FLASHROCKET: ARCHIVE-SCALE TIME SERIES CLASSIFICATION IN SECONDS

Anonymous authors
Paper under double-blind review

ABSTRACT

As wearables and the Internet of Things proliferate, AI systems will increasingly need to process many different sensors at once and adapt quickly, ideally within seconds, as those sensors drift. One time series foundation model may seem like the natural answer, but it still needs a classification head trained for every task. In our experiments, training and testing these classifiers takes from 15 minutes to an estimated 7.5 hours, far too slow for quick adaptation. Training and testing hundreds of classifiers in seconds on one GPU is hard: the tasks differ by orders of magnitude in size, length and channel count, so most are too small to keep the GPU busy while a few are large enough to stall the rest. We present FlashRocket, a system that trains and tests classifiers on MiniRocket features at archive scale as one GPU workload. It extracts features in a single fused on-chip pass, tiling long series so that the longest few do not become the bottleneck, and fits a cheaper ridge classification head. On 188 datasets spanning four archives (UCR, UEA, Bake Off Redux and Multiverse-core), with $279{,}247$ training and $249{,}041$ test series, FlashRocket trains and tests all 188 classifiers, from raw series to predicted labels, in **3.07 seconds** on one L40S GPU with 16 CPUs. This is $242\times$ faster than MiniRocket in `tsai` at matching accuracy (83.07% against 83.10%), and $298\times$ faster and 1.34 points more accurate than MantisV2. FlashRocket makes it possible to retrain and test archive-scale collections of classifiers in seconds.

1 INTRODUCTION

With the rise of wearables (Piwek et al., 2016; Dunn et al., 2018) and the Internet of Things (Atzori et al., 2010), AI systems will increasingly need to process many different sensors at once. In health, one person's wearables, home devices and clinical monitors could one day span hundreds to thousands of sensors, each needing its own personalised classifier (Kiranyaz et al., 2016). Similar multi-sensor settings arise in manufacturing, where every machine has its own sensors and fault types (Lee et al., 2015; Susto et al., 2015). Ideally, such systems could run all of these classifiers cheaply and retrain them in parallel within seconds whenever their sensors drift (Gama et al., 2014).

A natural approach is one large model that transfers across tasks. This is the motivation behind time series foundation models such as the general-purpose MOMENT (Goswami et al., 2024) and the classification-specific MantisV2 (Feofanov et al., 2026). Yet because time series are so diverse and tasks rarely share a label space, these models still train a classification head for every task, and every series must first pass through a large encoder. In our experiments on 188 datasets, training and testing their classifiers takes about 7.5 hours with MOMENT and about 15 minutes with MantisV2, far too slow to retrain and test within seconds.

If every task needs its own head anyway, the feature extractor should be as fast as possible while working well across whole archives of diverse datasets. MiniRocket (Dempster et al., 2021) fits this role naturally: a fixed bank of convolutional features with a ridge classifier, one of the best accuracy-for-cost trade-offs in recent benchmarks (Middlehurst et al., 2024) and more accurate than MantisV2 and MOMENT in our experiments. Yet its widely used GPU implementation (Oguiza, 2023) still needs about ten minutes for these 188 datasets, almost all of it in training ($533\,\mathrm{s}$) rather than inference ($35\,\mathrm{s}$).

The main obstacle to making MiniRocket orders of magnitude faster for a large collection of tasks is how much the tasks vary: training sets range from 12 to $112{,}186$ series, lengths from 9 to $17{,}984$

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

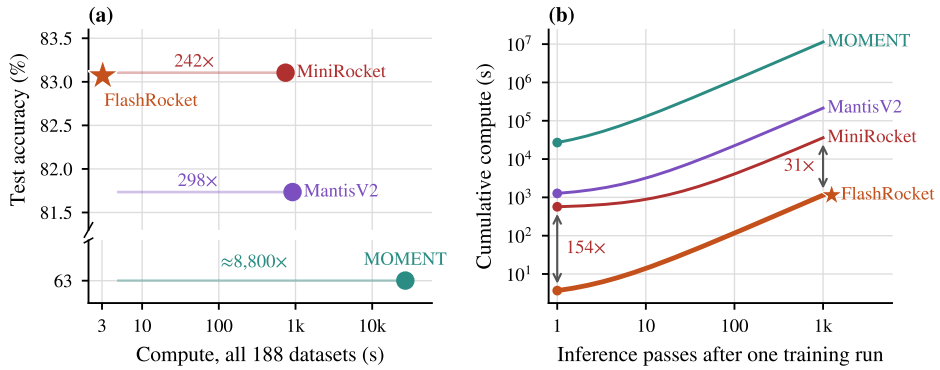


Figure 1: **FlashRocket trains and tests classifiers for all 188 datasets in 3 seconds, as accurately as MiniRocket and more accurately than MantisV2.** **(a)** Test accuracy against compute for the whole archive on one L40S, training and inference interleaved; the accuracy axis is broken to include MOMENT. FlashRocket is 242× faster than MiniRocket and 298× faster than MantisV2. **(b)** Cumulative compute when trained classifiers are reused for k further inference passes (training + k × inference, timed separately; Table 1). FlashRocket's lead over MiniRocket is 154× for one training run and one pass, and settles at 31× as inference dominates.

points and channels from 1 to 1,345. Most tasks sit at the small end (the median has 217 training series against 9,996 features), far too little work to occupy a GPU, so tasks must be batched and overlapped. A few sit at the extreme, and such outliers, like a single very long series, can become the bottleneck for the whole collection. Building one system that stays fast at both ends has been largely overlooked, because the challenge does not arise when classifiers are trained and tested one dataset at a time, as is standard in the field.

We address this challenge with three contributions:

1. **FlashRocket**, a system that trains and tests classifiers on MiniRocket features at archive scale as one GPU workload by overlapping the stages of different tasks (Section 3.1): all 188 from raw series in 3.07 s on one L40S GPU with 16 CPUs, 242× faster than MiniRocket in `tsai` at matching accuracy, and 298× faster and 1.34 points more accurate than MantisV2 (Table 1).
2. **A fused GPU kernel** for feature extraction that computes each convolution on chip and counts all of its thresholds before writing its features, with two exact optimisations: reusing the sum shared by all 84 kernels (1.37×), and tiling long series, which touches only three datasets yet speeds up the whole archive 3.90× (Sections 3.2–3.3).
3. **A cheaper ridge classification head**: one regularisation strength chosen on validation instead of a per-task search, solved exactly by Cholesky or approximately by eight conjugate-gradient steps, depending on training-set size; 4.27× faster without sacrificing accuracy (Section 3.4).

2 Background: MiniRocket

MiniRocket (Dempster et al., 2021) turns a time series into 9,996 features with a fixed set of convolutions and fits a linear classifier on top (Algorithm 1). The authors' implementation runs on CPU with Numba (Lam et al., 2015); the `tsai` library (Oguiza, 2023) reimplements the transform on GPU in PyTorch. Both handle one dataset at a time and fit the classifier on CPU.

Kernels and dilations. Every kernel has length 9, with weight +2 at three positions and −1 at the other six. There are $\binom{9}{3} = 84$ such kernels, and MiniRocket uses all of them. Writing S_k for the three +2 positions of kernel k and indexing the nine positions by $s \in \{-4, \dots, 4\}$, the response at position p under dilation d is

$$c_{k,d}[p] = 2\sum_{s\in S_k} x[p+sd] - \sum_{s\notin S_k} x[p+sd] = 3\sum_{s\in S_k} x[p+sd] - \beta_d[p], \quad (1)$$

Algorithm 1 MiniRocket on one dataset, as in the `tsai` implementation. Lines 1–6 fix the transform, lines 7–8 compute features and lines 9–11 fit the head. FlashRocket (Algorithm 2) computes the same features in one fused pass and replaces lines 9–11 with a cheaper head.

1 **for** each dilation d and kernel k **do**
2 draw a random subset of at most $\min(C, 9)$ channels ▷ *fixed*
3 **for** each kernel k **do**
4 draw one training series x_k
5 **for** each dilation d **do**
6 thresholds ← quantiles of $c_{k,d}(x_k)$ ▷ *the only fitted step*
7 **for** every series, kernel k, dilation d and threshold b **do**
8 feature ← $\mathrm{PPV}_{k,d,b}$ (Eq. 2) of the response summed over the subset ▷ $9{,}996$ *features*
9 scale the feature columns; encode labels as ± 1 one-vs-rest targets
10 eigendecompose the Gram matrix; **for** each α in a grid compute the leave-one-out error
11 solve Eq. 3 with the best α
12 **test:** compute features as in lines 7–8; predict the class with the largest score

where $\beta_d[p] = \sum_{s=-4}^{4} x[p+sd]$ is the sum over all nine points and values outside the series are taken as zero. The second form, used by the authors' CPU implementation, shows that β_d does not depend on the kernel: all 84 kernels at a dilation share it. Dilations are spaced exponentially from 1 to $(L-1)/8$ for series of length L, giving 1 to 29 distinct values per dataset on our archive.

