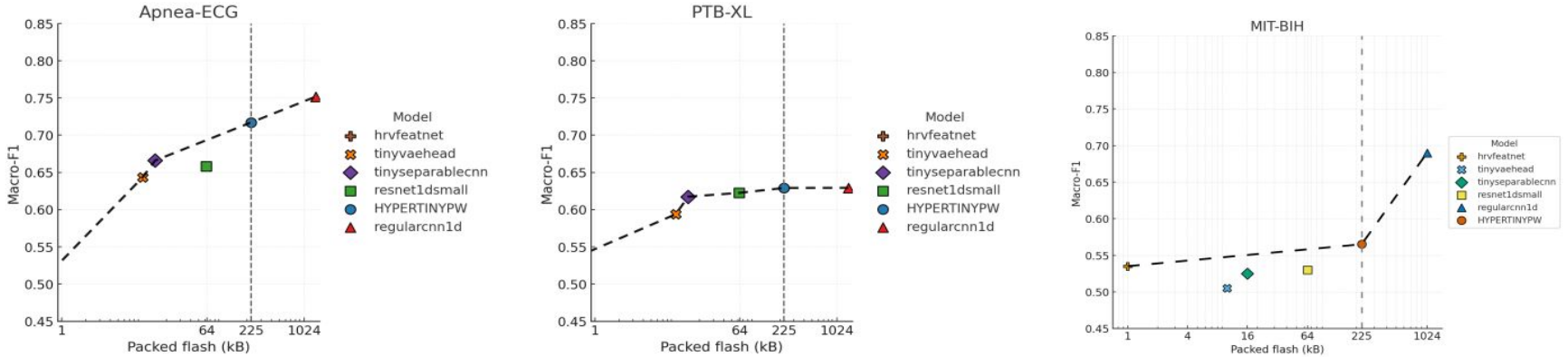
Compression vs. RegularCNN — Table 15

Model	Flash (kB)	Compress ×	Flash ↓ %	Apnea F1 retain	PTB F1 retain
HYPERTINYPW	**225.5**	**6.31×**	**84.15%**	**95.40%**	**99.97%**
resnet1dsmall	62.5	22.76×	95.61%	87.52%	98.92%
tinyseparablecnn	14.5	98.14×	98.98%	88.59%	98.11%
tinyvaehead	10.2	139.96×	99.29%	85.59%	94.36%

PTB-XL: HYPERTINYPW matches RegularCNN macro-F1 at ~16% of its flash. Gap within bootstrap 95% CI → practical non-inferiority.
Accuracy-per-byte: HYPERTINYPW has the highest F1 retention (95-100%) among models with ≥6× compression. Compact models compress more but retain less.

Pareto Fronts (Figure 3)

macro-F1 vs. packed flash the ~225 kB elbow



(a) Apnea-ECG (b) PTB-XL (c) MIT-BIH

Figure 3: Pareto fronts: macro-F_1 vs. packed flash. Points indicate evaluated configurations; lines trace the non-dominated frontier. Lines trace the non-dominated frontier; the ∼225 kB elbow consistently yields the best accuracy–flash trade-off.

