FROM 1D TO 4D: A UNIFIED SEQUENCE PREDICTION FRAMEWORK WITH THE KUN-V2 ARCHITECTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequence prediction is central to modern deep learning. In recent general-purpose time-series models, sequence completion and reconstruction, predicting the hidden part of a sequence from its visible part, have become the common basis of forecasting, imputation and anomaly detection. These models, however, are generally hard to extend to higher dimensionalities. Neural operators such as FNO and U-NO can be extended, but at a comparatively high complexity. To address the order in which subspaces are processed and the efficiency on high-dimensional data, we propose KUN-V2, a unified encoder–decoder framework that processes inputs of arbitrary spatial dimensionality via two ingredients: an auto-factorization that splits each spatial axis into balanced chunks, and a compact *spec language* that selects in what order and grouping the chunks are reduced. We show that the time complexity is linear in the product of all input dimensions and invariant under any reordering or regrouping of the chunk schedule. We validate the framework in-distribution on a single task, masked-half sequence completion, across all four targeted dimensionalities and, on synthetic data, a fifth with a single shared codebase: only the input shape, spec string, and chunk multiplicities change between settings. The evaluation covers 21 datasets, comprising synthetic and real data at $N = 1 \dots 4$ and the two synthetic families at $N = 5$. Against eleven learned baselines, KUN-V2 places in the top three on 17 and first on 11, and on a synthetic cost ladder its measured training time and peak memory stay within the linear-in-element-count bound. KUN-V2 thus reduces the cost of moving to a new dimensionality from architectural re-design to a configuration change. Code is available at https://github.com/jccxfgj-beep/kun-v2/tree/main.

1 INTRODUCTION

Sequence modelling is a central problem in deep learning (Vaswani et al., 2017; Gu et al., 2022; Gu & Dao, 2024), and sequence models are the backbone of time-series forecasting (Nie et al., 2023), video prediction (Wang et al., 2017) and physical simulation (Li et al., 2021). Among sequence tasks, completion, i.e. predicting the hidden part of a sequence from its visible part, can be regarded as a foundational one for forecasting, imputation, anomaly detection, etc. (Gao et al., 2024; Goswami et al., 2024).

Recent general-purpose time-series models accordingly treat several of these tasks within a single framework, either through a shared backbone (Wu et al., 2023; Zhou et al., 2023; Gao et al., 2024) or through pre-training on masked reconstruction (Goswami et al., 2024). These unified models, however, remain tied to one input dimensionality, and multidimensional sequence processing is still an open research topic: moving from one dimensionality to another usually means moving to a different family of models.

Each step up in dimensionality has so far required a new tokenization scheme, a new positional encoding, and a new mixing primitive. Related ideas can be found in neural operators such as FNO (Li et al., 2021) and U-NO (Rahman et al., 2023): although they extend to inputs of any dimensionality, their cost is comparatively high ($O(E \log E)$ in time), and their mixing scheme is fixed, so the order in which the model processes subspaces of the axes cannot be edited. We therefore treat per-dimensionality redesign as an avoidable implementation cost. The core operation is

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.

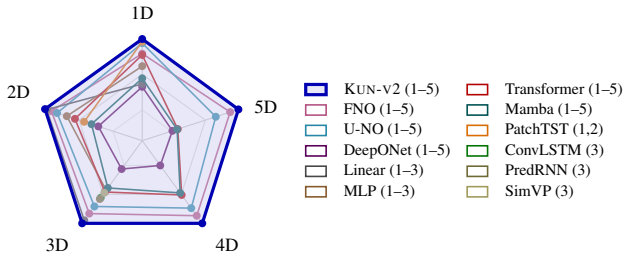


Figure 1: Accuracy-coverage radar of KUN-V2 against the eleven learned baselines of Table 2. Each row's masked MSE is min-max normalised linearly between the row's best learned model (floored at 10^{-4}; outer ring) and its worst (inner ring), trivial predictors excluded; a vertex is the mean over that axis' rows. KUN-V2 is one of 6 architectures spanning all five axes (with FNO, U-NO, DeepONet, Transformer, Mamba).

dimensionality-agnostic: decompose a high-dimensional tensor into chunks, mix within and across chunks, and reconstruct the tensor.

The KUN-V2 framework. We instantiate this view as KUN-V2, a follow-up to Kernel-U-Net (You et al., 2024), the prior single-axis-time-series member of the KUN architecture family. KUN-V2 generalizes the family to arbitrary input dimensionality via three components: (i) an auto-factorization that splits each spatial axis into balanced chunks; (ii) a compact *spec language*, namely a short word over the axis letters in which brackets mark joint groups and an optional count repeats a token, which declaratively specifies the order and grouping in which chunks are reduced; and (iii) a UNet-style symmetric decoder with skip connections that mirrors the encoder. Inside each chunk, the mixing primitive is a lightweight linear or MLP kernel. This operation admits one parameterized realization, whose cost is linear in the number of elements and whose processing order is set by the user.

Contributions. We summarize the contributions of this paper as follows.

1. A spec language that, together with an auto-factorization of each axis into balanced chunks, turns the order and grouping in which the subspaces of a high-dimensional input are processed into a user-set string, so that one architecture serves univariate and multivariate time series, 3D and 4D sequences, and higher-dimensional inputs without redesign.
2. A complexity analysis showing that a KUN-V2 module costs $O(E)$ in the element count E for every valid spec, so the choice of schedule does not change the bound, together with a proof that the spec-induced reshape / permutation pipeline is invertible end-to-end.
3. An empirical study of one codebase on a *single task*, masked-half completion, across 21 datasets at $N = 1 \ldots 5$ against fourteen other models: KUN-V2 places in the top three learned models on 17 of them and first on 11, and its measured training time grows at most linearly with the element count, as the complexity bound predicts.

Paper organization. Section 2 fixes notation and Section 3 presents the framework; Section 4 bounds its time complexity and proves the spec-induced reshape / permutation pipeline invertible; Section 5 reports results from one to four dimensions, with a synthetic check at five, and Section 6 concludes. Appendix A surveys prior art and Appendix B discusses higher-dimensional inputs and limitations.

2 PRELIMINARIES AND NOTATION

We write $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C}$ for an input batch with B samples, N spatial axes of sizes $D_1, \ldots, D_N$, and C feature channels. N is treated as a runtime parameter throughout this paper. We omit the batch axis B from notation when it is not load-bearing. For brevity we write $E := \prod_{i=1}^{N} D_i$ for the total number of spatial cells (the input volume), which controls all complexity claims in Section 4.

The batch axis B and channel axis C are fundamentally distinct from the N spatial axes: **(i) batch B.** Samples are i.i.d. in expectation; mixing across B is forbidden by design (it would leak in-

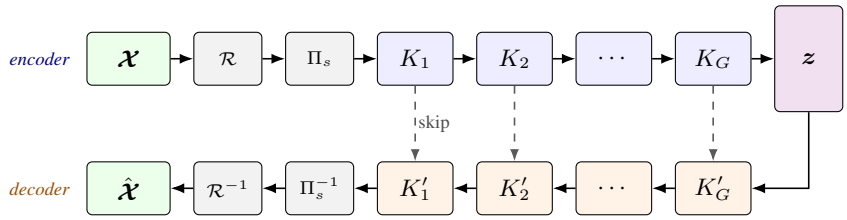


Figure 2: KUN-V2 pipeline. Top row: input tensor $\boldsymbol{\mathcal{X}}$ is auto-factorized ($\mathcal{R}$, Section 3.2), the $M = \sum_i m_i$ chunk axes are reordered by the spec-induced permutation Π_s (Section 3.3), then reduced one group g_t at a time by encoder layers $K_1, \ldots, K_G$ to the latent $\boldsymbol{z}$ of channel dim C_z. Bottom row mirrors this with decoder layers $K'_G, \ldots, K'_1$ followed by Π_s^{-1} and $\mathcal{R}^{-1}$ to restore the original $(D_1, \ldots, D_N)$ axis order (Theorem 2). Vertical dashed arrows are UNet skip connections from K_t to K'_t. Each KERNELWRAPPER K_t is a Linear kernel, that is, a small MLP (Section 3.4).

formation across samples). Operations on B are limited to vectorized per-sample evaluation. **(ii) channel C.** Features at a given spatial cell form a vector that is mixed within each kernel call (Section 3.4); the channel dimension changes from layer to layer along the encoder/decoder schedule $C \to h_1 \to \cdots \to C_z$ but never grows out of the cell. Crucially, *neither B nor C is subject to chunk decomposition*: they receive no chunk multiplicity, the spec language has no axis letter for them, and they do not appear as targets of `[joint]` groups. The spec letters cover only the N spatial axes.

For each axis i, we factorize its size D_i into m_i *chunk factors*:

$$D_i = c_{i,1} \cdot c_{i,2} \cdots c_{i,m_i}, \qquad i = 1, \ldots, N. \tag{1}$$

The vector $\boldsymbol{m} = (m_1, \ldots, m_N)$ of *chunk multiplicities* is a hyperparameter; the factors $\{c_{i,j}\}$ are produced by an auto-factorization that minimizes their standard deviation (Section 3.2). The factor count is $M = \sum_i m_i$. If a spec parses into G groups, the encoder has G layers, one per group; joint groups allow $G \leq M$.

Axis names are lowercase letters (e.g. `h`, `w`, `l`). A *spec string* like `hhhwwwlll`, `h2w2hwl3`, or `[hwl]hwl[hwl]` declares the order and grouping in which chunks are reduced. A bare letter consumes one chunk of that axis; `[hwl]` consumes one chunk from each of `h, w, l` jointly; a digit suffix repeats the preceding token. We denote by $\mathcal{S}_M$ the valid specs that emit exactly M axis-letter occurrences after expansion. A permutation π over chunk indices, induced by the spec string, is applied once at the encoder input and inverted at the decoder output.

Given $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$ and a target $\boldsymbol{Y}$, learn a parameterized map F_Θ such that the architecture and parameter count of F_Θ depend only on $\boldsymbol{m}$, the spec string, the channel dimensions, and the chosen kernel, and not on N itself. KUN-V2 achieves this by construction.

3 THE KUN-V2 FRAMEWORK

3.1 OVERVIEW

A KUN-V2 model is a symmetric encoder–decoder. The encoder takes a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C}$ and progressively reduces every spatial axis to size 1, producing a latent tensor $\boldsymbol{z} \in \mathbb{R}^{B \times 1 \times \cdots \times 1 \times C_z}$. The decoder symmetrically expands $\boldsymbol{z}$ back to the input spatial shape, with UNet-style skip connections from the encoder. Reductions and expansions act *only on the N spatial axes*; the batch axis B is preserved unchanged (per-sample evaluation), and the channel axis C is mixed inside each kernel call but is not chunk-decomposed (Section 2). Reductions and expansions of the spatial axes are performed in *chunks* whose sizes come from an auto-factorization (Section 3.2) and whose order is set by a *spec string* (Section 3.3).

This paper focuses on a single task at every N: *given a tensor with one half of its cells masked out, predict the masked half.* Concretely, we partition the spatial axes into a *visible* region Ω_v and a *masked* region Ω_m of equal volume $|\Omega_m| = E/2$ (e.g. the second half along the leading axis for $N = 1$ forecasting, or a contiguous half-block for $N \geq 2$), feed the input with Ω_m zeroed out, and supervise the decoder output on Ω_m via masked MSE. Because encoder input and decoder output

Algorithm 1: Spec parsing and expansion

function PARSESPEC(s) {e.g. `[hwl]hwl[hwl]`}
  first expand digit repeats: `h2w2hwl3` → `hhwwhwlll`
  scan s left to right
  on `[`: read until `]`, emit a joint group g = inner letters
  on a bare letter ℓ: emit a singleton group $g = \{\ell\}$
  return ordered list $\boldsymbol{g} = (g_1, \ldots, g_G)$
end function

function ASSIGNCHUNKS($\boldsymbol{g}$, $\{c_{i,j}\}$)
  maintain a per-letter queue of remaining chunk factors
  for g_t in $\boldsymbol{g}$: pop one factor per letter $\in g_t$ from its queue
  return per-layer chunk shapes $\{\boldsymbol{s}_t^{\text{in}}\}_{t=1}^{G}$
end function

Algorithm 2: Reshape and permutation

function RESHAPEANDPERMUTE($\boldsymbol{\mathcal{X}}$, $\{c_{i,j}\}$, $\boldsymbol{g}$)
  reshape $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$ to expose chunk factors:
    $\mathbb{R}^{c_{1,1} \times \cdots \times c_{N,m_N} \times C}$
  compute π: position chunks in spec order $\boldsymbol{g}$
  permute: $\boldsymbol{H} \leftarrow \boldsymbol{\mathcal{X}}.\mathrm{permute}(\pi)$
  return $\boldsymbol{H}$
end function

function INVERSEPERMUTEANDRESHAPE($\boldsymbol{H}$)
  apply π^{-1}
  reshape back to $(D_1, \ldots, D_N, C)$
  return reconstructed tensor
end function

share the shape $D_1 \times \cdots \times D_N \times C$, the same KUN-V2 instance handles every N by changing only the spec string and chunk multiplicities; the choice of mask is the only N-specific knob in the experimental protocol (Section 5.2).