Features. Each (kernel, dilation) pair produces a few features, one per threshold b, each the proportion of positions where the response exceeds b:

$$\mathrm{PPV}_{k,d,b}(x) = \frac{1}{|\mathcal{P}_{k,d}|} \sum_{p \in \mathcal{P}_{k,d}} \mathbf{1}[c_{k,d}[p] > b]\,. \tag{2}$$

Alternating pairs count over all L positions or only over the $L-8d$ positions where the dilated kernel lies inside the series. Each kernel has 119 thresholds divided among its dilations, for $84 \times 119 = 9{,}996$ features in total. Thresholds are quantiles of one random training series' response per kernel.

Multiple channels. For a series with C channels, each (kernel, dilation) pair is assigned a random subset of channels once, of size between 1 and $\min(C, 9)$. The kernel is applied to every channel in the subset and the responses are summed before thresholding. The feature count is therefore $9{,}996$ whatever C is, and at most nine channels enter any one feature.

Classifier. With features $V \in \mathbb{R}^{N \times 9996}$ for N training series and one-vs-rest targets $Y \in \{-1, +1\}^{N \times K}$, MiniRocket fits a ridge classifier

$$W(\alpha) = \arg\min_{W} \left\| \widetilde{Y} - \widetilde{V} W \right\|_F^2 + \alpha \|W\|_F^2, \tag{3}$$

with $\widetilde{V}$ the feature columns after scaling and $\widetilde{Y}$ the centred targets. The strength α is chosen per dataset by leave-one-out error, in closed form from one eigendecomposition of the Gram matrix (Rifkin & Lippert, 2007); the `tsai` recipe uses scikit-learn's `RidgeClassifierCV` over seven values from 10^{-3} to 10^{3} (Pedregosa et al., 2011), after dividing each feature by its standard deviation.

3 FLASHROCKET

3.1 RUNNING TASKS AS ONE GPU WORKLOAD

FlashRocket keeps MiniRocket's features and changes only the head and how the work is executed (Algorithm 2). Each task passes through three stages: *preparation* on CPU (fill gaps, zero-pad), *feature extraction* on GPU (choose thresholds, run the fused GPU kernel; Sections 3.2–3.3), and *training and inference* on GPU (solve the head, then predict the test series; Section 3.4).

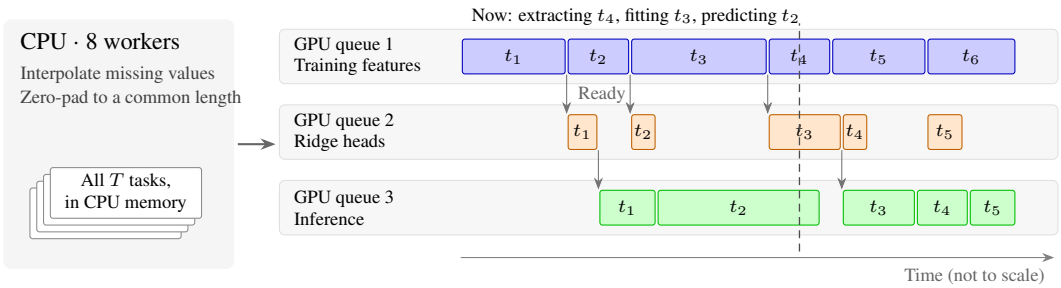


Figure 2: **FlashRocket keeps the GPU busy by overlapping the stages of different tasks.** CPU workers prepare every task ahead of time. The GPU runs three queues at once: one extracts training features task after task, one fits heads as soon as a group's features are ready, and one runs inference once the head exists. At the moment marked *Now*, it is extracting t_4, fitting t_3 and predicting t_2.

Algorithm 2 FlashRocket on T tasks. Lines 3–7 are the extraction (Sections 3.2–3.3), lines 9–12 the head (Section 3.4); the three loops run concurrently on separate GPU queues (Section 3.1).

```
1  send every task to 8 CPU workers: fill gaps, zero-pad ▷ all up front
2  for each task t do ▷ GPU queue 1: training features
3      choose thresholds as in Algorithm 1, lines 3–6
4      if C_t ≤ 9 then store the shared sum β_d per series, dilation and channel ▷ shared sum
5      for each (series, dilation, kernel) in parallel do
6          compute the response on chip; count points above each threshold
7          if L > 4,096 then count per 1,024-point tile and sum the tiles ▷ tiling
8      signal features ready(t)
9  on features ready(t) do ▷ GPU queue 2: heads
10     centre and unit-scale the features; use the fixed α
11     if N_t ≤ D then Cholesky of the N_t × N_t Gram matrix plus αI
12     else 8 conjugate-gradient steps on the D × D system, never formed
13 on head ready(t) do ▷ GPU queue 3: inference
14     extract test features as in lines 4–7; predict
```

Overlapping the stages of different tasks. When each task is finished before the next begins, these stages leave the GPU idle while inputs are prepared and mostly idle while a small head is solved. FlashRocket instead overlaps them across tasks (Figure 2): eight CPU workers prepare every task ahead of time, and extraction, fitting and inference each run on their own GPU queue (a CUDA *stream*). Extraction runs at most two groups of tasks ahead of fitting, and peak allocated GPU memory was 13.7 GB.

3.2 Fused feature extraction

Why MiniRocket in `tsai` is slow. A *GPU kernel* runs as many small *programs* in parallel; each has a few kilobytes of fast on-chip storage (registers) and shares a much larger but slower GPU memory with all other programs. For each dilation, MiniRocket in `tsai` applies a dense convolution with all 84 kernels to every channel, although a feature uses at most nine, and writes every intermediate to GPU memory: the responses, then one comparison per threshold at every position, which are read back only to be averaged (Figure 3a). Section 4.4 counts the resulting data.

How FlashRocket computes each feature in one pass. FlashRocket gives every (series, dilation, kernel) combination its own program, which does all the work for that combination in one pass (Figure 3b). The program reads only the channels assigned to its (kernel, dilation) pair, computes their summed response (Eq. 1) along the series and keeps it in registers. It then counts, for each of the pair's thresholds, the positions where the response exceeds it, and writes only the resulting proportions, the features. The thresholds come from a short first pass over the training data, as in MiniRocket (Algorithm 1, lines 3–6): the same program computes and stores the response of one training series per kernel, and each pair's thresholds are quantiles of that response.

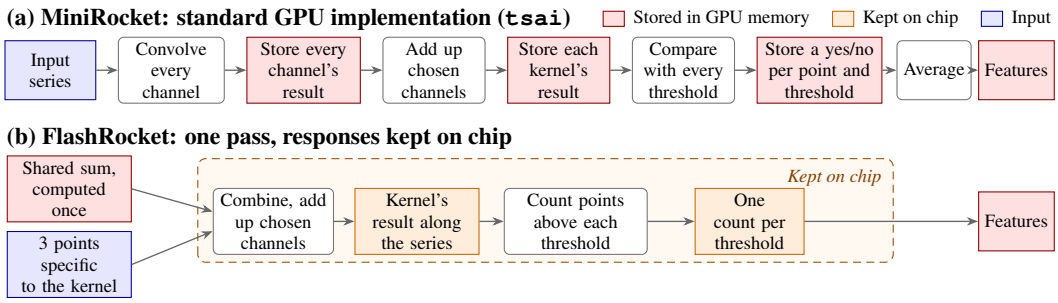


Figure 3: **FlashRocket computes each feature in one on-chip pass instead of storing every intermediate.** **(a)** MiniRocket in `tsai` stores every intermediate result in GPU memory (every channel's convolution, then a yes/no for every point and threshold) only to average it away. **(b)** FlashRocket keeps each kernel's response on chip, counts the points above each threshold, and writes only the final features. The part of the convolution shared by all 84 kernels, the *shared sum*, is computed once and reused (Section 3.3).