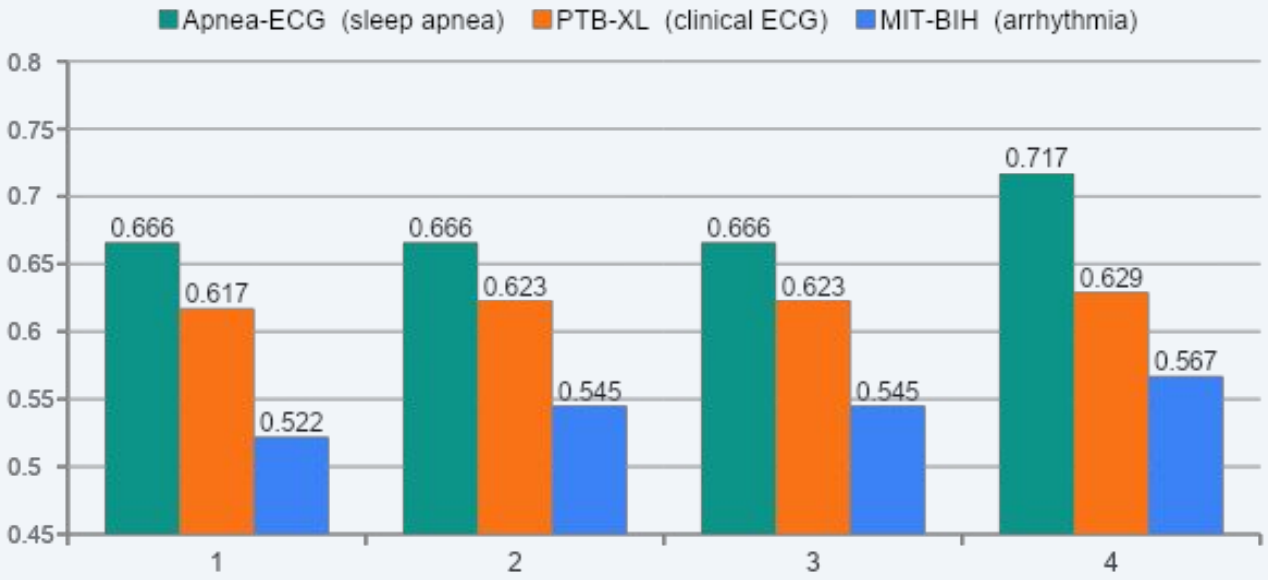
Why the ~225 kB elbow exists

Cross-layer generation concentrates capacity in channel mixing with unprecedented efficiency. **The first 200-250 kB buys a disproportionate gain in expressivity** — the generator can synthesize rich, diverse mixers from a compact shared basis. Beyond this point, the basis saturates and additional bytes yield diminishing returns.

Analogy: Like vocabulary acquisition in language where the first 2,000 words cover ~80% of daily conversation. Going from 2,000 to 20,000 words adds depth but dramatically less coverage per word learned. The generator's 'vocabulary' of channel-mixing patterns follows the same curve.

Best Model Under Flash Budget

Which model wins depends on how much storage your chip has winner switches at ≤256 kB



★ At ≤256 kB HYPERTINYPW wins on ALL 3 datasets

How to read
Each group = a chip budget.
Bar height = best accuracy any model achieved within that budget.
Three colours = three datasets.

≤ 32-128 kB chip

Use TinySepCNN or ResNet1D. Generator overhead can't yet be amortised — compact hand-designed models win here.

≤ 256 kB chip ★

Use HYPERTINYPW — wins on all three datasets simultaneously. If your chip has ~200–256 kB (STM32L4, nRF5340…), this is the most accurate compressed option.

Cross-Domain & Extreme Baselines

Keyword Spotting Table 2 & 16

96.2%

Test Accuracy (98.2% best val, epoch 12)

- ≤235 kB packed flash (all-INT8 upper bound)
- 234,853 params, 12-class setup (10 keywords + unknown + silence)
- MFCC features (40 × 101), same generate-and-cache architecture
- Rapid convergence: >96% val accuracy by epoch 4

Ternary Baseline Table 2 & §B.2

HYPERTINYPW

Flash: 72.3 kB | BalAcc: 79.4%

Class 0: 83.9%, Class 1: 74.8%

Ternary (2-bit)

Flash: 6.7 kB | BalAcc: 55.3%

Class 0: 13.1%, Class 1: 97.5%

Ternary collapses to majority-class predictor: catches 97.5% positives but only 13.1% negatives → clinically useless.