3.2 AUTO-FACTORIZATION OF SPATIAL AXES

Given a target chunk multiplicity m_i for axis i, we factorize D_i into m_i integer factors that are as balanced as possible. We solve

$$\{c_{i,j}\}_{j=1,\ldots,m_i} \;=\; \arg \min_{\substack{x_j \in \mathbb{Z}_{>0},\ j=1,\ldots,m_i \\ \prod_{j=1}^{m_i} x_j = D_i}} \mathrm{std}(\{x_j\}_{j=1,\ldots,m_i}), \tag{2}$$

where $\mathrm{std}(\cdot)$ denotes the (sample) standard deviation of the factor tuple; minimising it drives the m_i factors $\{x_j\}$ toward equality, so the chosen decomposition is as balanced as the integer constraint $\prod_j x_j = D_i$ permits (in the extreme case where all x_j coincide, $\mathrm{std} = 0$ and the factors are perfectly balanced). When no such exact balanced factorization exists, we fall back to a *ceiling-product* factorization (the smallest integer tuple with $\prod_j x_j \geq D_i$, whose surplus is truncated at the decoder output). For example, $D = 64$ with $m = 3$ yields $(4, 4, 4)$; $D = 27, m = 3$ yields $(3, 3, 3)$. Balanced factors keep chunk sizes comparable across reductions.

3.3 THE SPEC LANGUAGE

We use *spec* as an abbreviation of *specification*: the *spec language* is the small domain-specific notation in which the user declaratively specifies the per-layer chunk schedule, and a *spec string* is one concrete expression in that language. A spec string declares the order and grouping in which the chunks $\{c_{i,j}\}$ are consumed. Tokens are lowercase axis letters; brackets group axes processed jointly; trailing digits repeat tokens:

`hhhwwwlll`	→	sequential: reduce all h, then all w, then all l
`h2w2hwl3`	→	expand to `hhwwhwlll` (`h2`=`hh`, `l3`=`lll`)
`[hw][hw][hw]lll`	→	repeated joint spatial blocks then sequential

Formally, `parse_spec` (Algorithm 1) returns $\boldsymbol{g} = (g_1, \ldots, g_G)$, an ordered list of G non-empty groups of axis letters. The t-th encoder layer jointly reduces one chunk per letter in g_t. Validity requires $\sum_{t=1}^{G} |g_t| = M = \sum_i m_i$, with each axis letter occurring exactly m_i times. Since each spatial axis is named by one lowercase letter, the 26-letter alphabet caps the spec language at $N \leq 26$ axes, which is well beyond any dimensionality of practical interest.

3.4 ENCODER AND DECODER

The encoder is a chain of G *KernelWrapper* layers, one per spec group g_t. Writing $g_t(k)$ for the (axis-index, chunk-position) pair indexed by the k-th slot of g_t, layer t takes a tensor whose axes

Algorithm 3: KUN-V2 forward pass

Require: Input $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$; chunk multiplicities $\boldsymbol{m}$; spec string s; kernel choice K; latent dim C_z; hidden schedule $\{h_t\}$; UNet skip flag.

$\{c_{i,j}\} \leftarrow \textsc{AutoFactorize}(D_i, m_i)$ for $i = 1, \ldots, N$ {Eq. 2}
$\boldsymbol{g} \leftarrow \textsc{ParseSpec}(s)$ {Alg. 1}
$\boldsymbol{H}_{\text{enc}}^{(0)} \leftarrow \textsc{ReshapeAndPermute}(\boldsymbol{\mathcal{X}}, \{c_{i,j}\}, \boldsymbol{g})$ {Alg. 2}
Encoder:
for $t = 1, \ldots, G$ **do**
  $\boldsymbol{H}_{\text{enc}}^{(t)} \leftarrow \text{KernelWrapper}_{\text{enc}}^{K_t}(\boldsymbol{H}_{\text{enc}}^{(t-1)})$ $\{\boldsymbol{s}_t^{\text{in}} \to (1, \ldots, 1)\}$
  if skip **then** cache $\boldsymbol{S}^{(t)} \leftarrow \boldsymbol{H}_{\text{enc}}^{(t-1)}$
end for
$\boldsymbol{z} \leftarrow \boldsymbol{H}_{\text{enc}}^{(G)} \in \mathbb{R}^{1 \times \cdots \times 1 \times C_z}$
Decoder:
$\boldsymbol{H}_{\text{dec}}^{(G)} \leftarrow \boldsymbol{z}$ {decoder latent (mirror of $\boldsymbol{H}_{\text{enc}}^{(G)}$)}
for $t = G-1, \ldots, 0$ **do**
  $\boldsymbol{H}_{\text{dec}}^{(t)} \leftarrow \text{KernelWrapper}_{\text{dec}}^{K_{t+1}}(\boldsymbol{H}_{\text{dec}}^{(t+1)})$
  if skip **then** $\boldsymbol{H}_{\text{dec}}^{(t)} \mathrel{+}= \boldsymbol{S}^{(t+1)}$
end for
$\hat{\boldsymbol{\mathcal{X}}} \leftarrow \textsc{InversePermuteAndReshape}(\boldsymbol{H}_{\text{dec}}^{(0)})$
return $\hat{\boldsymbol{\mathcal{X}}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$

```python
import torch
from kun_lib_v2 import build_model

# (B, L, H, W, C): time first, then 2 spatial
model = build_model(
    input_shape=(64, 8, 8),
    shape_name="LHW",
    latent_shape=(1, 1, 1),
    chunk_multiples=(3, 3, 3),
    chunk_names="[hw][hw][hw]lll",
    input_dim=1,
    latent_dim=64,
    hidden_dim=64,
    kernel="linear",
    unet_skip=True,
)
y3 = model(torch.randn(32, 64, 8, 8, 1))
# y3.shape == (32, 64, 8, 8, 1)
```

Listing 1: A complete KUN-V2 call site at $N = 3$, on a $(B, L, H, W, C) = (32, 64, 8, 8, 1)$ tensor: $\boldsymbol{m} = (3, 3, 3)$ splits each axis into three chunks ($64 \to 4{\cdot}4{\cdot}4$, $8 \to 2{\cdot}2{\cdot}2$), consumed in the order the spec declares.

for letters in g_t have current chunk size $\prod_{k=1}^{|g_t|} c_{g_t(k)}$ and reduces them to size 1, while the channel dimension grows from C (input) to C_z (latent) along a per-layer schedule $C \to h_1 \to \cdots \to C_z$. Specifically, for layer t with input chunk shape $\boldsymbol{s}_t^{\text{in}}$ and output chunk shape $\boldsymbol{s}_t^{\text{out}} = (1, \ldots, 1)$, channel dims $(d_t^{\text{in}}, d_t^{\text{out}})$, and chosen kernel K_t:

$$\boldsymbol{H}^{(t)} = \text{KernelWrapper}_{K_t}\big(\boldsymbol{H}^{(t-1)};\ \boldsymbol{s}_t^{\text{in}} \to \boldsymbol{s}_t^{\text{out}},\ d_t^{\text{in}} \to d_t^{\text{out}}\big). \tag{3}$$

The decoder mirrors the G encoder layers in reverse order: decoder step t' expands the axes consumed by encoder group $g_{G-t'+1}$, with the channel schedule reversed.

Writing $s_t = \prod_{k=1}^{|g_t|} c_{g_t(k)}$ for the scalar chunk size consumed at layer t (with $g_t(k)$ as defined above), the spatial element count entering layer $t+1$ is

$$V_t = \frac{E}{\prod_{t' \le t} s_{t'}}, \qquad V_0 = E. \tag{4}$$

When every group consumes a nontrivial chunk ($s_t \ge 2$), $V_t \le V_{t-1}/2$. This geometric decay is the condition used in Theorem 1.

At encoder layer t, the input is cached and added after the decoder has expanded the corresponding group:

$$\boldsymbol{H}_{\text{dec}}^{(t-1)} \mathrel{+}= \boldsymbol{H}_{\text{enc}}^{(t-1)}, \qquad t = 1, \ldots, G. \tag{5}$$

The full forward pass is given in Algorithm 3.

A KernelWrapper maps a chunk-shaped sub-tensor to the target chunk shape and channel dim via a kernel module K_t. We use the *Linear kernel*, a dense network that flattens a chunk of shape $(\boldsymbol{s}^{\text{in}}, d^{\text{in}})$ and maps it to $(\boldsymbol{s}^{\text{out}}, d^{\text{out}})$. Its interior widths may be set to a fixed value or interpolated between the flattened input and output widths. The $O(s)$ per-call bound in Section 4 assumes a fixed number of interior layers and widths independent of chunk size s (and fixed channel dimensions); it does not apply in general to interpolated widths, which can grow with s. The kernel is applied to non-overlapping chunks, using the same chunk tiling as an operator whose kernel and stride equal $\boldsymbol{s}^{\text{in}}$.

3.5 PERMUTATION AND RESHAPE

The spec string also induces a permutation π over the chunk axes, applied once at the encoder input and inverted at the decoder output (Algorithm 2). Both directions are pure `view`/`permute`

operations, so they move no data and contribute no arithmetic to the cost model of Section 4, and they are exact inverses by construction (Theorem 2, proved in Appendix C). The spec is therefore free to reorder axes at will: nothing downstream has to compensate.

Listing 1 gives a complete 3D call site. Four arguments carry the construction of Sections 3.2 and 3.3: `input_shape` is the spatial shape $(D_1, \ldots, D_N)$, `shape_name` gives those axes the letters the spec refers to, `chunk_multiples` is $\boldsymbol{m}$, and `chunk_names` is the spec string. The remaining arguments set capacity: the channel dims C and C_z, the per-layer schedule $\{h_t\}$, the kernel and the skip flag. Retargeting to another rank therefore changes only the first four.

4 Theoretical Properties

Two results license the spec string as a pure architectural knob: changing it does not change the asymptotic cost under the assumptions of Theorem 1, or the values preserved by the reshape/permutation steps. We state them here; the proofs, together with the auto-factorization fallback case, are in Appendix C.

4.1 Complexity

Theorem 1 (Complexity of a KUN-V2 module)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$ have spatial volume $E = \prod_{i=1}^{N} D_i$. Auto-factorization splits each axis into m_i chunks with $\prod_j c_{i,j} = D_i$, and a valid spec schedules the $M = \sum_i m_i$ chunks across G encoder layers, layer t consuming a chunk of volume s_t with $\prod_{t=1}^{G} s_t = E$. Assume $s_t \geq 2$ for every group. Write $T_K(s)$ for an upper bound on the cost of one encoder or corresponding decoder kernel call at chunk size s (channel widths are architectural hyperparameters, absorbed into the constant) and $V_t = E / \prod_{t' \leq t} s_{t'}$ for the element count entering layer $t+1$ (Eq. (4)). Then a single encoder–decoder module has arithmetic cost bounded by*

$$T(\boldsymbol{\mathcal{X}}) \;\leq\; 2 \sum_{t=1}^{G} V_t \, T_K(s_t), \tag{6}$$

where the factor 2 accounts for the symmetric decoder, and with the Linear kernel ($T_K(s) = O(s)$), this tightens to

$$T(\boldsymbol{\mathcal{X}}) \;=\; O(E), \tag{7}$$

linear in the total spatial volume *and independent of the schedule under these assumptions. With a fixed number of fixed-width interior layers (or none), $T_K(s) = O(s)$; interpolated interior widths can grow with s and are not covered by this bound.*

4.2 Reshape/Permutation Invertibility

Theorem 2 (Invertibility of the spec-induced reshape / permutation pipeline)**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C}$ with auto-factorization $D_i = \prod_{j=1}^{m_i} c_{i,j}$ (Eq. (2)) and $M = \sum_i m_i$ chunk axes, and let s be a valid spec in the sense of Algorithm 1: the parsed groups consume every chunk factor exactly once. Let $\mathcal{R}$ be the auto-factorized reshape and Π_s the permutation of the M chunk axes induced by s (Algorithm 2). Then $\mathcal{R}$ and Π_s are bijections, so the encoder front-end $\Phi_s = \Pi_s \circ \mathcal{R}$ has the inverse $\Phi_s^{-1} = \mathcal{R}^{-1} \circ \Pi_s^{-1}$ applied by the decoder back-end, and*

$$\Phi_s^{-1} \circ \Phi_s = \mathrm{id}_{\mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C}}. \tag{8}$$

In particular, with every kernel set to the identity the full KUN-V2 *round trip returns each cell to its original multi-index $(b, i_1, \ldots, i_N, c)$.*

Corollary 3 (Shape compressibility and restoration)**.** *Under the same hypotheses, the encoder reduces every spatial axis to size 1 in G layers, one per parsed group, and the symmetric decoder restores the original sizes,*

$$\mathcal{E}_s : \mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C} \to \mathbb{R}^{B \times 1 \times \cdots \times 1 \times C_z}, \qquad \mathcal{D}_s : \mathbb{R}^{B \times 1 \times \cdots \times 1 \times C_z} \to \mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C},$$

so $\mathcal{D}_s \circ \mathcal{E}_s$ is well-typed on $\mathbb{R}^{B \times D_1 \times \cdots \times D_N \times C}$ and collapses to the identity when every KERNEL-WRAPPER *is the identity.*

The three results pin down what the spec could affect: cost, correctness, and shape. Theorem 1 controls *cost*: with fixed kernel interior widths, the per-call $O(s_t)$ cancels the $1/s_t$ in the call count, leaving a geometric series when every group has $s_t \geq 2$. Every valid schedule covered by the theorem therefore attains the same $O(E)$ bound, linear in the input cells and without an N or M prefactor. Theorem 2 controls *correctness*: reordering chunks is a scheduling device and does not destroy information. Corollary 3 closes the loop on *shape*: the encoder/decoder pair is well-typed end-to-end for any valid spec. The spec can therefore be tuned for accuracy without changing the asymptotic cost model within those conditions.