3.3 Exact optimisations of the fused kernel

Reusing the sum shared by all kernels. By Eq. 1, a response is the nine-point sum β_d, the same for all 84 kernels, plus three points specific to the kernel. A small preliminary GPU kernel computes this *shared sum* once for every series, dilation, channel and position and stores it; each of the 84 kernels' programs then reads it and adds its three points, instead of re-reading all nine. We store the shared sum only when a task has at most nine channels (168 of 188 tasks). For wider tasks it would have to be stored for every channel, most of which a given pair never reads, so the programs compute β_d inline.

Tiling long series. Because each program keeps the whole series in registers, the storage it needs grows with the series length L. Above $4{,}096$ points it no longer fits in the registers, so FlashRocket splits the series into tiles of $1{,}024$ positions: each program handles one tile of one (series, dilation, kernel) and writes integer counts, and a second small GPU kernel adds the counts of all tiles and divides by the number of positions counted. Only three of our 188 datasets are this long, one of them $17{,}984$ points, yet without tiling the whole archive runs $3.90\times$ slower (Section 4.3).

Exactness. Both optimisations only reorder the same arithmetic: with either one disabled, all 188 sets of test predictions are bit-identical. Against the MiniRocket extractor in `tsai` on synthetic inputs, thresholds differ by at most 6×10^{-6} and features by at most 3.5×10^{-4}, consistent with rounding moving a single borderline position to the other side of a threshold.

3.4 Fitting the ridge heads

One regularisation strength for all tasks. FlashRocket solves Eq. 3 with two departures from the `tsai` recipe. Each feature is centred and scaled to unit ℓ_2 norm with training statistics, as in the authors' original recipe (this is $\widetilde{V}$ below), and α is not searched per task: a single value, $\alpha = 0.464$, was chosen once, from a screen of four values on a validation split of the training data (Appendix D), and is used for every task. Without a search there is no need for the eigendecomposition or for evaluating leave-one-out error at every candidate, and each task reduces to one linear system. The fixed α also costs no accuracy (Section 4.3).

Two solvers, chosen by training-set size. Let N_t be the number of training series and $D = 9{,}996$. The minimiser of Eq. 3 has two equivalent forms, each suited to one solver:

$$W = \widetilde{V}^\top \left(\widetilde{V}\widetilde{V}^\top + \alpha I\right)^{-1} \widetilde{Y} \qquad \text{(Cholesky; } N_t \times N_t \text{ system, when } N_t \le D\text{),} \tag{4}$$

$$W = \left(\widetilde{V}^\top \widetilde{V} + \alpha I\right)^{-1} \widetilde{V}^\top \widetilde{Y} \qquad \text{(conjugate gradients; } D \times D\text{, when } N_t > D\text{).} \tag{5}$$

- **Cholesky (183 tasks, $N_t \leq D$).** We form the $N_t \times N_t$ Gram matrix $\widetilde{V}\widetilde{V}^\top$ and solve Eq. 4 by Cholesky. For the median task, $N_t = 217$, this system is tiny.
- **Conjugate gradients (5 tasks, $N_t > D$).** These tasks have between $11{,}918$ and $112{,}186$ training series. We run eight iterations of conjugate gradients (Hestenes & Stiefel, 1952) on Eq. 5, starting from zero. Each iteration needs one product with $\widetilde{V}$ and one with $\widetilde{V}^\top$, so the $D \times D$ matrix is never formed: for the largest task, forming it would take about $N_t D^2 \approx 10^{13}$ operations, eight iterations about 2×10^{10}. Eight is a fixed budget chosen on validation, not a convergence tolerance (relative residuals 0.05 to 0.17). Against an exact solve it costs 0.010 points on validation and 0.016 on test, and saves a further $1.10\times$.

4 Experiments

4.1 Setup

Datasets. We use 188 classification datasets: all 128 univariate UCR datasets (Dau et al., 2019), all 30 multivariate UEA datasets (Bagnall et al., 2018), 12 univariate datasets newly added by the Bake Off Redux benchmark (Middlehurst et al., 2024), and 18 multivariate datasets from the Multiverse-core archive (Middlehurst et al., 2026). Together they hold $279{,}247$ training and $249{,}041$ test series from eight broad domains, and differ in length, training-set size and channel count by up to four orders of magnitude (Figure 7); several added datasets are discretised regression problems.

Protocol. We use the official train/test splits. All methods start from the same series with missing values interpolated; FlashRocket and MiniRocket zero-pad them to a common length, and the foundation models resize them to their encoders' input length (Appendix B). The two choices that affect accuracy, α and the number of conjugate-gradient steps, were made on a *validation split*: a fixed split of each dataset's training examples, about $80/20$ (187 datasets, $55{,}636$ held-out series). Both were then frozen, and every final classifier is fit on its full training set before testing (Appendix D).

Metrics. We report accuracy and end-to-end compute time:

- **Accuracy.** Macro accuracy, the unweighted mean of test accuracy over the 188 datasets.
- **Compute time.** Wall-clock time to fit every classifier and predict every test series, from series in CPU memory to predicted labels, excluding file reading and one-time start-up (launching the process, compiling GPU code, loading models) for every method.

Baselines. Each baseline keeps its reference implementation's preprocessing and head:

- **MiniRocket** (Dempster et al., 2021): the GPU extractor from `tsai` (Oguiza, 2023) with the standard head, feature scaling without centring and `RidgeClassifierCV` over seven α values on CPU (Pedregosa et al., 2011).
- **MantisV2** (Feofanov et al., 2026), a time series classification foundation model: the frozen pre-trained encoder, standardised features from its second layer, and a logistic regression per dataset.
- **MOMENT** (Goswami et al., 2024): the frozen `MOMENT-1-large` encoder with its official linear classification head, trained for 250 epochs per dataset (Appendix D).

To keep the baselines competitive, all methods run in the same setup (Appendix D): one run for the whole archive, preparation prefetched in the background, encoders on the GPU, and heads fitted concurrently with extraction, on one NVIDIA L40S GPU with the same 16-CPU allocation for every method. In further repeated runs with identical predictions, MiniRocket took 634 to 955 s and FlashRocket 3.065 to 3.078 s (Appendix F).

4.2 Main results

Accuracy. FlashRocket matches MiniRocket: the two differ by 0.03 points, and a paired Wilcoxon test over the 188 datasets does not distinguish them ($p = 0.16$). It is 1.34 points more accurate than MantisV2 ($p = 3 \times 10^{-6}$) and 20.1 points more accurate than MOMENT. The small gap to MiniRocket likely comes from the ridge head rather than the features (Section 4.3). Per-dataset results and further statistics are in Appendices E and H.

Table 1: **FlashRocket matches MiniRocket's accuracy with 242× less compute, and beats both foundation models on accuracy and speed.** Macro test accuracy over all 188 datasets and compute time in seconds on one L40S; speedups are relative to FlashRocket. *Interleaved*: training and inference run together. *Sequential*: every classifier is trained before any inference, timing the two stages separately. FlashRocket's lead is largest in training (210× over MiniRocket) and smaller in inference (31×). Predictions are identical under both schedules. MOMENT timing: Appendix G.

		Interleaved		Sequential			
Method	Accuracy (%)	Total	Speedup	Train	Inference	Total	Speedup
FlashRocket (ours)	83.07	**3.07**	—	**2.54**	**1.14**	**3.68**	—
MiniRocket (`tsai`)	**83.10**	741.8	242×	532.9	35.4	568.4	154×
MantisV2	81.73	914.7	298×	1,058.5	212.7	1,271.3	345×
MOMENT	63.00	≈27,000	≈8,800×	≈15,600	≈11,400	≈27,000	≈7,300×

Speed. FlashRocket trains and tests all 188 datasets in 3.07 s, 242× faster than MiniRocket when both interleave the two stages and 154× when both run them sequentially (Table 1). The inference ratio, 31×, is the one that persists when trained classifiers are reused (Figure 1b).

4.3 Ablations

Table 2: **The fixed regularisation strength and tiling give the largest speedups.** Each row removes components from FlashRocket; slowdown is relative to the full system. Removing tiling or the shared sum changes no prediction. Conjugate gradients trade 0.02 points for 1.10×, and the fixed α is faster than a per-task search with no loss of accuracy. All six settings ran in one job after one data load (3.17 s here against 3.07 s in the headline run).