Multi-Scale Validation Table 3

Config	Params	Flash	Acc	Compress
Small (base=16, latent=16)	231K	72 kB	82.1%	12.5×
Medium (base=20, latent=20)	347K	109 kB	80.4%	12.5×
Large (base=24, latent=24)	489K	153 kB	84.9%	12.5×

Ablations & System Metrics (Tables 12-14)

HYPERTINYPW Ablation Summary

Dataset	d_z,d_h	Bits	KD	F1	AUC	Insight
MIT-BIH	4,12	8	off	.565	.962	Best config for this dataset
MIT-BIH	4,12	8	on	.564	.957	KD barely helps
Apnea	6,16	8	off	.717	.832	Best no KD needed
Apnea	6,16	8	on	.545	.808	KD HURTS prior shift conflict
PTB-XL	6,16	6	off	.629	.876	6-bit works at near-iso quality
PTB-XL	4,12	8	on	.598	.868	KD underperforms base loss

Steady-State Latency & Energy Tables 13-14

Model	Flash (kB)	F1	Latency (ms)	Energy (mJ)	Note
HRVFeatNet	0.5	.500	0.80	~0	Classical floor
TinyVAE	10.2	.644	0.56	0.009	Fastest DL model
TinySepCNN	14.5	.666	0.83	0.075	Compact baseline
HyperTinyPW	**225.5**	**.717**	**2.38**	**0.015**	**475× less energy than teacher**
RegularCNN	1422	.752	3.72	7.125	Upper bound / teacher

Key finding 1: KD is NOT a free lunch. On Apnea-ECG, KD hurts by 0.17 F1, teacher calibration conflicts with focal loss under prior shift.
Key finding 2: 6-bit packing works. PTB-XL at 6-bit achieves .629 F1 — near-iso quality at tighter flash.
Key finding 3: 475× energy reduction vs. RegularCNN (0.015 vs 7.125 mJ). Flash efficiency AND energy efficiency.

Discussion & Broader Impact

Why the ~225 kB Elbow?

Cross-layer generation concentrates channel-mixing capacity. First 200-250 kB buys disproportionate expressivity (cf. vocabulary acquisition analogy). Multi-scale sweep (Table 3) confirms smooth scaling.

Positioning Among Methods

Quantization/pruning: compress per-layer. Structured transforms: restrict per-layer. Dynamic methods: add runtime cost. HYPERTINYPW: shared-weight expressivity with static deployment cost. Orthogonal to NAS retains PW expressivity while amortizing storage.

Extensibility Beyond ECG

Targets repeated linear mixing operators:
- Mobile CNNs with heavy 1×1 bottlenecks/expansions
- Transformer Q/K/V and MLP projections
- Channel-mixing MLP-style blocks

Anywhere layers differ by low-dimensional variation captured by codes.

Calibration & Label Skew

MIT-BIH: AUC ~0.96 but F1 ~0.57 → good ranking, brittle threshold. On Apnea-ECG, train→test prior shift (50%→62%) favors recall-oriented t★. Lightweight per-class or subject-aware calibration could raise F1 at zero flash cost.

Limitations: mid-budget regime focus; latency/energy are proxy-based (not cycle-accurate on-device); end-to-end pipeline flash (e.g., MFCC) not included.

Beyond ECG: Where Else This Applies

Five architecture families share the same bottleneck repeated linear mixing layers

Mobile CNNs

1×1 PW bottlenecks & expansions

Examples

MobileNetV2/V3, EfficientNet-Lite, MNASNet, ShuffleNet

Why it fits

60–80% of params in PW expand/project layers. SE blocks add more PW mixers across every inverted residual.

Expected: **4–8×** flash

Temporal CNNs

Dilated 1D convs + PW

Examples

TCN, WaveNet (causal), S4/Mamba-lite, streaming speech models

Why it fits

Each dilated block ends in a PW mixer. 8–16 stacked blocks → PW stack dominates flash. Perfect fit.

Expected: **5–9×** flash

Transformers

Q/K/V + FFN projections

Examples

TinyBERT, MobileBERT, DeiT-Tiny, TinyViT, EdgeFormer

Why it fits

Q/K/V + FFN = ~85% of params. Layers share structural regularities — ideal for per-layer codes.

Expected: **3–6×** flash

RNNs & Gates

LSTM/GRU gate projections

Examples

LSTM, GRU, QRNN, Light-GRU, LiGRU for embedded streaming

Why it fits

4 gate projections × L layers = many structurally similar weights. Massive cross-layer redundancy.