Table 1: **Cost versus rank.** Parameter counts and time complexity. KUN-V2 keeps a comparatively small parameter count at every rank, and its time complexity is linear in E. *Italic*: out of memory, count computed analytically.

Model	Time	N				
		1	2	3	4	5
		$E=64$	512	4K	33K	262K
KUN-V2	$O(E)$	23.3K	46.3K	54.7K	71.6K	105K
Mamba	$O(E)$	20.1K	20.6K	24.2K	53.3K	286K
DeepONet	$O(E)$	8.48K	22.8K	138K	1.06M	8.40M
FNO	$O(E \log E)$	50.0K	210K	800K	3.16M	12.6M
U-NO	$O(E \log E)$	58.3K	177K	410K	1.10M	3.36M
Linear	$O(E^2)$	4.16K	263K	16.8M	*1.07B*	*68.7B*
MLP	$O(E^2)$	8.32K	525K	33.6M	*2.15B*	*137B*
Transformer	$O(E^2)$	27.6K	41.9K	157K	*1.07M*	*8.41M*

5 EXPERIMENTS

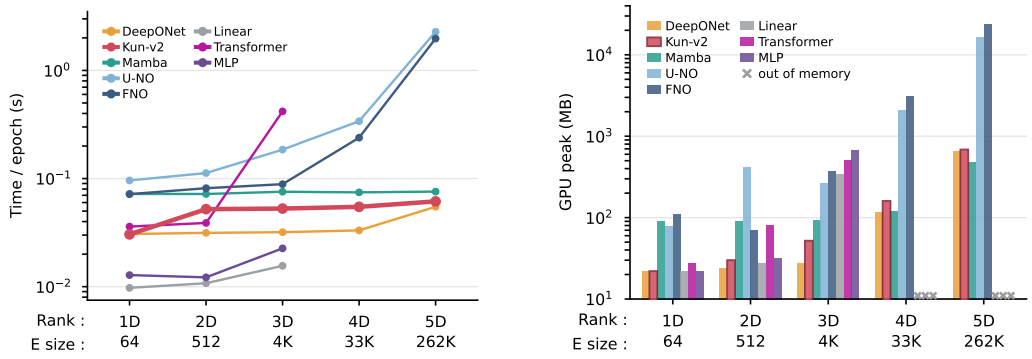
Four studies support the claims of this paper, each varying one thing and all running the task of Sec. 5.2. **(i) Cost versus rank** (Table 1, Fig. 3) raises only N on the sine field, testing whether the $O(E)$ bound of Theorem 1 holds on a real GPU; it measures only cost, namely trainable parameters, training time per epoch and peak GPU memory, and not accuracy. **(ii) The cross-model benchmark** (Table 2) compares KUN-V2 against every model of Sec. 5.1 on every dataset, asking whether one spec-driven architecture stays competitive across ranks and data types without per-rank redesign; it measures accuracy as test masked MSE (Eq. (9)) at the epoch the validation split selects. **(iii) Spec ablations** (Appendix G) vary only the spec string, asking whether the better spec is a property of the architecture or of the data; **(iv) capacity ablations** (Appendix G) vary depth, width and the kernel interior, asking whether capacity has saturated. Both read the same test masked MSE as (ii).

5.1 DATASETS AND MODELS

Two synthetic families, generated in-repo, differ in exactly one property: whether the pattern moves. The *sine field* is a sum of one or two sinusoids with integer frequencies on the normalised grid, so every sample is exactly periodic and its masked half is a deterministic function of the visible half; the pattern stands still and can be completed by continuing it in place. The *wavelet bank* sweeps one packet, a Ricker, Gaussian-derivative or Daubechies-4 wavelet, across the field along a spatial direction at a fixed speed, so the masked half cannot be completed without reading the packet's velocity off the visible half. The two are run at matched shapes wherever possible, which makes a sine row and its wavelet row a controlled contrast between a standing and a travelling pattern (Appendix D.1, Appendix D.2).

Five kinds of real data complete the $N = 1 \ldots 4$ rows (the two synthetic families add one $N = 5$ row each, which serves only as a check of accuracy and efficiency; 21 rows in all): PDE fields (Burgers, Darcy, Navier–Stokes (Li et al., 2021)), reanalysis (ERA5 (Hersbach et al., 2020)), a standardised multivariate series (ETTh1/ETTh2 (Zhou et al., 2021), entering at $N = 1$ and 2), video (Moving MNIST (Srivastava et al., 2015)) and rasterised trajectories (GeoLife (Zheng et al., 2010), KITTI (Geiger et al., 2012)), the last entering as fields rather than coordinate lists because N counts tensor rank. Each dataset holds S samples, split $0.7/0.1/0.2$ into train, validation and test. Appendix D gives the generators, the shape and S of every row (Table 3), and records the three rows re-drawn after a leakage audit, which are not comparable to earlier numbers of the same name.

Fifteen models run on each row: KUN-V2; eleven learned baselines covering the three families a rank-agnostic model has to face, namely sequence models (a linear matrix and an NLinear-style MLP (Zeng et al., 2023), Mamba (Gu & Dao, 2024), a dense Transformer (Vaswani et al., 2017), PatchTST (Nie et al., 2023)), video predictors (ConvLSTM (Shi et al., 2015), PredRNN (Wang et al., 2017), SimVP (Gao et al., 2022)) and neural operators (FNO (Li et al., 2021), U-NO (Rahman et al., 2023), DeepONet (Lu et al., 2021a)); and the three untrained predictors Zero, Last and Copy, carried



(a) Training time per epoch, first epoch dropped; a line stops where its model runs out of memory.

(b) Peak GPU memory; bars keep one order in every group, the five models that reach rank 5 before the three that do not.

Figure 3: **Measured cost on the synthetic sine field, rank 1 to rank 5.** Same runs and protocol as Table 1; log axes; E grows $4096\times$. At rank 5, KUN-V2 (0.061 s / 686 MB), DeepONet and Mamba sit in one band, while the two spectral operators are 30–40$\times$ slower and reach 24.2 and 16.6 GB. x: out of memory.

as floors. A model enters only at the ranks where it reads a rank-N input without flattening away the adjacency the rank is about, which is why PatchTST stops at $N = 2$.

5.2 EXPERIMENTAL SETUP

We choose *masked-half completion along the leading (temporal) axis* as the core task of this paper: for a sample $\boldsymbol{\mathcal{X}}^{(s)} \in \mathbb{R}^{L\times D_2\times\cdots\times D_N}$ we zero the second half of the leading axis, $\Omega_m = \{[L/2{+}1, L]\} \times \prod_{n\geq 2}[1, D_n]$, and supervise $\hat{\boldsymbol{\mathcal{X}}} = f(\tilde{\boldsymbol{\mathcal{X}}})$ on Ω_m alone through masked MSE,

$$\mathcal{L}(\hat{\boldsymbol{\mathcal{X}}}, \boldsymbol{\mathcal{X}}; \Omega_m) \;=\; \frac{1}{|\Omega_m|} \sum_{\boldsymbol{i}\in\Omega_m} \big(\hat{\boldsymbol{\mathcal{X}}}[\boldsymbol{i}] - \boldsymbol{\mathcal{X}}[\boldsymbol{i}]\big)^2. \tag{9}$$

The same rule instantiates univariate forecasting at $N = 1$, multivariate forecasting at $N = 2$, and frame- and voxel-sequence prediction at $N = 3, 4$, so one objective carries unchanged across every rank, dataset and baseline below, and no rank needs a task of its own.

5.3 MAIN RESULTS: ACCURACY AND COST VS. N

Table 1 and Fig. 3 report the cost comparison, and the reading is simple: of the eight models, KUN-V2 is the only one that stays cheap on all three axes as the rank climbs. Over a $4096\times$ growth in E its parameter count grows $4.5\times$ (23.3K $\rightarrow$ 105K), the smallest growth of the eight models, while its time per epoch grows $2.0\times$ and its peak memory $31\times$, ending at 0.061 s and 686 MB, which is the linear-in-E behaviour Theorem 1 predicts. Every alternative pays for its own flatness somewhere: Mamba is flatter in time only because it folds each spatial axis into the channel dimension of one length-64 scan, DeepONet's branch net reads the whole field so its parameter count grows $991\times$, the two spectral operators end at 24.2 and 16.6 GB, and the two $O(E^2)$ families, dense Linear/MLP mixing and per-grid-point attention, cannot be run beyond $N = 3$.

Table 2 compares fifteen models on 21 dataset rows; since every dataset carries its own target scale, the marks compare models *along a row* only. Figure 1 condenses the ten synthetic rows (sine and wavelet) in three steps. On each row, a learned model with masked MSE e scores $(e_{\max}-e)/(e_{\max}-\max(e_{\min}, 10^{-4}))$, capped at 1, where $e_{\min}$ and $e_{\max}$ are the row's best and worst learned MSE (Zero, Last and Copy excluded), so the best model scores 1 and the worst 0. A model's vertex on the N axis is the mean of its scores over the rows of that rank, and a model is drawn only on the ranks where the table has a number. The mean is plotted at radius $0.3 + 0.7\times$ score, so the inner ring marks the worst learned model and the outer ring the best. KUN-V2 places in the top three

Table 2: Masked-half completion, $N = 1 \dots 4$ (synthetic rows also at $N = 5$): test masked MSE at the validation-selected epoch, *mean over seeds* 2026, 2016, 2006; standard deviations in Table 5. **Bold**/underline: best/second learned model *in a row* by mean; scales differ across rows. $^{(n)}$: only n of the three seeds so far. −: not applicable; ·: not run yet. Per-cell settings: Appendix E.

N	task	KUN	FNO	U-NO	DeepONet	Linear	MLP	Transformer	Mamba	PatchTST	ConvLSTM	PredRNN	SimVP	Zero	Last	Copy
1	1d_sine	$\mathbf{1.52\times10^{-4}}$	0.4456	0.1306	0.3784	0.4621	0.6284	0.4614	1.0787	0.0980	–	–	–	0.9510	0.9630	0.9370
2	2d_sine	$\mathbf{1.13\times10^{-5}}$	0.1485	0.2587	0.4756	0.0282	0.4999	0.5962	0.8426	0.1825	–	–	–	1.2913	1.0287	1.2320
3	3d_sine	$\mathbf{3.29\times10^{-4}}$	0.1838	0.3168	0.4862	0.0392	0.2606	0.5338	0.5045	–	0.3917	0.3997	0.5543	0.9517	1.0141	0.9038
4	4d_sine	$\mathbf{6.85\times10^{-4}}$	0.1865	0.3679	0.7053	–	–	0.6070	0.6733	–	–	–	–	0.9969	1.0035	0.9396
5	5d_sine	$\mathbf{6.51\times10^{-5}}$	0.2599	$0.5817^{(1)}$	0.9256	–	–	0.9392	0.9543	–	–	–	–	0.9890	1.0599	0.9554
1	1d_wavelet	6.83×10^{-4}	5.88×10^{-4}	6.27×10^{-4}	0.0241	0.0204	0.0049	0.0011	0.0031	$\mathbf{3.64\times10^{-4}}$	–	–	–	0.6685	5.6536	2.0193
2	2d_wavelet	**0.0029**	0.0108	0.0325	0.6474	0.0125	0.0324	0.1133	0.2394	0.6028	–	–	–	1.0010	1.9729	2.0019
3	3d_wavelet	$\mathbf{8.47\times10^{-4}}$	0.0053	0.0081	0.7024	0.0233	0.4725	0.0854	0.2193	–	0.1260	0.0868	0.0398	0.9990	2.0476	1.9979
4	4d_wavelet	0.0029	**0.0023**	0.0057	0.8644	–	–	0.1051	0.0852	–	–	–	–	1.0003	2.0018	2.0006
5	5d_wavelet	0.0786	$\mathbf{0.0509}^{(1)}$	0.1324	$0.9996^{(1)}$	–	–	0.8357	0.8389	–	–	–	–	0.9995	2.0225	1.9990
1	1d_burgers	**0.1111**	0.1237	0.1156	0.1709	0.2103	0.1281	0.2568	0.1377	0.1178	–	–	–	0.9968	2.2079	3.4388
2	2d_darcy	0.3805	**0.3367**	0.3981	0.4902	0.4022	0.4041	0.5738	0.4631	0.4542	–	–	–	0.9608	1.0683	2.0447
3	3d_ns	0.1533	**0.0639**	0.0923	0.2637	0.2285	0.1929	0.1442	0.1585	–	0.0664	0.1615	0.1548	1.6382	0.8612	1.1957
1	1d_ETTh1	**0.0649**	0.1170	0.1227	0.0764	0.0832	0.0926	0.1040	0.1166	0.0919	–	–	–	1.0842	0.0738	0.1648
1	1d_ETTh2	0.1210	0.1522	0.1538	0.1989	**0.1198**	0.1364	0.1812	0.2137	0.1567	–	–	–	1.0317	0.3160	0.4002
2	2d_ETTh1	0.3342	0.5605	0.5308	0.4442	0.3397	**0.3239**	0.6058	0.5467	0.3616	–	–	–	0.9873	0.8686	1.1226
2	2d_ETTh2	**0.2356**	0.4409	0.3967	0.4231	0.2769	0.2672	0.3158	0.2721	0.2760	–	–	–	1.3169	0.2682	0.3946
3	3d_geolife	0.6242	0.6253	**0.6065**	0.6457	0.6643	0.6433	0.6805	0.7530	–	0.6612	0.6801	0.6181	0.9577	1.8482	1.9561
3	3d_mnist	0.8832	0.7795	0.7689	0.8893	0.9268	0.8848	0.9351	0.8546	–	0.7669	**0.6880**	0.8699	1.0789	1.5547	1.9375
4	4d_era5	**0.2136**	0.2162	0.2213	0.2224	–	–	0.2163	0.2148	–	–	–	–	1.0190	0.2551	0.3407
4	4d_kitti	1.1633	**1.0126**	1.0525	1.0710	–	–	1.2019	1.1082	–	–	–	–	1.3173	1.1186	1.8200

learned models on 17 of the 21 rows and takes the row outright on 11, spread across the ranks rather than concentrated at $N = 1$, and on the four rows where it falls outside the top three it is still only $1.15\times$ to $2.40\times$ from the winner. That is the claim this paper makes, one spec-driven architecture competitive at every rank without per-rank redesign, and not that it dominates everywhere: where it loses, FNO is the most frequent winner, taking five of those ten rows, and PatchTST, Linear, MLP, U-NO and PredRNN take one each. On the travelling wavelet bank KUN-V2 takes $N = 2$ and 3 but loses $N = 1$ to PatchTST and $N = 4, 5$ to FNO (one seed so far at $N = 5$), which is the clearest dataset-shaped limit here: a packet whose completion is a phase shift is what a spectral operator is built for. Every row is won by a learned model, none by Zero, Last or Copy, but the margin over those floors separates the regimes. It ranges from $20\times$ to about five orders of magnitude on the ten synthetic rows, but falls below $1.3\times$ on `4d_kitti`, `4d_era5`, `1d_ETTh1` and `2d_ETTh2`, which therefore measure persistence about as much as forecasting skill.