Setting	Test accuracy (%)	Time (s)	Slowdown
FlashRocket (all components)	83.07	**3.17**	—
– conjugate gradients: exact solve at the fixed α	83.09	3.51	1.10×
– fixed α: per-task leave-one-out over 20 values, exact	83.01	13.55	4.27×
– tiling of long series	83.07	12.39	3.90×
– reusing the shared sum	83.07	4.35	1.37×
– tiling, shared sum and fixed α together	83.01	60.66	19.1×

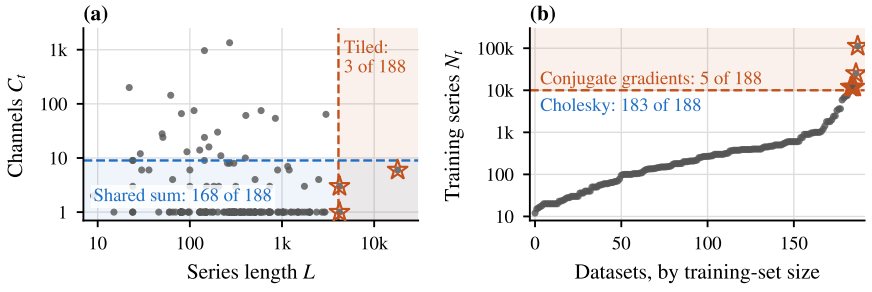


Figure 4: **A few outlying datasets decide what each component is worth. (a)** Datasets by length and channel count. Only three are long enough to be tiled (stars), yet removing tiling slows the whole archive by 3.9×; the shared sum is stored for the 168 datasets with at most nine channels. **(b)** Datasets ordered by training-set size. Only five have more training series than features ($N_t > D$, stars) and need conjugate gradients; the other 183 are solved by Cholesky.

Extraction. Removing tiling or the shared sum changes no prediction, only time (Table 2). Each component activates on a different slice of the archive (Figure 4). Tiling applies to only three datasets, yet removing it slows all 188 by 3.9×: when many tasks share one GPU, the total can be set by the few that scale worst, here the 17,984-point EigenWorms. Per-dataset benchmarks do not show these effects, but at archive scale they strongly affect the runtime.

One regularisation strength. With both solved exactly, one α for all tasks is $3.9\times$ faster than a per-task search over 20 values, with no loss of test accuracy (83.09% against 83.01%). The extra freedom does not help because each task chooses α from few examples: on the validation split, the per-task search (81.66%) falls below every one of the four fixed values we screened (82.03–83.19%; Appendix D). The head also likely explains the small gap to MiniRocket: on synthetic inputs, the extractors differ by at most one borderline position per count (Section 3.3), while changing only the head moves test accuracy between 83.01% and 83.09%, a wider range than the 0.03-point gap.

Combined removals. Removing all three together slows FlashRocket by $19.1\times$, less than the $22.8\times$ product of the single removals. Even then, the fused kernel and overlapped execution alone take 61 s, $12\times$ faster than MiniRocket's 742 s.

4.4 Runtime breakdown and memory traffic

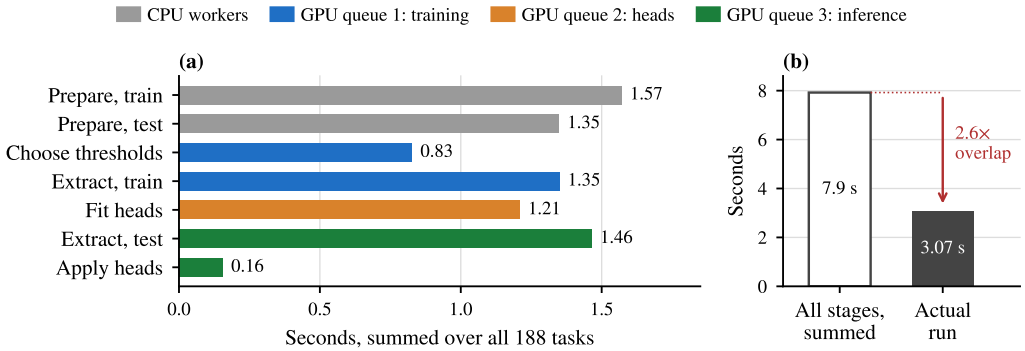


Figure 5: **7.9 s of stage work finishes in 3.07 s because the stages overlap.** **(a)** Each stage's time summed over all 188 tasks. CPU preparation (2.9 s) and feature extraction (2.8 s) take the most, but no single stage dominates, so no one remaining bottleneck limits the run. **(b)** Summed stages against the actual run: CPU preparation and the three GPU queues work on different tasks at once (Figure 2). GPU stage times are measured from the CPU.

Why fused extraction is faster than `tsai`. Feature extraction is among FlashRocket's largest stages (Figure 5), yet far cheaper than in `tsai`. Writing and reading back the 5.9 TB of intermediates that `tsai` stores in the test pass alone (Table 3) would take about 14 s at the L40S's peak 864 GB/s if none of it were cached, about 40% of MiniRocket's 35 s inference time (Table 1). FlashRocket avoids nearly all of this traffic.

Table 3: **MiniRocket in `tsai` writes hundreds of times more intermediate data to GPU memory than FlashRocket.** Bytes written during feature extraction besides the inputs and final features, counted from tensor shapes rather than profiled: caching can absorb some of this traffic, while reading the data back adds more (Appendix C). The reduction is $790\times$ over the archive and grows with channel count, because `tsai` convolves every channel even though each feature uses at most nine: on 64-channel MotorImagery it reaches $87{,}000\times$.

	All 188 datasets		One series	MotorImagery
	Train and test	Test only	($L = 300$)	($C = 64$, $L = 3{,}000$)
MiniRocket (`tsai`)	14.7 TB	5.9 TB	15.3 MB	2.4 TB
FlashRocket (ours)	**18.6 GB**	**10.0 GB**	**25 kB**	**27 MB**
Reduction	$790\times$	$590\times$	$610\times$	$87{,}000\times$

4.5 Speed–accuracy trade-off with fewer MiniRocket kernels

Because FlashRocket computes features only for the kernels it keeps, the number of kernels directly controls its cost. We draw five random subsets at each of nine sizes, from 10% to 90% of the 84

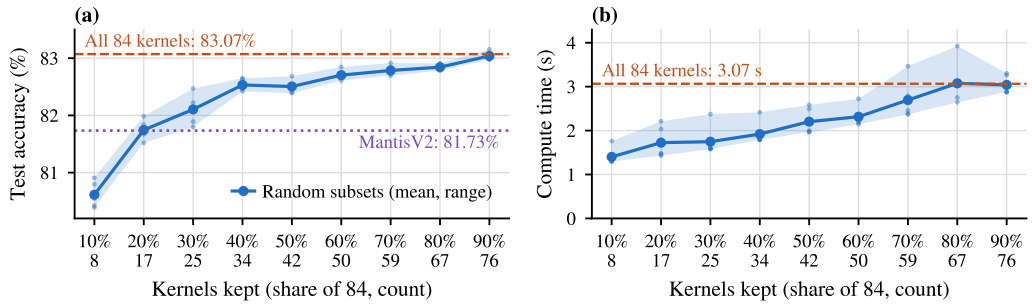


Figure 6: **A fifth of MiniRocket's kernels matches MantisV2's accuracy in 1.7 s, and half come within 0.6 points of all 84.** We keep a random subset of the kernels, the same for all 188 tasks, and measure **(a)** test accuracy and **(b)** compute time; five random subsets per size: means (lines), each subset (dots) and range (bands). Accuracy saturates quickly, while compute falls only about 2× at the smallest size, because preparation and per-task overheads do not depend on kernel count.

kernels, each shared by every task (Figure 6). Accuracy rises quickly and then flattens. A fifth of the kernels (17) already reaches 81.74%, matching MantisV2 (81.73%) in 1.7 s; half the kernels give 82.50%, 0.57 points below the full set, in 2.2 s; the last 40% add only 0.37 points. Compute grows much more slowly than kernel count, from 1.4 s at 8 kernels to 3.0 s at 76, because part of the cost—input preparation and per-task overheads—does not depend on the number of kernels.

Which kernels are kept matters little: at every size the five subsets span at most 0.67 points (Figure 6a), and at most 0.3 from 40% of the kernels on. Pruning methods select kernels or features per dataset (Chen et al., 2024; Uribarri et al., 2024), but one random subset shared by all tasks recovers most of the accuracy, suggesting that how many kernels they keep matters more than which.

5 Related Work

Fast time series classification. Fast variants of ROCKET (Dempster et al., 2020) and MiniRocket include MultiRocket (Tan et al., 2022) and Hydra (Dempster et al., 2023), whose combination is among the most accurate classifiers (Middlehurst et al., 2024). Other fast pipelines use summary features or dictionaries (Lubba et al., 2019; Schäfer & Leser, 2023).