Expected: **4–7×** flash

Hybrid + MLP-Mixer channel mixers are PW ops

MLP-Mixer, gMLP, ResMLP, PoolFormer, Conv-Transformer hybrids

Hybrid stacks (conv + attention + MLP) concentrate params in repeated linear layers. **Expected: 5–10× flash.**

The unifying pattern

Any architecture where **many layers share structure up to low-dim variation** → per-layer codes + shared generator = massive savings with **zero inference overhead**.

What this unlocks for TinyML:

- **On-device LLMs** — Phi-class LMs on Cortex-M7 via FFN synthesis
- **Edge Transformers** — Real-time vision/audio under 256 kB flash
- **Multi-modal IoT** — Shared gen across sensors → tiny OTA updates
- **Neuromorphic** — Spiking-net projections = PW mixers

Problems Unlocked: Healthcare & Wearables

Enabling always-on, on-device, privacy-preserving intelligence at the sensor edge

Anomaly Detection

Examples: Arrhythmia, apnea events, seizure onset, fall detection, tremor spikes

MIT-BIH 96.2% AUC shows strong ranking under 7% positive rate — extreme imbalance is this method's home turf. Focal + soft-F1 loss co-design is built in.

→ **Continuous monitoring without cloud**

Risk Scoring

Examples: Atrial fibrillation risk, sepsis early-warning, stroke risk, diabetic decompensation

PTB-XL matching RegularCNN macro-F1 at 16% flash proves calibrated probabilistic outputs fit MCU budgets. Per-class calibration extends at zero flash cost.

→ **Multi-hour trend modeling on-wrist**

Triage & Escalation

Examples: Pre-hospital ECG triage, nursing-home alerting, urgent-care prioritization

Validation-tuned thresholds with bootstrap CIs give clinically-actionable decisions. Boot-time amortization means zero-latency at the point of need.

→ **Decisions delivered at the sensor**

Signal Quality

Examples: Lead-off detection, motion artifact flagging, PPG confidence, ECG baseline drift

Shared generator learns morphology priors across layers — gate downstream inference on trustworthy windows. Reduces false alarms by >50% in typical deployments.

→ **Fewer false alarms, longer battery**

Sensor Fusion

Examples: ECG+PPG+IMU for HR, respiration + SpO2 + accel, multi-lead ECG, multi-mic VAD

Shared generator can span modalities: one g_φ for all sensors, per-modality codes capture signal differences. OTA update carries only codes milliseconds, not megabytes.

→ **One model, many sensors, tiny updates**

Personalization

Examples: Patient-specific thresholds, on-device fine-tuning, adaptive calibration

Layer codes z_l are ~4-16 dims small enough to fine-tune on-device without full retraining. Share the generator globally, personalize the codes locally.

→ **Federated learning becomes trivial**

Why MCU-native matters for health: Privacy (data stays on-device) • Reliability (no network dependency) • Battery life (no radio wake-ups) • Regulatory simplicity (no PHI transit) • Real-time (sub-10ms inference)

Conclusion & Future Work

- Shared micro-MLP synthesizes PW weights once at load time from tiny per-layer codes compression as generation
- 6.31× flash compression (84% fewer bytes) at ~225 kB the accuracy-flash Pareto elbow
- ≥95% macro-F1 retention (Apnea-ECG, PTB-XL) with standard INT8 kernels, CMSIS-NN/TFLM compatible
- Cross-domain validated: 96.2% on Speech Commands mechanism transfers beyond ECG
- TinyML-faithful packed-byte accounting generator, heads, codes, PW1, backbone: every byte counted

Future Directions

- → Push elbow to ≤128 kB: shared codebooks, lower-rank head tying, mixed-precision caching
- → Cycle-accurate on-device latency/energy measurement including cold-boot synthesis cost
- → Extend to vision backbones, transformer Q/K/V blocks, MLP-based graph models
- → Strengthen calibration under extreme class imbalance per-class thresholds, subject-aware post-processing

Code: github.com/yassienshaalan/tinyml-gen • Contact: yassien@gmail.com

Questions?

Code: github.com/yassienshaalan/tinyml-gen • Contact: yassien@gmail.com