The ablation shows that width and depth increase the model's capacity: on the sine field, widening from 32 to 128 channels lowers the error by $14\times$ to $387\times$, and going from one block to two by $3.6\times$ to $766\times$ (Appendix G). The spec instead changes the structure, that is, the order and grouping in which the axes are reduced, and thus lets KUN-V2 find the model that suits the data: no spec wins at every rank, and the best one changes with the dataset.

6 CONCLUSION

We presented KUN-V2, an encoder–decoder framework that processes inputs of arbitrary spatial dimensionality through an auto-factorization of each axis into balanced chunks and a compact spec language that schedules their reduction. By making the order and grouping of chunk reductions a spec string, KUN-V2 lets the user choose how subspaces are processed without changing the architecture. We evaluated it on 21 datasets, comprising synthetic and real data at $N = 1 \dots 4$ and synthetic data at $N = 5$, all served by one codebase in which only the input shape, spec string and chunk multiplicities change. KUN-V2 places in the top three learned models on 17 of them and is the best on 11. Its time complexity is linear in the element count for every valid spec, and on the synthetic cost ladder (ranks 1 to 5) its measured training time and peak memory grow far more slowly than the element count itself. KUN-V2 thus reduces the cost of moving to a new dimensionality from architectural re-design to a configuration change. Future work includes more complex data and other types of learning tasks beyond masked completion.

REFERENCES

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Hehe Fan, Yi Yang, and Mohan Kankanhalli. Point 4D transformer networks for spatio-temporal modeling in point cloud videos. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. SlowFast networks for video recognition. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

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 (NeurIPS)*, volume 37, 2024.

Zhangyang Gao, Cheng Tan, Lirong Wu, and Stan Z. Li. SimVP: Simpler yet better video prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3170–3180, 2022.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361, 2012.

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 (ICML)*, pp. 16115–16152, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024. arXiv:2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, Adrian Simmons, Cornel Soci, Saleh Abdalla, Xavier Abellan, Gianpaolo Balsamo, Peter Bechtold, Gionata Biavati, Jean Bidlot, Massimo Bonavita, Giovanna De Chiara, Per Dahlgren, Dick Dee, Michail Diamantakis, Rossana Dragani, Johannes Flemming, Richard Forbes, Manuel Fuentes, Alan Geer, Leo Haimberger, Sean Healy, Robin J. Hogan, Elías Hólm, Marta Janisková, Sarah Keeley, Patrick Laloyaux, Philippe Lopez, Cristina Lupu, Gabor Radnoti, Patricia de Rosnay, Iryna Rozum, Freja Vamborg, Sebastien Villaume, and Jean-Noël Thépaut. The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51 (3):455–500, 2009.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research (JMLR)*, 24(89):1–97, 2023.

Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. MViTv2: Improved multiscale vision transformers for classification and detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

Jiuming Liu, Jinru Han, Lihao Liu, Angelica I. Aviles-Rivero, Chaokang Jiang, Zhe Liu, and Hesheng Wang. Mamba4D: Efficient 4D point cloud video understanding with disentangled spatial-temporal state space models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. arXiv:2405.14338.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021b.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, et al. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, 2023.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning (ICML)*, 2023.

Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research (TMLR)*, 2023. arXiv:2204.11127.

Stephan Rasp, Stephan Hoyer, Alexander Merose, Ian Langmore, Peter Battaglia, Tyler Russell, Alvaro Sanchez-Gonzalez, Vivian Yang, Rob Carver, Shreya Agrawal, Matthew Chantry, Zied Ben Bouallegue, Peter Dueben, Carla Bromberg, Jared Sisk, Luke Barrington, Aaron Bell, and Fei Sha. WeatherBench 2: A benchmark for the next generation of data-driven global weather models. *Journal of Advances in Modeling Earth Systems (JAMES)*, 16(6):e2023MS004019, 2024.

Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with Einstein-like notation. In *International Conference on Learning Representations (ICLR)*, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *International Conference on Machine Learning (ICML)*, pp. 843–852, 2015.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Cheng Tan, Siyuan Li, Zhangyang Gao, Wenfei Guan, Zedong Wang, Zicheng Liu, Lirong Wu, and Stan Z. Li. OpenSTL: A comprehensive benchmark of spatio-temporal predictive learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. MaxViT: Multi-axis vision transformer. In *European Conference on Computer Vision (ECCV)*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008, 2017.

Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S. Yu. PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 879–888, 2017.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. TimesNet: Temporal 2D-variation modeling for general time series analysis. In *International Conference on Learning Representations (ICLR)*, 2023.

Jiang You, Arben Cela, René Natowicz, Jacob Ouanounou, and Patrick Siarry. Kernel-U-Net: Multivariate time series forecasting using custom kernels. In *2024 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pp. 1–8, 2024. doi: 10.1109/INISTA62901.2024.10683862.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *AAAI Conference on Artificial Intelligence*, 2023.

Yu Zheng, Xing Xie, and Wei-Ying Ma. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin*, 33(2):32–39, 2010.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI Conference on Artificial Intelligence*, 2021.

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 (NeurIPS)*, 2023.

AI USE STATEMENT

The architecture and the spec language, the two theorems and their proofs, the design of the benchmark and of the experimental protocol, and the interpretation of the reported results are the authors' own work. We used generative AI tools in constructing the synthetic datasets, in cleaning the datasets, and in qualitative analysis. We also used them for two tasks with recommended disclosure: drafting and editing parts of the manuscript for readability, and suggesting experimental parameters for a subset of the baseline models. All AI-assisted text was checked line by line against the underlying results by the authors. We have reviewed all AI-assisted work. The authors take responsibility for the final content of this work, including all text, claims, and artifacts.

Ethics Statement

This work studies sequence prediction on synthetic tensor fields and on public benchmark data, and involves no human subjects, no personally identifying information, and no newly released dataset. We see no direct path from the architecture itself to harmful application beyond the risks already carried by the general-purpose forecasting and video-prediction models it is compared against. All baselines run through the same codebase, loss, folds and normalisation as KUN-V2; their budgets and widths are not all equal, and every difference is reported in Appendix E.

Reproducibility Statement

The code for KUN-V2, the baselines and every experiment is provided. The main experimental parameters, including the training settings, per-round budgets and model configurations, are given in Appendix E.

A Related Work

A recent line of general-purpose time-series models handles forecasting, imputation, classification and anomaly detection within one framework. TimesNet (Wu et al., 2023) folds a series by its dominant periods into 2D variations and processes them with one task-general backbone; GPT4TS (Zhou et al., 2023) reuses a largely frozen pre-trained language model as that backbone; UniTS (Gao et al., 2024) casts the tasks as prompted token generation in a single multi-task model; and MOMENT (Goswami et al., 2024) pre-trains a Transformer by masked reconstruction of time-series patches and adapts it to downstream tasks. Several of these reduce imputation and anomaly detection to reconstructing the hidden part of the input, which is the completion view this paper adopts. All of them, however, operate on univariate or multivariate series, a tensor of rank 1 or 2; a video, a PDE field or a 4D reanalysis cube has no place in their input layout.

Long-horizon time-series forecasting has produced Informer (Zhou et al., 2021), Autoformer (Wu et al., 2021), PatchTST (Nie et al., 2023), and the linear-baseline NLinear (Zeng et al., 2023), alongside the broader 1D-sequence family of Transformers (Vaswani et al., 2017), S4 (Gu et al., 2022), Mamba (Gu & Dao, 2024), RWKV (Peng et al., 2023), RetNet (Sun et al., 2023) and Hyena (Poli et al., 2023). KUN-V2 reduces to a 1D encoder/decoder when $N = 1$ (Section 3.4).

For multivariate time series, TimesNet (Wu et al., 2023) reshapes 1D series into 2D via period-folding, while NLinear-multivariate (Zeng et al., 2023) and PatchTST-multivariate (Nie et al., 2023) extend their 1D variants to a (time $\times$ variable) layout. For 2D images, Vision Transformers (Dosovitskiy et al., 2021), Swin (Liu et al., 2021), ConvNeXt (Liu et al., 2022), MLP-Mixer (Tolstikhin et al., 2021), and MaxViT (Tu et al., 2022) use patchification and hand-coded multi-axis attention. Each line of work commits to its specific 2D layout (time $\times$ variable, or height $\times$ width).

ConvLSTMs (Shi et al., 2015) stack a recurrent cell over per-frame 2D feature maps, while SlowFast (Feichtenhofer et al., 2019) uses parallel pathways at different temporal rates; ViViT (Arnab et al., 2021) factorizes spatio-temporal attention along axes; MViTv2 (Li et al., 2022) introduces multiscale pooling tailored to video. Video *prediction*, which is the masked-half task of Section 5 as opposed to recognition, is carried by the same (T, H, W) commitment: PredRNN (Wang et al., 2017) threads a second, zigzag memory through a stack of spatio-temporal LSTM cells, and SimVP (Gao et al., 2022) drops recurrence entirely for a CNN encoder / translator / decoder over the stacked frames. Each new axis requires a new tokenization, positional scheme, and attention factorization.

Beyond 3D, learning architectures specialize further. Point 4D Transformer (Fan et al., 2021) combines 4D convolution with attention for point-cloud video; Mamba4D (Liu et al., 2025) disentangles spatial and temporal state-space models; dense 4D Transformers (Vaswani et al., 2017) apply full self-attention over the flattened 4D token grid as the quadratic upper bound. Each is a fresh architecture tailored to its target dimensionality.

A separate line learns maps between function spaces rather than between one fixed grid and another (Kovachki et al., 2023), and is N-agnostic by construction. DeepONet (Lu et al., 2021a) splits the map into a branch net that reads the input function at fixed sensors and a trunk net that reads the

query coordinate, so the output grid is decoupled from the input one; the branch net nevertheless reads the whole field, so its parameter count is $\Theta(E)$ and grows with every added axis (Table 1). FNO (Li et al., 2021) and its U-shaped variant UNO (Rahman et al., 2023) instead parameterize the operator in the Fourier domain, at $O(E \log E)$ per layer. These three enter Table 2 at every rank from $N = 1$ to 5 unchanged, as do the Transformer and Mamba, which fold every non-temporal axis into the channel dimension of one sequence along the masked axis. None of these five, however, exposes any control over *which* axes a layer acts on or in what order. DeepONet has no spatial mixing operator at all, the two spectral models transform every axis in every layer, and the two sequence models never mix a spatial axis on its own.

Classical multilinear algebra factorizes a rank-N tensor along its axes, as in CP, Tucker and their relatives (Kolda & Bader, 2009), and Einstein-summation tools such as `einsum` (Harris et al., 2020) and einops (Rogozhnikov, 2022) name those axes with letter patterns much like a spec string. What none of them provides is the object KUN-V2 is built around: a learned model on an N-axis tensor in which the subspaces to be processed, and the order in which they are processed, can be chosen freely. A decomposition fixes its factor structure and fits it to one data tensor; an `einsum` contraction order changes only the cost, never the result; and an einops pattern describes a single parameter-free rearrangement. A spec string instead selects any sequence of subspaces, single axes or bracketed groups of axes, each mixed by a learned kernel, so that changing the order changes the model while its cost stays $O(E)$ for every valid spec (Theorem 1).

The encoder/decoder structure of KUN-V2 draws on the UNet skeleton (Ronneberger et al., 2015) and directly extends Kernel-U-Net (You et al., 2024), which introduced custom-kernel chunk reductions for 1D multivariate time-series forecasting. Compared to standard UNets, KUN-V2 (i) generalizes to arbitrary N rather than 1D or 2D, (ii) replaces fixed pool/upsample with auto-factorized chunk reductions, and (iii) exposes a spec language to schedule axis processing order and grouping. Compared to Kernel-U-Net, KUN-V2 adds the spec language, axis auto-factorization, and a complexity analysis together with an invertibility result for the reshape / permutation pipeline.