Time series foundation models. Examples include zero-shot forecasters (Garza et al., 2023; Rasul et al., 2023; Ansari et al., 2024; Das et al., 2024; Woo et al., 2024; Liu et al., 2024; Ekambaram et al., 2024; Shi et al., 2025; Liu et al., 2025), language models adapted to time series (Zhou et al., 2023; Jin et al., 2024) and general-purpose models for many tasks (Gao et al., 2024; Talukder et al., 2024).

Efficient GPU execution. Triton (Tillet et al., 2019), Halide (Ragan-Kelley et al., 2013) and TVM (Chen et al., 2018) make fused, tiled GPU code practical, and the roofline model (Williams et al., 2009) explains why memory traffic often limits speed. FlashAttention (Dao et al., 2022) keeps intermediates on chip, and HFTA (Wang et al., 2021) trains many models together on one GPU.

6 Conclusion

With FlashRocket, archive-scale collections of classifiers can be retrained and tested in seconds. Our work has several limitations. First, we compare against two foundation models with per-task heads on 188 datasets; other models or heads may close the gap. Second, each baseline runs in our shared benchmarking setup, and faster implementations of them may exist. Third, we study one GPU; porting these ideas to CPUs or edge devices, and optimising memory and model size, remain open. Fourth, we use public archives as a stand-in for multi-sensor collections and measure the cost of retraining, not accuracy under sensor drift. Fifth, the single α was chosen on these archives and may need re-choosing for a new collection. Finally, we study only MiniRocket's features; the same approach may carry over to other convolutional features.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, implement methods, clean and reformat datasets (interpolating missing values and zero-padding series), and interpret results. We have not used generative AI tools to develop theoretical models or conceptual frameworks or to propose or refine hypotheses, and the remaining required tasks (generating synthetic data, formulating mathematical claims or proofs, translation and qualitative data analysis) are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, create and modify scientific figures, identify relevant literature and format references, suggest a structure for the paper, draft parts of the paper, and edit it to improve readability. We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey, and LLM-generated code was verified and tested for correctness by one author. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The supplementary ZIP contains the full FlashRocket code (the fused GPU kernels, the ridge heads and the scheduler), the baseline scripts, the fixed validation splits, and scripts that regenerate the paper's tables and plotted figures from the recorded runs. Appendix B lists every seed and constant needed to rebuild the features. Compiling the GPU kernels takes about two minutes once per process and is excluded from all timings. All datasets are public and are not redistributed.

REFERENCES

Abdul Fatir Ansari, Lorenzo Stella, Ali Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Bernie Wang. Chronos: Learning the language of time series. *Transactions on Machine Learning Research*, 2024.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The UEA multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018. URL https://arxiv.org/abs/1811.00075.

Shaowu Chen, Weize Sun, Lei Huang, Xiao Peng Li, Qingyuan Wang, and Deepu John. POCKET: Pruning random convolution kernels for time series classification from a feature selection perspective. *Knowledge-Based Systems*, 300:112253, 2024. doi: 10.1016/j.knosys.2024.112253.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*, pp. 578–594, 2018. URL https://www.usenix.org/conference/osdi18/presentation/chen.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2205.14135.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10148–10167, 2024.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

Angus Dempster, François Petitjean, and Geoffrey I. Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020. doi: 10.1007/s10618-020-00701-z. URL https://doi.org/10.1007/s10618-020-00701-z.

Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. MiniRocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 248–257, 2021. doi: 10.1145/3447548.3467231. URL https://arxiv.org/abs/2012.08791.

Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. HYDRA: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5):1779–1805, 2023. URL https://doi.org/10.1007/s10618-023-00939-3.

Jessilyn Dunn, Ryan Runge, and Michael Snyder. Wearables and the medical revolution. *Personalized Medicine*, 15(5):429–448, 2018.

Vijay Ekambaram, Arindam Jati, Pankaj Dayama, Sumanta Mukherjee, Nam H. Nguyen, Wesley M. Gifford, Chandra Reddy, and Jayant Kalagnanam. Tiny time mixers (TTMs): Fast pre-trained models for enhanced zero/few-shot forecasting of multivariate time series. In *Advances in Neural Information Processing Systems*, 2024.

Vasilii Feofanov, Songkang Wen, Jianfeng Zhang, Lujia Pan, and Ievgen Redko. MantisV2: Closing the zero-shot gap in time series classification with synthetic data and test-time strategies. *arXiv preprint arXiv:2602.17868*, 2026. URL https://arxiv.org/abs/2602.17868.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014. doi: 10.1145/2523813.

Shanghua Gao, Teddy Koker, Owen Queen, Thomas Hartvigsen, Theodoros Tsiligkaridis, and Marinka Zitnik. UniTS: A unified multi-task time series model. In *Advances in Neural Information Processing Systems*, 2024.

Azul Garza, Cristian Challu, and Max Mergenthaler-Canseco. TimeGPT-1. *arXiv preprint arXiv:2310.03589*, 2023.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. MOMENT: A family of open time-series foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16115–16152. PMLR, 2024. URL https://proceedings.mlr.press/v235/goswami24a.html.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-LLM: Time series forecasting by reprogramming large language models. In *International Conference on Learning Representations*, 2024.

Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Real-time patient-specific ECG classification by 1-D convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3):664–675, 2016. doi: 10.1109/TBME.2015.2468589.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015. URL https://doi.org/10.1145/2833157.2833162.

Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32369–32399, 2024.

Yong Liu, Guo Qin, Zhiyuan Shi, Zhi Chen, Caiyin Yang, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Sundial: A family of highly capable time series foundation models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery*, 33(6): 1821–1852, 2019. doi: 10.1007/s10618-019-00647-x. URL https://doi.org/10.1007/s10618-019-00647-x.

Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 38(4):1958–2031, 2024. doi: 10.1007/s10618-024-01022-1.

Matthew Middlehurst, Aiden Rushbrooke, Ali Ismail-Fawaz, Maxime Devanne, Germain Forestier, Angus Dempster, Geoffrey I. Webb, Christopher Holder, and Anthony Bagnall. The Multiverse of time series machine learning: an archive for multivariate time series classification. *arXiv preprint arXiv:2603.20352*, 2026. URL https://arxiv.org/abs/2603.20352.

Ignacio Oguiza. tsai — a state-of-the-art deep learning library for time series and sequential data. https://github.com/timeseriesAI/tsai, 2023.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Lukasz Piwek, David A. Ellis, Sally Andrews, and Adam Joinson. The rise of consumer health wearables: Promises and barriers. *PLoS Medicine*, 13(2):e1001953, 2016.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Fredo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 519–530, 2013. URL https://doi.org/10.1145/2491956.2462176.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. Lag-Llama: Towards foundation models for probabilistic time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

Ryan M. Rifkin and Ross A. Lippert. Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, MIT Computer Science and Artificial Intelligence Laboratory, 2007.

Patrick Schäfer and Ulf Leser. WEASEL 2.0: A random dilated dictionary transform for fast, accurate and memory constrained time series classification. *Machine Learning*, 112:4763–4788, 2023. URL https://doi.org/10.1007/s10994-023-06395-w.

Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Zhou Ye, Qingsong Wen, and Ming Jin. Time-MoE: Billion-scale time series foundation models with mixture of experts. In *International Conference on Learning Representations*, 2025.

Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2015. doi: 10.1109/TII.2014.2349359.

Sabera Talukder, Yisong Yue, and Georgia Gkioxari. TOTEM: TOkenized time series EMbeddings for general time series analysis. *Transactions on Machine Learning Research*, 2024.

Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I. Webb. MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5):1623–1646, 2022. doi: 10.1007/s10618-022-00844-1. URL https://doi.org/10.1007/s10618-022-00844-1.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019. URL https://www.eecs.harvard.edu/~htk/publication/2019-mapl-tillet-kung-cox.pdf.

Gonzalo Uribarri, Federico Barone, Alessio Ansuini, and Erik Fransén. Detach-ROCKET: sequential feature selection for time series classification with random convolutional kernels. *Data Mining and Knowledge Discovery*, 38(6):3922–3947, 2024. doi: 10.1007/s10618-024-01062-7.

Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. Horizontally fused training array: An effective hardware utilization squeezer for training novel deep learning models. In *Proceedings of Machine Learning and Systems*, volume 3, pp. 599–623, 2021.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. URL https://doi.org/10.1145/1498765.1498785.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One fits all: Power general time series analysis by pretrained LM. In *Advances in Neural Information Processing Systems*, 2023.

A DATASETS

Figure 7 summarises the 188 datasets by archive, domain, length, training-set size and channel count.