B DISCUSSION

KUN-V2 delivers a single codebase that retargets to a new dimensionality by changing only the spec string, axis letters, and chunk multiplicities, without a new positional scheme, mixing primitive or tokenization. The chunk-decomposition view further makes joint vs. sequential axis processing a runtime choice rather than an architectural commitment, switchable by spec edit alone. The construction is dimension-agnostic by design: spec strings impose no upper bound on N short of the 26-letter alphabet that names the axes, which gives a uniform recipe for $N \in \{1, 2, 3, 4\}$ and a clear path toward $N \geq 5$. Plausible $N \geq 5$ targets include parametric spatio-temporal PDE solutions (space-time-parameter sweeps), multi-modal medical volumes (3D space + time + modality), and high-order tensor signals from scientific instruments.

KUN-V2's expressive power within a chunk is bounded by the kernel's expressivity; very long-range dependencies that span many chunks rely on stack depth and the spec choice. Auto-factorization assumes integer-factorizable axis sizes; padding is required when an axis length is prime or when the requested number of chunk layers along an axis does not divide that axis length. Optimal spec selection is currently manual and informed by domain knowledge, and learned spec search remains open. On the empirical side, this paper does not yet evaluate KUN-V2 in strongly nonlinear regimes, on PDE-style benchmarks, or against the most recent state-of-the-art baselines for each dimensionality. We plan to address these directions in subsequent work, including broader nonlinear and PDE evaluations and head-to-head comparisons with up-to-date baselines.

C PROOFS

This appendix gives the full proofs of Theorem 1 (Complexity of a KUN-V2 module) and Theorem 2 (Invertibility of the spec-induced reshape / permutation pipeline), together with the supporting lemmas used implicitly in the body. Throughout, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{D_1 \times \cdots \times D_N \times C}$ has total spatial volume $E = \prod_{i=1}^{N} D_i$ (Section 2), chunk multiplicities $\boldsymbol{m} = (m_1, \ldots, m_N)$, auto-factorized factors $\{c_{i,j}\}$ with $\prod_{j=1}^{m_i} c_{i,j} = D_i$, total chunk count $M = \sum_i m_i$, and a valid spec parse $\boldsymbol{g} = (g_1, \ldots, g_G)$ in

the sense of Algorithm 1. We write $s_t = \prod_{k=1}^{|g_t|} c_{g_t(k)}$ for the chunk size of layer t (with $g_t(k)$ the (axis-index, chunk-position) pair indexed by the k-th slot of g_t, as in Section 3.4), d_t for its channel dimension, and $V_t = E/\prod_{t' \le t} s_{t'}$ for the spatial element count entering layer $t+1$ (with $V_0 = E$). The batch axis B is preserved per sample (Section 2) and is suppressed.

C.1 Proof of Theorem 1

Both ReshapeAndPermute and InversePermuteAndReshape (Algorithm 2) are pure tensor `view`/`permute` operations: they relabel strides without touching the underlying buffer. Hence they contribute no arithmetic operations to the cost.

At encoder layer t, the KernelWrapper consumes the chunk axes in g_t jointly while leaving every other spatial axis untouched. After unfolding the non-chunk axes into the batch dimension, the kernel is invoked once per non-chunked tensor cell. The number of calls is

$$\frac{V_{t-1}}{s_t} \;=\; V_t \;=\; \frac{E}{\prod_{t' \le t} s_{t'}}.$$

Each call processes an input of chunk size s_t and channel dim d_t^{in} and produces an output of chunk size 1 and channel dim d_t^{out}. Let $T_K(s)$ be an upper bound on the per-call cost for both the encoder kernel and its corresponding decoder kernel. Assume the channel widths and number of interior layers are fixed independently of s, with bounds uniform across layers. If T_t denotes the arithmetic cost of encoder layer t, then

$$T_t \;\le\; V_t T_K(s_t). \tag{10}$$

The corresponding decoder layer has the same bound, so summing both paths over all G groups gives

$$T(\boldsymbol{\mathcal{X}}) \;\le\; 2\sum_{t=1}^{G} V_t T_K(s_t).$$

For the Linear kernel with no interior layer, the affine map costs $O(s)$. With a fixed number of interior layers of fixed width, the first and final dense maps cost $O(s)$ and the intermediate maps have constant cost. Thus $T_K(s) = O(s)$ when the channel widths do not depend on s; interpolated interior widths that grow with s are not covered by this bound. Substituting $T_K(s) = O(s)$ into the total-cost bound gives

$$T(\boldsymbol{\mathcal{X}}) \;\le\; 2C\sum_{t=1}^{G} V_t s_t \;=\; 2C\sum_{t=1}^{G} V_{t-1},$$

for a constant C independent of s and uniform across layers. The factor s_t in the per-call cost cancels the $1/s_t$ in the call count. Since $s_t \ge 2$ for every group, $V_{t-1} \le E/2^{t-1}$, and therefore

$$\sum_{t=1}^{G} V_{t-1} \;\le\; E\sum_{t=0}^{\infty} 2^{-t} \;=\; 2E.$$

Consequently, $T(\boldsymbol{\mathcal{X}}) \le 4CE = O(E)$, proving Eq. 7. With uniform $D_i = D$, this is $O(D^N)$. □

For $N = 1$ ($E = D$), this recovers the linear-in-D bound of the original Kernel-U-Net (You et al., 2024).

C.2 Proof of Theorem 2

The argument decomposes into the bijectivity of two maps, their composition, and a round-trip identity.

For each spatial axis $n \in \{1, \dots, N\}$, auto-factorization gives $D_n = \prod_{j=1}^{m_n} c_{n,j}$ *exactly* (Section 3.2; when no exact factorization exists, the fall-back pads to a product $\ge D_n$ and is treated separately in Section F). Define the mixed-radix encoding $\rho_n : \mathbb{Z}_{D_n} \to \prod_{j=1}^{m_n} \mathbb{Z}_{c_{n,j}}$ by

$$\rho_n(i_n) \;=\; (k_{n,1}, \dots, k_{n,m_n}) \quad \text{such that} \quad i_n = \sum_{j=1}^{m_n} k_{n,j} \prod_{j'>j} c_{n,j'}, \qquad k_{n,j} \in \mathbb{Z}_{c_{n,j}}.$$

ρ_n is the standard mixed-radix bijection between $\mathbb{Z}_{D_n}$ and the Cartesian product of digit alphabets; its inverse is the radix recomposition $\rho_n^{-1}(k_{n,1}, \ldots, k_{n,m_n}) = \sum_j k_{n,j} \prod_{j'>j} c_{n,j'}$. The aggregate map $\mathcal{R}$ acts as $\rho_1 \otimes \cdots \otimes \rho_N$ on the spatial multi-index $(i_1, \ldots, i_N)$, with id_B on the batch axis and id_C on the channel axis (both preserved unchanged by the reshape). A finite Cartesian product of bijections is a bijection, so $\mathcal{R}$ is bijective with inverse $\mathcal{R}^{-1} = \rho_1^{-1} \otimes \cdots \otimes \rho_N^{-1}$ on the chunk axes and identities on B, C.

By the validity condition (Algorithm 1), the parse $(g_1, \ldots, g_G)$ satisfies $\sum_{t=1}^{G} |g_t| = M$ and the multiset of emitted factor labels equals $\mathcal{L} \coloneqq \{(i,j) : 1 \le i \le N,\ 1 \le j \le m_i\}$ with no repeats and no omissions. This is exactly the condition for the assignment $\pi_s : [M] \to [M]$, $\ell \mapsto$ (position of the ℓ-th emitted label under the canonical enumeration of $\mathcal{L}$), to be a bijection of $[M]$, i.e. a permutation. Its inverse π_s^{-1} is obtained by reading the parse output in reverse order (Algorithm 3, decoder branch). The induced tensor map Π_s is the standard `permute`: it reorders axes by π_s without touching data, hence is bijective with inverse Π_s^{-1} (the permute by π_s^{-1}). Π_s acts as the identity on the B and C axes by construction (the spec language has no letter for them; Section 2).

The composition of two bijections is a bijection, and a routine calculation gives

$$\Phi_s^{-1} \;=\; \mathcal{R}^{-1} \circ \Pi_s^{-1},$$

because

$$(\mathcal{R}^{-1} \circ \Pi_s^{-1}) \circ (\Pi_s \circ \mathcal{R}) \;=\; \mathcal{R}^{-1} \circ (\Pi_s^{-1} \circ \Pi_s) \circ \mathcal{R} \;=\; \mathcal{R}^{-1} \circ \mathrm{id} \circ \mathcal{R} \;=\; \mathrm{id}.$$

Inspection of Algorithm 3 confirms that the decoder back-end applies exactly Π_s^{-1} (inverse permutation, line(s) in the decoder cleanup block) followed by $\mathcal{R}^{-1}$ (inverse reshape into the original $(D_1, \ldots, D_N)$ shape), so the implemented decoder back-end matches the analytical inverse.

Let F_{id} denote the full encoder/decoder pipeline with every KERNELWRAPPER replaced by the identity, and with the UNet skip-connection scaling chosen so that the skip-add does not double-count (e.g. skips disabled; equivalently, since skip-adds are bilinear and the identity kernel preserves shape, we set the kernel branch to 0 and pass through the skip). Then the encoder collapses to Φ_s followed by G identity passes and the decoder collapses to G identity passes followed by Φ_s^{-1}, i.e. $F_{\mathrm{id}} = \Phi_s^{-1} \circ \Phi_s = \mathrm{id}$ by Step 3. For any input multi-index $(b, i_1, \ldots, i_N, c)$, the cell value at $(b, i_1, \ldots, i_N, c)$ in $F_{\mathrm{id}}(\boldsymbol{\mathcal{X}})$ equals the cell value at the same multi-index in $\boldsymbol{\mathcal{X}}$. □

C.3 PROOF OF COROLLARY 3

Each encoder layer reduces the chunk factors in its parsed group to size 1. Since a valid spec consumes every factor exactly once (Step 2 of the proof of Theorem 2), after G layers every spatial axis has size 1; the channel schedule gives the latent shape $(B, 1, \ldots, 1, C_z)$. The decoder processes the groups in reverse order and restores each factor, so its output has shape $(B, D_1, \ldots, D_N, C)$.

□

When D_i admits no exact m_i-way factorization, the fall-back factorizes into integers whose product $\widetilde{D}_i \ge D_i$; the leading axis is then truncated back to D_i at the decoder output (Section F). The bijection arguments above remain valid on the subspace $\mathbb{Z}_{D_i} \hookrightarrow \mathbb{Z}_{\widetilde{D}_i}$, and the round-trip identity holds on the truncated index range.

D DATASET CONSTRUCTION

This appendix builds each of the 21 rows of Table 2 (Sec. 5.1): two synthetic families, then the real data. The shape and sample count of every row are in Table 3, and the training recipe in Table 4.

D.1 SYNTHETIC: THE SINE FIELD

Two synthetic families carry the synthetic rows of Table 2: the *sine field*, whose pattern stands still, and the *wavelet bank*, whose packet travels. Both are generated from a seeded `numpy` RNG in `experiments_code/data.py`, so a (seed, shape, window count) triple reproduces a pool bit-for-bit; neither family is stored in the repository. The two are run at matched shapes and sample

Table 3: Per-row data configuration of Table 2: tensor shape (leading axis = time), for real data the pool the S samples are drawn from, samples S, and the train/validation/test counts.

N	Row	Shape	Pool	S	Train/Val/Test
1	`1d_sine`	(64)	–	64	42/6/12
2	`2d_sine`	$(64, 8)$	–	128	84/12/24
3	`3d_sine`	$(64, 8, 8)$	–	256	175/25/50
4	`4d_sine`	$(64, 8, 8, 8)$	–	512	357/51/102
5	`5d_sine`	$(64, 8, 8, 8, 8)$	–	1024	704/96/224
1	`1d_wavelet`	(64)	–	60	42/6/12
2	`2d_wavelet`	$(64, 16)$	–	60	42/6/12
3	`3d_wavelet`	$(64, 16, 16)$	–	180	126/18/36
4	`4d_wavelet`	$(64, 16, 16, 16)$	–	420	294/42/84
5	`5d_wavelet`	$(64, 16, 16, 16, 16)$	–	900	630/90/180
1	`1d_burgers`	(64)	2048	100	0.7/0.1/0.2
2	`2d_darcy`	$(64, 8)$	5000	100	0.7/0.1/0.2
3	`3d_ns`	$(20, 8, 8)$	1200	100	0.7/0.1/0.2
1	`1d_ETTh1`, `1d_ETTh2`	(64)	272	100	0.7/0.1/0.2
2	`2d_ETTh1`, `2d_ETTh2`	$(64, 7)$	272	100	0.7/0.1/0.2
3	`3d_geolife`	$(64, 8, 8)$	1024	100	0.7/0.1/0.2
3	`3d_mnist`	$(20, 8, 8)$	10000	100	0.7/0.1/0.2
4	`4d_era5`	$(64, 13, 8, 8)$	700	700	490/70/140
4	`4d_kitti`	$(64, 8, 8, 8)$	407	100	0.7/0.1/0.2

counts wherever possible, so a sine row and the corresponding wavelet row differ in one respect only, namely whether the pattern moves. That pairing is the point of including both: a standing field can be completed by continuing a per-position pattern, while a travelling packet cannot be completed without recovering its velocity from the visible half.