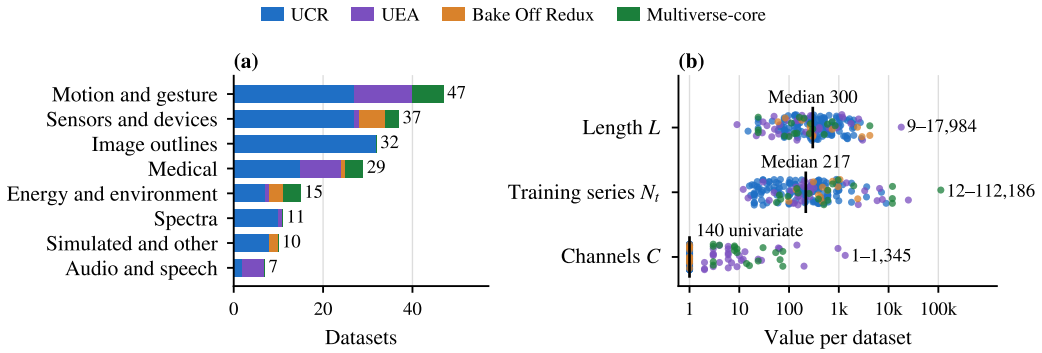


Figure 7: **188 datasets span four archives, eight domains and up to four orders of magnitude in size.** All 128 UCR and 30 UEA datasets, plus 30 recent ones from Bake Off Redux and Multiverse-core. **(a)** Datasets per domain, grouped from the archives' own type labels (e.g. UCR's ECG, EEG and EOG as Medical); we assigned the 30 recent datasets. **(b)** Length, training-set size and channel count, one dot per dataset, with the median (black) and range (right).

B IMPLEMENTATION DETAILS

Preparation. Eight CPU worker processes, started with Python's `forkserver` method and a precompiled Numba interpolation routine, write each task's training and test series into shared memory as float32 arrays of shape (series, channels, length). Within each series and channel, missing values are filled by linear interpolation between finite neighbours; leading and trailing gaps take the nearest finite value, and an entirely missing channel is set to zero. Series are right-padded with zeros to $\max(9, L_{\text{train}}, L_{\text{test}})$, where L_{train} and L_{test} are the longest training and test series of the dataset. This uses the lengths, never the labels, of the test series; it affects five datasets whose longest test series exceeds the longest training series. The MiniRocket baseline receives identical arrays; the foundation models additionally resize every series to their 512-point input with their authors' procedure. Preparation of every task is submitted before extraction begins.

Feature geometry. The geometry follows the `tsai` implementation exactly: 84 kernels of length 9; 119 features per kernel, divided among the deduplicated dilations in proportion to how many of the 32 candidates round to each, with the remainder assigned round-robin; channel subsets drawn with `RandomState(42)`; and the alternating choice between padded and valid positions. For calibration, one training series is drawn per kernel with the same seeded procedure as `tsai`, and the thresholds of a (kernel, dilation) pair are the quantiles of that series' response at levels $(i\varphi) \bmod 1$, with φ the golden ratio. On CPU, dilation counts and channel subsets match `tsai` exactly for 88 combinations of length and channel count.

Extraction kernels. Four Triton (Tillet et al., 2019) kernels, all launched with four warps and floating-point contraction disabled. `make_base` computes $-\beta_d$ on a grid of (series, dilation $\times$ channel, block of 1,024 positions). `joint` runs one program per (series, (dilation, kernel) pair) with a block of the next power of two above L; it reads the shared sum (or computes it inline when $C > 9$), adds three times the kernel's three taps for each channel in the subset, and compares the response with the pair's thresholds four at a time, masking the last incomplete group; each feature is stored through a round-to-nearest division, so repeated runs agree bit for bit. For $L >$ 4,096, `joint_tiled` adds a tile axis of 1,024 positions and writes `int32` partial counts, and `reduce_tiled` sums the tiles and divides, 32 features per program. Each kernel is specialised on the task's length and channel count, so the 188 tasks require one compilation per distinct shape; this one-time compilation took about 127 s per process and is excluded from compute time. Series are processed in batches of at most 1,024 series and 128 MiB of input.

Head. Columns are centred with training means (constant columns become zero) and divided by their training ℓ_2 norm, with norms below 10^{-12} set to one. Targets are ± 1 one-vs-rest, a single column for binary tasks, centred. The fixed $\alpha = 0.4641588833612777$ is index 9 of the 20-value grid used by the per-task search: five decades from 10^{-8} to 10^{-4}, ten log-spaced values from 10^{-3} to 10^{3}, and five decades from 10^{4} to 10^{8}. Tasks with $N_t \leq D$ use `cholesky_ex` and `cholesky_solve` on the $N_t \times N_t$ system; tasks with $N_t > D$ run eight conjugate-gradient iterations from zero with all target columns updated together. Weights are mapped back to the original feature scale and the intercept is recovered from the centring statistics. Heads and matrix products are float32, with TF32 tensor-core matrix multiplication enabled. Multiclass scores take the arg-max; binary scores are thresholded at zero.

Execution. Training features are extracted on the default stream. One ridge thread owns a second stream and waits on the event recorded after each task's features. Tasks are processed in a fixed order that groups similar head sizes, and the main thread waits for the head of the group before last before extracting the next group, which bounds the feature matrices held at once. One prediction thread owns a third stream, starts once the first head is ready, and runs inference on test series task by task.

C Intermediate data written to GPU memory

Table 3 (Section 4.4) counts the bytes each implementation writes to GPU memory besides its inputs and final features, computed from the tensor shapes in the code for every dataset's actual length, channel count, dilations and number of series (training and test, plus calibration). For `tsai`, per dilation: the convolution output over all channels ($N \times 84C \times L$ float32); for $C > 1$, the channel-mask product and the channel sum; and, for each half of the kernels, the threshold comparison as a boolean tensor and its float copy ($N \times 42 \times L' \times n_d$, with $L' = L$ or $L - 8d$). For FlashRocket: the shared sum ($N \times n_{\text{dil}} \times C \times L$ float32, for $C \leq 9$), the tile counts for $L > 4{,}096$, and the calibration responses. Every intermediate is read back at least once, so actual traffic is higher; the counts ignore caching and are not a profiler measurement. Both methods also write the same 21.1 GB of final features.

The three largest datasets by this count (MotorImagery, EigenWorms and PEMS-SF, the last with 963 channels) account for 31% of `tsai`'s total, because it convolves every channel and compares at every position.

D Validation protocol and selection record

The validation split. Both accuracy-relevant choices were made on a fixed partition of each dataset's official *training* examples, recorded index by index and checked at load to be disjoint, to cover the training set, and to keep every class in the fitting part. It is approximately, not exactly, an $80/20$ split per task. The denominator is 187 datasets and $55{,}636$ held-out series: Fungi has no held-out examples, and the three Pig tasks hold out 52 of their 104 training examples. The kernel-subset sweep (Section 4.5) fits on full training sets and reports test accuracy only.

Choosing one regularisation strength. A four-value screen over indices 8–11 of the 20-value grid was run on the validation split, with highest macro validation accuracy as the criterion. Index 9 was frozen before any full-training or test fitting (Table 4). The screen ran on an earlier version of the pipeline that merged duplicate features; the final pipeline scores the chosen α at 83.17% (exact solve) and 83.16% (eight conjugate-gradient steps) on the same split.

Table 4: Global-α screen on the 187-dataset validation split.

Index	α	Validation accuracy (%)
8	0.1	83.135
9	0.4642	**83.191**
10	2.1544	82.980
11	10	82.032

Choosing the number of conjugate-gradient steps. With α fixed, three screens on the same split covered 1–7, 8, 12, 16, 32 and 64 steps against the exact solve. Accuracy rises monotonically and saturates early: 83.011% after one step, 83.154% after seven, 83.159% after eight, and 83.170% for the exact solve. Eight steps were kept as a short fixed budget within 0.010 points of exact. The residual gap is concentrated in the two largest tasks and closes steadily with more steps (Table 5). Timings across these screens come from separate jobs and are not an iteration-cost curve.

Table 5: Validation accuracy (%) of the two tasks most affected by the conjugate-gradient budget.

Steps	1	4	8	12	16	exact
InsectWingbeat	42.62	63.18	66.66	67.16	67.64	67.94
AustraliaRainfall	72.27	75.05	76.02	76.24	76.37	76.65

Baseline recipes. We follow each baseline's reference recipe for preprocessing, features and head, and change only how the work is scheduled, so that every method uses the same GPU and CPUs as fully as its recipe allows. MiniRocket uses the `tsai` `MiniRocketFeatures` class unchanged except for chunking its calibration convolution, 9,996 features, `StandardScaler(with_mean=False)` and `RidgeClassifierCV` with $\alpha \in \{10^{-3}, \ldots, 10^{3}\}$, seed 42. MantisV2 uses the frozen author encoder with layer-2 features, standard scaling and logistic regression ($C = 1$, L-BFGS, 500 iterations); missing values are interpolated as for the other methods, which is our assumption rather than a documented author policy. MOMENT uses the frozen `MOMENT-1-large` encoder with the official linear head, dropout 0.1, Adam at 10^{-4}, batch 64, and a fixed 250-epoch budget. Features are extracted on the GPU in batches of 133 series for MiniRocket, the largest that fits in GPU memory on 64-channel MotorImagery (found by a binary search that checks only memory, not accuracy), 256 for MantisV2 and 512 for MOMENT. The budget came from a separate five-dataset pilot on training data only (ACSF1, ECG5000, FordA capped at 1,000 series, BasicMotions, NATOPS), whose validation accuracy was 81.32% at 250 epochs and 80.72% at 300. No baseline used our validation split to choose its head. The MOMENT authors also report an SVM on frozen embeddings, which scores higher on UCR but needs a per-dataset hyperparameter sweep; we use their official linear head instead.

E Paired comparison with MiniRocket

Across the 188 datasets (Figure 8) FlashRocket wins 47, loses 63 and ties exactly 78, for a mean difference of -0.034 points; a paired Wilcoxon signed-rank test gives $W = 2577.5$, $p = 0.157$. On the 158 UCR and UEA datasets the mean difference is -0.010 points, $p = 0.227$. The median absolute per-dataset difference is 0.18 points, with extremes of $+10.6$ and -6.7. By collection, FlashRocket scores 85.25% on UCR, 74.78% on UEA and 82.07% on the 30 added datasets, against 85.20%, 75.04% and 82.23% for MiniRocket and 83.61%, 73.35% and 82.13% for MantisV2.

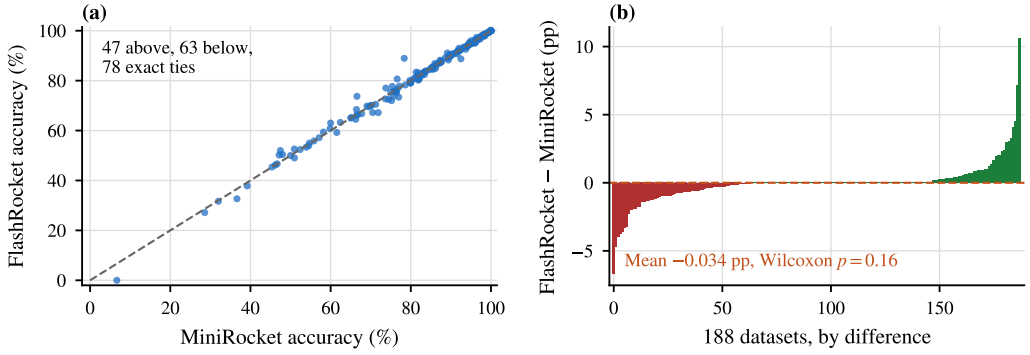


Figure 8: **FlashRocket against MiniRocket, dataset by dataset.** **(a)** Paired accuracies; the diagonal is equality. **(b)** The same 188 differences, sorted. 78 datasets agree exactly and the mean difference is -0.034 points.

F TIMING REPEATS

Four sequential MiniRocket runs on one node, in the order interleaved, sequential, sequential, interleaved, all produced identical predictions at 83.105%: 954.9 s (interleaved), 643.6 s (sequential), 645.1 s (sequential) and 634.2 s (interleaved). The first run is $1.51\times$ the last, and we keep it rather than discard it as warm-up. Run order and other activity on the node were not controlled. The two FlashRocket runs of the selected configuration took 3.078 s and 3.065 s with identical predictions; we report the second.

G MOMENT RUNTIME RECONSTRUCTION

MOMENT completed all 188 datasets at 63.003%, with identical predictions under both schedules, but never in one uninterrupted run. Measured cumulative worker time was $29{,}031$ s (interleaved) and $29{,}012$ s (sequential). Removing set-up and export (376 and 384 s), synchronous checkpoint writes (999 and $1{,}042$ s), the unsaved tail of a failed attempt (621 and 616 s) and replay of already-finished datasets (11 and 17 s) leaves an estimated $27{,}024$ and $26{,}952$ s, about 7.5 hours per schedule. This is an accounting estimate, not a measurement: some segment endpoints rest on file timestamps, some checkpoint-read and replay overhead remains included, and segments ran on different machines. The one-minute difference between schedules is not evidence of a speed difference. Because these figures still include some overhead, they overstate MOMENT's true compute time. For Figure 1b we split the sequential total into training and one inference pass using the inference time measured in the final sequential recovery segment ($11{,}366$ s), leaving about $15{,}600$ s for training.

H PER-DATASET RESULTS

Test accuracy (%) on each of the 188 datasets for FlashRocket, MiniRocket and MantisV2, grouped by source archive. Bold marks the best of the three on each row. MOMENT is omitted for width and is lower on almost every dataset. FlashRocket and MiniRocket agree exactly on 78 datasets. Some names are abbreviated.

UCR 2018, univariate (128 datasets).

Dataset	FlashRocket	MiniRocket	MantisV2	Dataset	FlashRocket	MiniRocket	MantisV2
ACSF1	**93.0**	92.0	75.0	ItalyPowerDemand	96.5	**96.7**	94.6
Adiac	81.6	81.8	**83.1**	LargeKitchenAppl	**83.2**	81.9	81.6
AllGestWiimoteX	**75.4**	75.0	71.3	Lightning2	77.0	73.8	**78.7**
AllGestWiimoteY	75.7	**76.0**	73.3	Lightning7	80.8	82.2	**86.3**
AllGestWiimoteZ	**73.7**	66.6	64.3	Mallat	95.0	**95.1**	92.0
ArrowHead	**85.1**	**85.1**	**85.1**	Meat	**100.0**	**100.0**	96.7
Beef	**86.7**	**86.7**	73.3	MedicalImages	79.3	**79.9**	77.1
BeetleFly	**90.0**	**90.0**	85.0	MelbournePedestrian	**96.4**	96.0	95.8
BirdChicken	**90.0**	**90.0**	**90.0**	MidPhalanxOutlAgeGrp	**55.8**	**55.8**	54.5
BME	**100.0**	**100.0**	**100.0**	MidPhalanxOutlCorr	84.5	**85.6**	84.2
Car	**91.7**	**91.7**	81.7	MidPhalanxTW	53.9	**54.5**	51.3
CBF	**99.9**	**99.9**	99.7	MixedShapesRegTrain	97.3	97.2	**97.5**
Chinatown	**98.3**	**98.3**	96.2	MixedShapesSmTrain	**95.1**	**95.1**	94.6
ChlorineConc	73.8	**75.7**	71.1	MoteStrain	92.3	92.3	**93.9**
CinCECGTorso	84.3	84.6	**85.1**	NonInvFetalECGThx1	**93.4**	93.2	93.1
Coffee	**100.0**	**100.0**	**100.0**	NonInvFetalECGThx2	95.0	**95.9**	93.5
Computers	72.0	**75.2**	70.0	OliveOil	**93.3**	**93.3**	86.7
CricketX	**81.0**	**81.0**	78.2	OSULeaf	94.6	94.6	**96.3**
CricketY	**82.8**	**82.8**	81.3	PhalangesOutlsCorr	**83.4**	**83.4**	79.5
CricketZ	**83.6**	**83.6**	81.3	Phoneme	27.2	28.6	**35.4**
Crop	75.2	**76.4**	73.5	PickupGestWiimoteZ	**86.0**	**86.0**	80.0
DiatomSizeRed	92.5	92.5	**92.8**	PigAirwayPress	**88.9**	78.4	57.7
DistPhalanxOutlAgeGrp	70.5	71.2	**77.7**	PigArtPress	**96.2**	**96.2**	90.9
DistPhalanxOutlCorr	78.3	**78.6**	76.8	PigCVP	**94.2**	**94.2**	93.8
DistPhalanxTW	**69.8**	69.1	67.6	PLAID	92.6	93.7	**94.0**
DodgerLoopDay	**65.0**	**65.0**	52.5	Plane	**100.0**	**100.0**	**100.0**
DodgerLoopGame	**87.7**	**87.7**	86.2	PowerCons	**98.9**	**98.9**	97.2
DodgerLoopWkend	**98.6**	**98.6**	97.1	ProxPhalanxOutlAgeGrp	82.4	83.4	**84.4**
Earthquakes	73.4	**77.0**	72.7	ProxPhalanxOutlCorr	**90.4**	89.7	87.6
ECG200	**92.0**	**92.0**	88.0	ProxPhalanxTW	79.0	**80.0**	76.1
ECG5000	**94.6**	94.5	93.5	RefrigerationDevices	52.0	47.5	**52.3**
ECGFiveDays	**100.0**	**100.0**	99.3	Rock	82.0	82.0	**84.0**
ElectricDevices	72.5	**74.4**	71.8	ScreenType	50.4	48.0	**50.9**