Each sample $\boldsymbol{\mathcal{X}}^{(s)} \in \mathbb{R}^{D_1 \times \cdots \times D_N}$ is a sum of P sinusoids evaluated on the normalised grid $u_n = i_n / D_n$, $i_n \in \{0, \ldots, D_n - 1\}$:

$$\boldsymbol{\mathcal{X}}^{(s)}[i_1, \ldots, i_N] = \sum_{p=1}^{P} a_p^{(s)} \sin\Big(2\pi \sum_{n=1}^{N} \omega_{p,n}^{(s)} \frac{i_n}{D_n} + \phi_p^{(s)}\Big), \tag{11}$$

with, drawn i.i.d. per sample, the component count $P \sim \mathcal{U}\{1, 2\}$ (a fair coin between a single sinusoid and a superposition of two), amplitude $a_p \sim \mathcal{U}(0.8, 1.2)$, integer frequency $\omega_{p,n} \sim \mathcal{U}\{1, 2\}$ per axis, and phase $\phi_p \sim \mathcal{U}(0, 2\pi)$. The frequencies are integers on the normalised grid, so every sample is exactly periodic on that grid and the masked half is a deterministic function of the visible half: the task has an exact solution and a model's error measures how well it recovers that map, not how well it guesses. The generator always draws the full two (a, ω, ϕ) triples and zeroes the amplitude of the unused one when $P = 1$, so neither the draw order nor, therefore, the pool depends on the coin flip. The same parameter distribution is used for every fold.

The leading axis is time, length 64, and the masked half is its last 32 entries in every rank. The per-rank shapes are (64), $(64, 8)$, $(64, 8, 8)$, $(64, 8, 8, 8)$ and $(64, 8, 8, 8, 8)$, i.e. each added dimension is a spatial axis of 8 cells. Two sampling regimes use this generator:

- the cross-model benchmark (Table 2) draws $S = 64$, 128, 256, 512 and 1024 samples at $N = 1, \ldots, 5$ and cuts them into 10 folds read as $0.7/0.1/0.2$ train/validation/test, the same cut every other dataset in that table gets (Table 3);
- the cost ladder (Table 1, Fig. 3) holds $S = 100$ at every rank $N = 1, \ldots, 5$, so that only the input volume changes along it.

In the benchmark the field is z-scored by a single (μ, σ) pair per dataset, fitted *on the training fold only* and applied unchanged to validation and test.

D.2 SYNTHETIC: THE WAVELET BANK

A wavelet-bank sample is one wavelet packet sweeping across the field along a spatial direction. With ψ a mother wavelet, T the window length and $x_1, \ldots, x_n$ the spatial indices,

$$f(t, x_1, \ldots, x_n) = \psi\Big(\frac{\delta(t, \boldsymbol{x})}{\sigma}\Big), \quad \delta = \Big\langle p(\boldsymbol{x}) - b(t) \Big\rangle, \quad p(\boldsymbol{x}) = \sum_{j=1}^{n} w_j \frac{x_j}{D_j}, \quad b(t) = b_0 + u\frac{t}{T}, \tag{12}$$

where $\langle\cdot\rangle$ wraps its argument into $[-\frac{1}{2}, \frac{1}{2})$, $\boldsymbol{w} \in \{0, 1\}^n$ selects the direction of travel, and $u = \pm 1$ is the signed speed, corresponding to one traversal of the period per window. The crest is therefore a straight line in the $(t, \boldsymbol{x})$ plane: over one window it sweeps exactly one period along $\boldsymbol{w}$, in the direction given by the sign of u. Masking the second half of the time axis turns this into "watch the packet for half a window, say where it goes next", which cannot be answered without reading its velocity off the visible half.

A *group* is a triple (wavelet, direction, sign of u), and within a group the only free parameter is the offset b_0, taken on an equally spaced grid of 100 points per period, of which the middle 10 are used. Splitting "spacing" from "count" this way is deliberate: the spacing fixes how close the nearest training window is to a given test window, i.e. the nearest-neighbour coverage that decides whether copying beats extrapolating, while the count fixes the wall-clock. At a spacing of $1/100$ of a period, copying the nearest training window lands within a few per cent for the smoothest members of the bank. This is a real baseline to beat, but not a duplicate of the test window; denser grids would make it one. Directions are the non-empty subsets of the spatial axes: X at $N = 2$; X, Y, XY at $N = 3$; X, Y, Z, XY, XZ, YZ, XYZ at $N = 4$; the 15 non-empty subsets of four axes at $N = 5$. This is the same rule as for the sine family's category masks, so per-category errors can be read side by side. With three mother wavelets and two signs that is 6, 18, 42 and 90 groups, hence $S = 60$, 180, 420 and 900 samples at $N = 2, 3, 4, 5$. Those pools are what the wavelet rows of Table 2 are trained and scored on, under the same $0.7/0.1/0.2$ ten-fold cut (Table 3).

The bank is `ricker`, `gauss1` and `db4`. The first two are analytic; `db4` has no closed form and is computed by the cascade algorithm from the Daubechies scaling filter, which `data.py` obtains by spectral factorisation rather than a hard-coded table (the coefficients agree with the published values to $\sim 10^{-13}$, and the self-test checks $\sum h = \sqrt{2}$, orthogonality and four vanishing moments). `db4` is the only asymmetric member, and it is in the bank specifically to close the "mirror the visible half" shortcut that the symmetric wavelets leave open. Each wavelet has two landmark lobes at $z = \pm z_{\text{land}}$, and the scale is set from them, $\sigma = \text{frac}/(2z_{\text{land}})$, so that the significant support occupies a fixed *fraction of one period* rather than a fixed number of cells, which makes the family self-similar across ranks. On a 64-cell axis this reproduces the reference scales: $z_{\text{land}} = 1.73$, $s = 3.01$ for `ricker`; 1.00, 3.20 for `gauss1`; 1.02, 5.00 for `db4`.

The spatial axes of this family are 16 cells, not the 8 used by the other benchmark rows. With integer direction weights the number of distinct phase samples in one period equals the shortest spatial axis along the direction of travel, so an 8-cell axis offers 8 samples to cover all significant lobes of the packet, which is enough for `gauss1`, marginal for `ricker` and aliased for `db4`. At 16 cells every member of the bank has at least four samples per lobe. All four ranks were moved together, so the "only the rank changes" pairing across $N = 2, \ldots, 5$ is preserved.

Rank 1 has no spatial axis, so Eq. (12) degenerates, since there is no direction to travel along. The `1d_wavelet` row is therefore the *static* member of the family,

$$f(t) = \psi\Big(\frac{t - b}{a}\Big), \tag{13}$$

with the same three mother wavelets and b drawn on the same kind of grid: 100 points spanning that wavelet's own admissible interval $[b_{\text{lo}}, b_{\text{hi}}]$, of which the middle 20 are used, giving $S = 3 \times 20 = 60$ samples. The interval is fixed by the wavelet's own lobes rather than chosen: at b_{lo} the mask boundary falls in the centre of the final lobe, so the visible half shows nearly the whole packet; at b_{hi} it falls in the centre of the first, so nearly all of it is hidden. Sweeping b therefore sweeps the difficulty, and that axis is built into the data instead of being left to a random draw. Because the packet does not move, this row shares the training recipe and the mask rule with the other four but *not* the generator; it reports how the same models behave at rank 1, and it is not part of the "only the rank changes" comparison.

The wavelet rows at $N = 1, \dots, 4$ were run with early stopping off at a constant learning rate of 5×10^{-4} and batch size 16: 500 epochs for KUN-V2 and 200 for the neural-operator baselines.

D.3 STRATIFIED FOLDS ON THE SYNTHETIC ROWS

Both synthetic pools are designed grids, and every sample belongs to a known category. A sine sample's category is its frequency support, the subset of axes along which the field is allowed to vary (2^N subsets at rank N); the generator assigns it round-robin, sample s to category $s \bmod 2^N$. A wavelet-bank sample's category is its sub-family (wavelet, direction), the groups of Appendix D.2 with the two signs of u merged; on the static `1d_wavelet` row it is the mother wavelet. Category membership is therefore a closed-form function of the sample index, so a split can be computed without generating the data, and it is the same for every model on a row.

Within each category the members are shuffled with a fixed seed and dealt into train, validation and test in the proportion $0.7/0.1/0.2$, with at least one sample per fold; on the wavelet rows the categories are dealt in rotation, so every fold receives the floor or the ceiling of its proportional share of every category. Burgers, Darcy, Navier–Stokes, ETT and Moving MNIST have no categories and keep the random split; the trajectory rows use a leak-free split of non-overlapping windows taken at a fixed stride and cut $0.7/0.1/0.2$.

The default split is a single unstratified random permutation of the pool. That is harmless on a large pool of interchangeable samples and fails on a pool spread over many small categories. On `5d_sine` at $S = 1024$ it put between 0 and 6 samples of each of the 32 categories into the validation fold, with one category absent and five represented by a single sample. The validation fold's only job is to pick the reported epoch, so an unstratified split selects the epoch that is best on whichever categories happened to land in it, and a per-category test error resting on one or two samples is noise. Stratifying gives every fold every category in the same proportion, so both the epoch choice and the per-category errors see the whole family.

D.4 PDE AND REAL-WORLD DATASETS

The three PDE rows come from the reference Fourier-Neural-Operator release (Li et al., 2021), converted to the repository's (N, shape) float32 layout and pooled to the benchmark shape of their rank. `1d_burgers` is the viscosity-0.1 Burgers dataset on a grid of 8192 points; `2d_darcy` is Darcy flow through a semiporous medium, taken from the 64×64 release; `3d_ns` is the fixed-forcing Navier–Stokes family. Windows are independent solutions rather than overlapping slices of a single trajectory, so a per-window split leaks nothing and no group handling is needed. Navier–Stokes carries 20 frames rather than 64, which is a property of the source rather than a choice, and $20 = 2 \cdot 2 \cdot 5$ factorises for $\boldsymbol{m} = (3, 3, 3)$, so the spec strings stay the paper's own.

`4d_era5` is ERA5 reanalysis (Hersbach et al., 2020) read from the public WeatherBench 2 release (Rasp et al., 2024), as a (time, pressure level, latitude, longitude) tensor of shape $(64, 13, 8, 8)$. The pool used by the paper is the redesigned one: all seven multi-level variables, $T = 64$, and windows drawn at random over the whole record rather than from a single contiguous block, giving $7 \times 100 = 700$ windows of which 490 train. The thirteen levels are standardised independently. Thirteen is prime, so the chunk factorisation pads to 16 and truncates at the decoder, exactly as the seven ETT variates already do (Appendix F).

ETTh1 and ETTh2 (Zhou et al., 2021) enter at two ranks from the same committed CSV, windowed at a stride equal to the sequence length. The $N = 1$ rows forecast the `OT` column alone at shape (64); the $N = 2$ rows take all seven variables jointly at shape $(64, 7)$ and standardise each variable separately. Rank 2 uses the paper's own 2D spec `vvvttt`: reduce the three variate chunks first, then the three temporal ones.

`3d_mnist` is the original Moving-MNIST benchmark (Srivastava et al., 2015), i.e. 10,000 sequences of 20 frames showing two bouncing digits on a 64×64 canvas, pooled to $(20, 8, 8)$; it is the same file the video-prediction literature evaluates on. As with Navier–Stokes, the 20-frame length belongs to the source and factorises for $\boldsymbol{m} = (3, 3, 3)$.

The paper's N counts tensor rank, so a coordinate list cannot enter a rank-3 or rank-4 cell as it stands. Both trajectory sources are rasterised into fields, and masked-half completion on the field *is*

trajectory forecasting. GeoLife 1.3 (Zheng et al., 2010) ships 182 users and 18,670 raw GPS logs. At each of the T steps the point (longitude, latitude) becomes a Gaussian blob on an $R \times R$ grid and altitude becomes that blob's amplitude, so position survives as *where* the blob sits and altitude as *how bright* it is. Each window is centred on its own mean (longitude, latitude) and scaled isotropically so that the longer axis spans the grid with a margin: isotropic scaling preserves the shape of the path, and per-window scaling means a 60 m walk and a 5 km drive both fill the canvas. Altitude is mapped through a fixed metre window around the window mean, so amplitude stays comparable across samples. The benchmark row reads the $R = 16$ cache at a spatial stride of 2, giving shape $(64, 8, 8)$.

KITTI Tracking (Geiger et al., 2012) ships 21 training sequences of per-frame object boxes. Each frame of 3D boxes is rasterised into an ego-centric voxel grid whose cell value is the fraction of that cell's volume covered by any box, so position enters through *which* voxels are occupied and heading θ through the *orientation* of the occupied slab. Coordinates are KITTI camera coordinates (x right, y down, z forward, with (x, y, z) the bottom centre of a box and θ its rotation about y), and the benchmark row uses $R = 8$, $T = 64$, giving shape $(64, 8, 8, 8)$.

E Models and Experimental Settings

E.1 Implementation and training recipe

KUN-V2 is implemented in PyTorch as a single self-contained module; one codebase serves every $N \in \{1, \ldots, 5\}$ used in the paper. The encoder/decoder, spec parser, auto-factorization, and the kernels all live in the same module. A per-run YAML config supplies the shape, $\boldsymbol{m}$, spec string, kernel, and training hyperparameters; a per-cell training engine consumes it, and a sweep orchestrator deterministically expands sweep specifications into the per-run configs. Every run writes one JSON record carrying its own resolved config (shape, spec, $\boldsymbol{m}$, depth, widths, kernel, learning rate, epoch budget, seed, parameter count, per-epoch history and per-category test error), so the tables of this paper are generated from the records rather than transcribed.

Table 4 lists the settings shared by every cell of Table 2. The epoch budget, early stopping and learning-rate schedule were not shared: the cells come from several training rounds. The scheduler is cosine annealing from the initial rate to a floor; a *constant* rate means the floor equals the initial rate.

Table 4: Training settings of the cells of Table 2.

Parameter	**Value**
Optimizer	AdamW (β_1=0.9, β_2=0.999, ε=10^{-8})
Learning rate	4×10^{-4} or 5×10^{-4}
Weight decay	1×10^{-4}
Gradient clipping	global ℓ_2 norm $\tau = 1.0$
Epochs	200 or 500
Batch size	16
Loss	masked MSE on Ω_m (Eq. (9))
Data	S samples per dataset, per row in Table 3
Folds	10, read as $0.7/0.1/0.2$ train/validation/test
Normalisation	z-scoring fitted on the training fold only
Hidden width	32, 64 or 128, set per run by GPU memory and runtime
Model selection	test masked MSE at the epoch the validation fold selects
Seed	2026, 2016, 2006
Hardware	single NVIDIA A100 (Google Colab)

Row by row, Table 3 gives the tensor shape, the number of samples S and how they split. Each row's S samples are cut into ten folds read as $0.7/0.1/0.2$ train/validation/test; for real data they are drawn from a larger pool of windows, whose size is listed alongside. $S = 100$ holds for ten of the 21 rows, namely every real-data row except ERA5; the synthetic rows use larger or rank-dependent pools. The synthetic rows are split with the stratified folds of Appendix D.3.

Table 5: Seed spread of Table 2: standard deviation (population, ddof $= 0$) of the test masked MSE over seeds 2026, 2016, 2006, same rows and columns. $^{(2)}$: two seeds so far. –: not applicable; ·: fewer than two seeds run yet.

N	task	KUN	FNO	U-NO	DeepONet	Linear	MLP	Transformer	Mamba	PatchTST	ConvLSTM	PredRNN	SimVP	Zero	Last	Copy
1	1d_sine	1.46×10^{-4}	0.2695	0.1196	0.2736	0.1488	0.2694	0.1783	0.4481	0.0301	–	–	–	0.1489	0.3578	0.4086
2	2d_sine	9.79×10^{-6}	0.0603	0.0832	0.0910	0.0070	0.2698	0.1291	0.2348	0.0797	–	–	–	0.0952	0.1859	0.2585
3	3d_sine	1.57×10^{-4}	0.1113	0.1006	0.0808	0.0279	0.1012	0.1273	0.1168	–	0.0696	0.2169	0.3633	0.1535	0.1405	0.0875
4	4d_sine	1.40×10^{-4}	0.0382	0.0240	0.0367	–	–	0.0325	0.0480	–	–	–	–	0.0378	0.1077	0.0202
5	5d_sine	2.37×10^{-5}	0.0701	·	0.0347	–	–	0.0582	0.0336	–	–	–	–	0.0251	0.0113	0.0241
1	1d_wavelet	1.47×10^{-4}	5.00×10^{-5}	1.91×10^{-4}	0.0053	0.0058	0.0017	4.53×10^{-4}	0.0011	8.58×10^{-5}	–	–	–	0.0205	0.6174	0.0042
2	2d_wavelet	0.0014	0.0049	0.0131	0.0428	0.0043	0.0136	0.0215	0.1174	0.0749	–	–	–	0.0022	0.0217	0.0044
3	3d_wavelet	1.59×10^{-4}	0.0020	0.0018	0.0194	0.0085	0.1740	0.0162	0.0147	–	0.0386	0.0215	0.0084	3.60×10^{-4}	0.0113	7.19×10^{-4}
4	4d_wavelet	3.76×10^{-4}	4.50×10^{-4}	9.05×10^{-4}	0.0260	–	–	0.0272	0.0196	–	–	–	–	9.17×10^{-4}	0.0306	0.0018
5	5d_wavelet	0.0220	·	0.0403	·	–	–	0.0143	0.0033	–	–	–	–	3.58×10^{-4}	0.0247	7.17×10^{-4}
1	1d_burgers	0.0443	0.0326	0.0130	0.0638	0.0728	0.0610	0.1624	0.0729	0.0469	–	–	–	0.1234	0.3958	0.3144
2	2d_darcy	0.0284	0.0283	0.0842	0.0134	0.0387	0.0346	0.0355	0.0887	0.0649	–	–	–	0.0699	0.1163	0.0338
3	3d_ns	0.0036	0.0023	0.0066	0.0139	0.0041	0.0071	0.0075	0.0062	–	0.0043	0.0586	0.0043	0.0212	0.0424	0.0310
1	1d_ETTh1	0.0027	0.0228	0.0181	0.0035	0.0237	0.0175	0.0082	0.0102	0.0175	–	–	–	0.6247	0.0045	0.0134
1	1d_ETTh2	0.0189	0.0447	0.0561	0.0407	0.0234	0.0367	0.0235	0.0528	0.0526	–	–	–	0.2167	0.0320	0.0476
2	2d_ETTh1	0.0122	0.0564	0.0404	0.0367	0.0130	0.0177	0.0216	0.0109	0.0114	–	–	–	0.0670	0.0522	0.0374
2	2d_ETTh2	0.0377	0.0383	0.0619	0.1684	0.0522	0.0406	0.0434	0.0245	0.0357	–	–	–	0.0707	0.0357	0.0052
3	3d_geolife	0.0281	0.0033	0.0188	0.0065	8.48×10^{-4}	0.0160	0.0188	0.0336	–	0.0239	0.0149	0.0102	0	0	0
3	3d_mnist	0.0642	0.0442	0.0426	0.0386	0.0671	0.0565	0.0313	0.0251	–	0.0192	0.0275	0.0343	0.0294	0.0860	0.0425
4	4d_era5	0.0016	0.0020	0.0018	0.0015	–	–	0.0019	9.48×10^{-4}	–	–	–	–	0.0075	8.25×10^{-4}	9.02×10^{-4}
4	4d_kitti	0.0625	0.0249	0.0301	0.0246	–	–	0.0140	0.0121	–	–	–	–	0	0	0

Each dataset is z-scored by statistics fitted *on the training fold only* and applied unchanged to validation and test. Most datasets use a single (μ, σ) pair; the exceptions are those whose second axis carries incommensurable physical units, which are standardised per index along that axis: ETT (seven variables) and ERA5 (thirteen pressure levels, where temperature varies by more than a hundred kelvin across the column). Every run record stores which of the two it used and how many statistics were fitted.

Per run we touch `numpy.random.default_rng(seed)` (data generator plus fold assignment) and `torch.manual_seed(seed)` (model init and training stochasticity); the folds are shared by every model compared on a dataset. Each cell of Table 2 is the mean over seeds 2026, 2016 and 2006 of one fixed configuration; Table 5 gives the standard deviation over the same seeds. Cells with fewer than three seeds so far are marked in both tables.

E.2 Model configurations

Table 6 gives each column of Table 2: how it reads a rank-N tensor, its architecture, the widths it ran at, and the ranks it enters. A model enters only at the ranks where it reads a rank-N input without flattening away the adjacency the rank is about (Sec. 5.1).

Every learned baseline wraps a published reference implementation instead of a re-implementation: PatchTST from the Time-Series-Library (Wu et al., 2023); ConvLSTM, PredRNN and SimVP from OpenSTL (Tan et al., 2023), whose Apache-licensed ports are the maintained versions of the original Theano and unlicensed author code; FNO and UNO from `neuraloperator` (Kovachki et al., 2023); DeepONet from DeepXDE (Lu et al., 2021b), the authors' own library; Mamba from the authors' `mamba-ssm`; and the Transformer encoder from PyTorch's reference modules. The only code written for this paper is the shape adapter that maps its $(B, D_1, \ldots, D_N, C)$ contract onto each library's expected layout. The Linear and MLP columns are the exception: they were implemented for this paper as dense channel-mixing controls (a full $E \times E$ matrix, and the same map through a width-128 bottleneck with a GELU) that instantiate the $O(E^2)$ row of Table 1, rather than the channel-*independent* Linear of Zeng et al. (2023), whose parameter count does not grow with the field at all.

The constraint that ties $\boldsymbol{m}$ to the spec is: m_i equals the count of axis-letter i in the spec string. The kernel is either a single linear map or a linear–ReLU–linear map with one hidden layer (`kernel_hidden_layer` $= 0$ or 1); with the single linear map the whole model is affine, and 13 of the 21 KUN-V2 cells of Table 2 are affine. The spec, depth and kernel of the sine and wavelet cells come from the search of Appendix G.

E.3 Cost ladder: Table 1 and Fig. 3

The ladder runs the sine field alone at the five shapes of Table 7, so $E = 64$, 512, 4,096, 32,768 and 262,144 and only the input volume changes along it. Every rank holds $S = 100$ samples on a

Table 6: Model configurations in Table 2. E is the number of cells of one sample; w the hidden width. Widths other than the usual one are named with their rows.

Model	N	Input	Architecture	Width
KUN-V2	1–5	every axis factorised into $m = 3$ chunks; the spec orders and groups the chunk reductions	encoder of kernel reductions, mirrored decoder, additive U-Net skips; 1–3 stacked modules; kernel is one linear map (affine) or linear–ReLU–linear	128 (64 on Burgers)
FNO	1–5	field on its grid, channels first	4 Fourier layers (2 on ERA5 and KITTI, 3 on GeoLife); 16 modes on the time axis, 4 per other axis, at most half the axis length (3 on Navier–Stokes)	128 or 32; 16 on `5d_wavelet`
U-NO	1–5	field on its grid	5 layers, widths $(w, 2w, 2w, 2w, w)$, scalings $(1, \frac{1}{2}, 1, 1, 2)$, modes as FNO	128, 64 (ERA5) or 32; 8 on `5d_wavelet`
DeepONet	1–5	branch net reads the flattened field, trunk net the coordinates	3 hidden layers in each net, latent size w, ReLU	128; 64 (KITTI); 32 at $N = 5$
Linear	1–3	flattened field	one dense $E \to E$ map ($E^2 + E$ parameters)	–
MLP	1–3	flattened field	$E \to E \to E$ with GELU ($2E^2 + 2E$ parameters)	–
Transformer	1–5	one token per time step, carrying the flattened spatial slice	pre-norm encoder, 2 layers (3 on KITTI, 1 on ERA5), 4 heads, feed-forward $4w$, learned positions	128; 64 (KITTI); 32 at $N = 5$ and on `1d_ETTh1`
Mamba	1–5	each time step's spatial slice embedded linearly	2 Mamba blocks (1 on GeoLife), state 16, convolution 4, expansion 2	128; 64 (GeoLife); 32 at $N = 5$
PatchTST	1–2	non-overlapping patches of 16 time steps, all channels of a patch embedded jointly	RevIN-style normalisation, 2 blocks, 8 heads, feed-forward $2w$, dropout 0.1	128
ConvLSTM, PredRNN	3	video (T, H, W)	4 recurrent layers, filter 5; the masked half is rolled out with no teacher forcing	128; 32 on GeoLife (3 layers) and for PredRNN on `3d_wavelet`
SimVP	3	video (T, H, W)	IncepU; spatial hidden w, temporal hidden $8w$, 4 spatial and 8 temporal blocks	128; 32 on `1d_wavelet`
Zero, Last, Copy	1–5	–	predict zero, repeat the last visible slice, paste the visible half	–

random $0.7/0.1/0.2$ split, at batch size 16. The wavelet bank is left out because its pool grows with rank.

Table 7: Shapes and KUN-V2 specs of the cost ladder. The leading axis is time; spatial axes are length 8. Axis letters name the axes of the shape in order, and the spec is written in their lowercase forms.

N	axis letters	shape	spec	$\boldsymbol{m}$
1	`L`	(64)	`lll`	(3)
2	`TV`	$(64, 8)$	`vvvttt`	$(3, 3)$
3	`LHW`	$(64, 8, 8)$	`[hw][hw][hw]lll`	$(3, 3, 3)$
4	`LXYZ`	$(64, 8, 8, 8)$	`[xyz][xyz][xyz]lll`	$(3, 3, 3, 3)$
5	`LWXYZ`	$(64, 8, 8, 8, 8)$	`[wxyz][wxyz][wxyz]lll`	$(3, 3, 3, 3, 3)$

Eight models, at width 32 unless stated: KUN-V2 (one module, kernel with one hidden layer of width 32, specs of Table 7); Mamba (2 blocks); DeepONet (3 hidden layers per net); FNO (4 layers, 16 modes on the time axis and 4 per spatial axis); U-NO at width 16 with the same modes; the Transformer with *one token per grid point* (4 heads, 2 layers), unlike the one-token-per-time-step Transformer of Table 2; and the width-free Linear and MLP.

Each run trains for 6 epochs with early stopping off, AdamW at a constant 4×10^{-4} and weight decay 10^{-4}, seed 2026 only, in fp32 with TF32 matrix multiplies, on one NVIDIA A100-SXM4-40GB. Six epochs are far from convergence, so the ladder reports cost and no accuracy. Parameters are the trainable count. Time per epoch is the mean over epochs 2–6 (the first epoch, dominated by compilation and allocation, is dropped) and includes that epoch's validation and test pass. Peak GPU memory is `torch.cuda.max_memory_reserved`.