Dataset	FlashRocket	MiniRocket	MantisV2	Dataset	FlashRocket	MiniRocket	MantisV2
EOGHorizontalSignal	**63.3**	62.4	57.7	SemgHandGenderCh2	**90.0**	89.5	89.7
EOGVerticalSignal	**55.0**	54.7	51.4	SemgHandMovementCh2	**68.4**	66.4	62.4
EthanolLevel	**59.4**	58.2	59.2	SemgHandSubjectCh2	**86.9**	86.0	85.3
FaceAll	**80.8**	**80.8**	75.7	ShakeGestWiimoteZ	**90.0**	**90.0**	**90.0**
FaceFour	**100.0**	**100.0**	98.9	ShapeletSim	**100.0**	**100.0**	97.8
FacesUCR	**95.8**	**95.8**	91.2	ShapesAll	92.0	**92.2**	90.0
FiftyWords	**84.0**	83.7	82.2	SmallKitchenAppl	80.5	**81.9**	78.7
Fish	**98.3**	**98.3**	96.0	SmoothSubspace	96.7	**97.3**	**97.3**
FordA	**95.2**	94.6	94.4	SonyAIBOSurf1	89.2	**89.4**	83.4
FordB	**82.2**	81.7	81.1	SonyAIBOSurf2	91.5	91.5	**93.7**
FreezerRegTrain	**100.0**	**100.0**	99.6	StarLightCurves	**98.1**	**98.1**	97.3
FreezerSmTrain	96.5	96.5	**98.5**	Strawberry	97.8	**98.4**	96.5
Fungi	**100.0**	**100.0**	97.3	SwedishLeaf	97.0	96.8	**97.4**
GestureMidAirD1	**77.7**	75.4	76.9	Symbols	**98.3**	**98.3**	97.8
GestureMidAirD2	59.2	61.5	**65.4**	SyntheticControl	98.3	98.3	**99.0**
GestureMidAirD3	**45.4**	**45.4**	40.8	ToeSegmentation1	95.2	96.1	**96.5**
GestPebbleZ1	90.7	90.7	**97.1**	ToeSegmentation2	90.8	91.5	**93.1**
GestPebbleZ2	**87.3**	**87.3**	85.4	Trace	**100.0**	**100.0**	**100.0**
GunPoint	**99.3**	**99.3**	98.0	TwoLeadECG	99.8	99.8	**99.9**
GunPointAgeSpan	**99.7**	**99.7**	99.4	TwoPatterns	99.6	99.6	**99.7**
GunPointMvF	**100.0**	**100.0**	**100.0**	UMD	**99.3**	**99.3**	**99.3**
GunPointOldVersusYoung	**100.0**	**100.0**	99.7	UWaveGestLibAll	**97.0**	**97.0**	95.1
Ham	69.5	69.5	**70.5**	UWaveGestLibX	**85.2**	**85.2**	84.4
HandOutls	94.6	**94.9**	93.0	UWaveGestLibY	**77.8**	77.2	76.7
Haptics	49.0	**51.0**	50.0	UWaveGestLibZ	**80.1**	79.8	77.5
Herring	67.2	**71.9**	68.8	Wafer	**99.9**	**99.9**	99.7
HouseTwenty	**95.0**	**95.0**	93.3	Wine	**87.0**	**87.0**	83.3
InlineSkate	**46.2**	**46.2**	38.9	WordSynonyms	75.9	**76.0**	71.0
InsectEPGRegTrain	**100.0**	**100.0**	**100.0**	Worms	70.1	70.1	**77.9**
InsectEPGSmTrain	98.8	98.8	**100.0**	WormsTwoClass	76.6	76.6	**79.2**
InsectWingbeatSnd	66.2	**66.5**	59.5	Yoga	**90.8**	90.5	85.6

Bake Off Redux additions, univariate (12 datasets).

Dataset	FlashRocket	MiniRocket	MantisV2	Dataset	FlashRocket	MiniRocket	MantisV2
AconityPrinterLarge	95.8	**95.9**	95.3	Covid3Month	67.2	**70.5**	59.0
AconityPrinterSmall	**97.9**	**97.9**	97.6	FloodModeling1	**91.1**	89.1	88.6
AsphaltObstUni	87.2	87.5	**90.0**	FloodModeling2	96.0	95.0	**97.0**
AsphaltPaveTypeUni	88.3	89.4	**93.1**	FloodModeling3	89.1	88.6	**90.2**
AsphaltRegUni	97.3	96.4	**99.1**	SharePriceIncr	64.5	**66.3**	64.0
Colposcopy	32.7	**36.6**	34.7	Tools	79.1	**79.9**	76.1

UEA 2018, multivariate (30 datasets).

Dataset	FlashRocket	MiniRocket	MantisV2	Dataset	FlashRocket	MiniRocket	MantisV2
ArticularyWordRec	**99.3**	**99.3**	**99.3**	InsectWingbeat	66.8	67.7	**69.9**
AtrialFibrillation	0.0	6.7	**13.3**	JapaneseVowels	**98.6**	98.4	98.1
BasicMotions	**100.0**	**100.0**	**100.0**	Libras	91.7	91.7	**94.4**
CharTrajectories	99.3	**99.4**	98.3	LSST	65.2	65.2	**65.9**
Cricket	**98.6**	**98.6**	**98.6**	MotorImagery	**50.0**	**50.0**	**50.0**
DuckDuckGeese	**70.0**	**70.0**	50.0	NATOPS	**92.2**	**92.2**	89.4
EigenWorms	95.4	**96.2**	88.5	PEMS-SF	82.1	82.7	**83.2**
Epilepsy	**100.0**	**100.0**	**100.0**	PenDigits	96.3	96.6	**97.5**
ERing	98.1	97.8	**98.9**	PhonemeSpectra	31.7	**32.0**	31.9
EthanolConc	**50.2**	47.1	42.2	RacketSports	89.5	89.5	**90.1**
FaceDetection	**60.8**	59.8	58.3	SelfRegSCP1	88.7	**92.5**	86.7
FingerMovements	63.0	60.0	**64.0**	SelfRegSCP2	53.3	**53.9**	51.1
HandMovementDir	37.8	**39.2**	31.1	SpokenArabicDigits	**99.4**	**99.4**	98.7
Handwriting	**52.6**	50.9	39.8	StandWalkJump	**46.7**	**46.7**	40.0
Heartbeat	72.7	73.7	**79.0**	UWaveGestLib	93.4	**94.1**	92.2

Multiverse-core additions, multivariate (18 datasets).

Dataset	FlashRocket	MiniRocket	MantisV2	Dataset	FlashRocket	MiniRocket	MantisV2
ApplEnergy	85.7	85.7	**90.5**	BoneProbAgeGrp	66.3	66.7	**68.8**
AsphaltObstCoord	84.9	**86.2**	84.4	EmoPain	84.8	**85.4**	82.0
AsphaltRegCoord	98.0	98.4	**99.6**	EyesOpenShut	**52.4**	**52.4**	47.6
AustraliaRainfall	75.8	76.3	**77.0**	IRDS-SFL	80.2	81.9	**91.4**
AutomotiveRoad	**83.1**	**83.1**	79.2	KIMORE-PR-C	57.1	57.1	**71.4**
BeijingPM10Qual	**83.4**	81.3	79.8	KINECAL-QSEO	**94.1**	**94.1**	**94.1**
BeijingPM25Qual	**87.8**	87.5	84.2	UCDHE-Rowing	**80.7**	76.6	75.7
Blink	**97.6**	97.3	97.3	UIPRMD-DS-C	94.4	94.4	**97.2**
BoneIntAgeGrp	81.3	**82.0**	78.4	WISDM	**88.2**	87.3	80.6