Two cells are blocked before training by guards: MLP at $N = 4$ (2.15×10^9 parameters, above the 1.2×10^9 limit) and the Transformer at $N = 4$ (32,768 tokens, above the 8,192-token limit); Linear, MLP and the Transformer are not run at $N = 5$, and Linear at $N = 4$ has no record. Their parameter counts in Table 1 are analytic ($E^2 + E$ for Linear, $2E^2 + 2E$ for MLP) and set in italics.

F SPEC-LANGUAGE REFERENCE

This appendix specifies the spec-string mini-language consumed by the `parse_spec` and `expand_spec` routines of the KUN-V2 module.

Tokens are lowercase ASCII letters (one per spatial axis) and three meta-characters: `[`, `]`, and digits `0--9`. The batch axis B and channel axis C have no letter and never appear in a spec (Section 2). Because each spatial axis is named by a distinct lowercase letter, the 26-letter alphabet bounds the spec language at $N \leq 26$ spatial axes.

A spec is a non-empty sequence of *terms*; each term is a *letter* or a *bracket group*, optionally followed by a repeat count.

```
<spec>        ::= <term>+
<term>        ::= <atom> <repeat>?
<atom>        ::= <letter> | "[" <inside> "]"
<inside>      ::= <term>+
<letter>      ::= "a" | "b" | ... | "z"
<repeat>      ::= <digit>+
<digit>       ::= "0" | "1" | ... | "9"
```

Brackets may nest; `<repeat>` attaches to the immediately preceding `<atom>`, so both `h3` and `[hw]3` are legal.

`expand_spec` rewrites repeats by replicating the preceding atom r times (function `expand_spec` in `kun_lib_v2.py`, lines `232--274`). Examples (verifiable verbatim with `expand_spec`):

```
h2w2hwl3       →  hhwwhwlll
h2w2[hl2]wl    →  hhww[hll]wl
[ab2]3         →  [abb][abb][abb]
[hwl]hwl[hwl]  →  [hwl]hwl[hwl] (no change)
```

After expansion, `parse_spec` (`kun_lib_v2.py`, lines `34--51`) returns an ordered list of groups $\boldsymbol{g} = (g_1, \ldots, g_G)$. A bare letter ℓ emits a singleton group $g = \ell$; a bracket `[abc]` emits a joint group $g =$ `abc`. Each group will be consumed by exactly one encoder layer (Section 3.4).

A spec is *valid* for chunk multiplicities $\boldsymbol{m} = (m_1, \ldots, m_N)$ on a shape with axis letters $\Sigma = \{\ell_1, \ldots, \ell_N\}$ iff after expansion, the count of ℓ_i across all groups equals m_i for every i, and every letter in the expansion belongs to Σ. Equivalently, $\sum_{t=1}^{G} |g_t| = M = \sum_i m_i$ and the multiset of letters across $\boldsymbol{g}$ equals $\{\ell_i^{(m_i)}\}_i$. `AssignChunks` (Algorithm 1) raises if any per-letter chunk-factor queue under- or over-runs.

`create_chunk_size_raw` factorizes D_i into m_i factors $(c_{i,1}, \ldots, c_{i,m_i})$ minimising $\mathrm{std}(c_{i,j})$ (Eq. (2)). When no balanced exact factorization of D_i into m_i positive integers exists, i.e. when

no m_i-tuple of integer factors in the candidate window below satisfies $\prod_j x_j = D_i$, the procedure falls back to a *ceiling-product* factorization: it relaxes the equality constraint to $\prod_j x_j \geq D_i$ and picks the smallest such balanced integer tuple, then truncates the surplus at the decoder output (`brute_min_score_ge_N` in `kun_lib_v2.py`, lines `101--190`):

1. Set $r = D_i^{1/m_i}$ and form the integer candidate window $[\lfloor r \rfloor - d,\ \lceil r \rceil + d]$ (default d=2).
2. Enumerate m_i-tuples $\boldsymbol{x}$ from the candidate window with $\prod_j x_j \geq D_i$.
3. Pick the tuple minimising the combined score below, whose 0.8 weight on percentage overshoot keeps the working volume close to D_i rather than blindly balanced:

$$0.2 \cdot \mathrm{std}(\boldsymbol{x}) + 0.8 \cdot \frac{\prod_j x_j - D_i}{D_i}. \tag{14}$$

The fall-back produces $\widetilde{D}_i = \prod_j c_{i,j} \geq D_i$; the encoder operates on the inflated volume $\widetilde{D}_i$ and the leading-axis output is truncated back to D_i at the decoder boundary. The bijection arguments of Theorem 2 continue to hold on the embedded subspace $\mathbb{Z}_{D_i} \hookrightarrow \mathbb{Z}_{\widetilde{D}_i}$ (Step 1 of the proof).

With $(D_1, D_2, D_3) = (27, 64, 8)$ and $\boldsymbol{m} = (3, 3, 3)$:

$$27 \to (3, 3, 3), \quad 64 \to (4, 4, 4), \quad 8 \to (2, 2, 2),$$

all three factorizations exact. With $(D_1, D_2) = (27, 17)$ and $\boldsymbol{m} = (3, 3)$:

$$27 \to (3, 3, 3), \quad 17 \to (3, 3, 2) \text{ (no exact triple exists; product } 18 \geq 17\text{, fall-back).}$$

Note that a fall-back is triggered by the *absence* of an exact factorization, not merely by an awkward-looking D_i: 16 with $m_i = 3$ factorizes exactly as $(2, 2, 4)$, and the score of Eq. (14) prefers it (0.189) over the over-shooting $(3, 3, 2)$ (0.194). With the spec `[hwl]hwl[hwl]` and $\boldsymbol{m} = (3, 3, 3)$, the expansion is unchanged and parsing yields the five groups (`hwl`, `h`, `w`, `l`, `hwl`), i.e. five encoder layers; the count of each letter is 3, matching $\boldsymbol{m}$.

G Spec, Kernel, Depth and Width Ablations

The spec string is the only argument that changes when KUN-V2 is retargeted to a new rank, so the question here is how much it is worth next to the ordinary capacity knobs: at fixed data and budget, how far apart do two legal specs land, and is the best one a property of the architecture or of the data? We run the search on both synthetic families, the standing sine field and the travelling wavelet bank of Appendix D, because the pair differs in exactly one property, whether the pattern moves, and that property is what a spec schedules against.

With $\boldsymbol{m} = (3, \ldots, 3)$ the encoder applies $M = 3N$ chunk reductions, and a spec partitions and orders them. Enumerating every legal spec is not affordable, since at $N = 3$ alone the 9 reductions admit on the order of 10^4 groupings. We therefore group specs into the families of Table 8, named by the shape of the partition rather than by the axis letters, so that a family means the same thing at every rank. Every family representative is crossed with width, depth and kernel (Table 9) on the class-stratified folds of each synthetic dataset, with one seed, 200 epochs on sine and 500 on the wavelet bank; the KUN-V2 cells of the sine and wavelet rows of Table 2 are each dataset's best cell of this search. On `5d_wavelet` every cell stops at its time budget, 38 cells after ≈25 of 200 epochs and the other 189 after ≈4, so that row of Table 2 takes a separate full-length run; on `5d_sine` 18 of the 257 cells stop at a budget after 23–75 epochs, the rest run the full 200.

Figure 4 pins one knob at a time and reports the mean and spread of the five best cells that remain, so a bar is what KUN-V2 reaches reliably with that knob fixed and the others free, not its single luckiest cell. The bars separate the knobs by kind. Width and depth act as capacity: increasing them increases the model's capacity. The spec does not: no family leads at every rank, and the family that leads changes between the standing sine field and the travelling wavelet bank at the same rank, even with width and depth free. The best spec is therefore a property of the data rather than of the architecture. The `5d_wavelet` bars are early-training numbers, for the budget reason above.

Table 8: The spec families. Examples are at $N = 4$ (axes `l`, `x`, `y`, `z`; `l` is the masked axis). F needs $N \geq 4$.

Family	Structure	Examples
A: single-axis sequential	Each axis is consumed to exhaustion before the next begins. The family member is the axis *order*, which is what the suffix in an arm name records.	`lllxxxyyyzzz`, `xxxyyyzzzlll`, `zzzyyyxxxlll`
B: round-robin	One axis per layer, rotating.	`xlyzxlyzxlyz`
C: subset-joint	One bracket group over a proper subset S of the axes, the remaining axes sequential; the group first (`front`), interleaved (`inter`) or last (`back`).	`[xy][xy][xy]lllzzz`, `[xy]lz[xy]lz[xy]lz`, `lllzzz[xy][xy][xy]`
D: fully joint	Every axis in every layer.	`[lxyz][lxyz][lxyz]`
E: bookend	Joint endpoints with a sequential middle, in three placements (`front`, `mid`, `back`).	`[lxyz][lxyz]lxyz`, `[lxyz]lxyz[lxyz]`, `lxyz[lxyz][lxyz]`
F: two-group joint	The axes split into two disjoint bracket groups of at least two axes each: first-then-second, second-then-first, or alternating.	`[lx][lx][lx][yz][yz][yz]`, `[lx][yz][lx][yz][lx][yz]`

Table 9: The four knobs of the KUN-V2 search of Appendix G. Every combination of spec representative, width, depth and kernel is one cell; the cheapest tier (width 32, depth 1, linear kernel) additionally runs every spec the six family generators produce, with A and B restricted to moving the time axis at $N \geq 4$ (8, 25, 51, 119 specs at $N = 2, 3, 4, 5$).

Knob	What it controls	Values swept	Effect on the model
spec	Order and grouping of the $3N$ chunk reductions of the encoder: which axes are mixed together in one layer, and in which order the axes are consumed (Appendix F).	families A–F (Appendix G); one representative per family, three for F: 4 at $N = 2$, 5 at $N = 3$, 8 at $N = 4$, 8 at $N = 5$	which cross-axis interactions one layer can express
width	Channel width of the field and of every kernel (hidden = latent = kernel width).	32, 64, 128	parameters grow $\approx$ quadratically with width
depth	Number of stacked KUN blocks (encoder–decoder pairs), no residual between blocks.	1, 2, 3	parameters grow linearly; more passes over the axes
kernel	Map applied inside each chunk reduction: a single linear layer (the whole model is then affine) or a two-layer MLP with ReLU.	linear, MLP	linear keeps KUN affine; MLP adds a nonlinearity per reduction

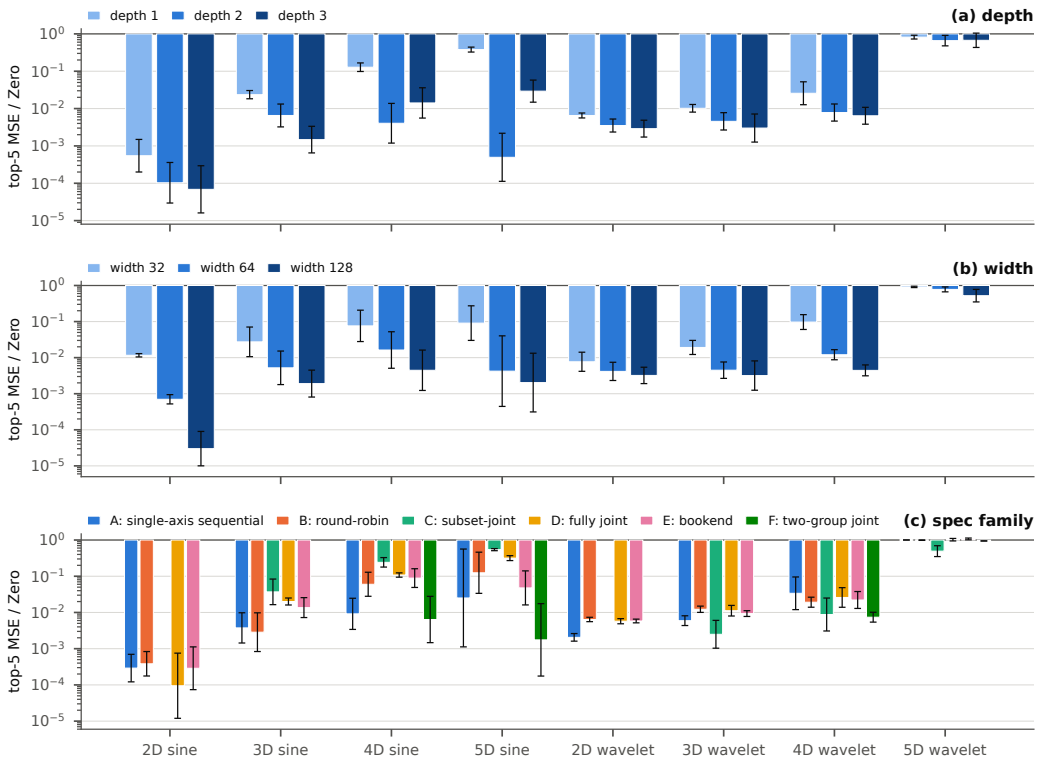


Figure 4: Best test masked MSE of the KUN-V2 search with one knob pinned, relative to the Zero predictor of the same dataset (log scale; lower is better, the top line is Zero). Each bar is the geometric mean over the 5 best cells at that level (fewer where the level has fewer cells), all other knobs and both kernels free, and the whisker spans $\pm$ one standard deviation of $\log_{10}$ MSE, i.e. it is symmetric on the log axis. Missing bars: the family does not exist at that rank (C needs $N \geq 3$, F needs $N \geq 4$). `5d_wavelet` cells are stopped by a time budget, 38 after $\approx$25 and 189 after $\approx$4 of 200 epochs; 18 `5d_sine` cells after 23–75.