# Disentangling Transformer Design Choices for Physical Trajectory Prediction

Brennan Dury

**Abstract**

Transformers are increasingly used as data-driven surrogates for physical trajectory prediction. We show that Attentional Neural Integral Equations (ANIE), block-causal video transformers (VT), and backpropagation-through-time (BPTT) are instances of a unified $2\times2\times3$ design grid with three axes: (i) weight sharing across sequential blocks of layers, (ii) residuals across blocks, and (iii) how temporal dependencies are modeled (broadcasted / teacher forcing / recursive). This yields a single, flag-controlled implementation of all twelve configurations and enables controlled ablations. Across four PDE datasets (2D Navier-Stokes at two viscosities, 1D Burgers, and Shallow-Water), the dominant factor is the distributional shift between training and evaluation time: teacher forcing variants underperform. Within the non-teacher-forcing variants, sharing weights and omitting block-level residuals generally slightly or moderately help. Two combinations,(Broadcasted, Shared Weights, No Block Residuals) and (Recursive, Shared Weights, No Block Residuals), are near-optimal across datasets, with worst-case normalized losses of $1.3071\times$ and $1.2323\times$ the per-dataset minima, and geometric-mean normalized losses of $1.1114\times$ and $1.1458\times$, respectively. Our framing clarifies connections and differences between prior methods, identifies novel optimal configurations, and provides a compact codebase to compose and evaluate trajectory predictors.

## 1 Introduction

Transformers are increasingly used as data-driven surrogates for physical trajectory prediction. The Neural Integral Equations introduced Attentional Neural Integral Equations, (ANIE) [20] an algorithm that parameterizes an integral operator with a transformer, and then searches for a fixed point. Framed this way, ANIE appears to occupy a very different algorithmic space from more conventional transformer-based approaches.

In this work, we unify seemingly disparate approaches in a structured design space. We group transformer layers into sequential blocks. Then, we construct a $2 \times 2 \times 3$ grid of architectural choices along three axes: (1) whether transformer blocks share weights, (2) whether residuals are added to each block, and (3) how temporal dependencies are modeled. Within this grid, ANIE and more conventional architectures emerge as specific points, which can be compared and recombined.

This framing lets us implement a suite of algorithms in a unified codebase, where each axis is exposed as a single flag and architectural differences correspond to small, transparent code changes. Then we can perform controlled experiments across all twelve configurations, allowing us to isolate which design decisions consistently influence performance and allowing us to find novel combinations that outperform the previous methods.

Across datasets, we find that the largest factor distinguishing configurations is whether there is a distributional shift in the inputs between training and evaluation. Within the subset

of configurations without distributional shift, the impact of architectural choices is smaller, but we find that overall weight sharing between blocks and not adding residuals to each block lead to lower validation loss. Further, we find two novel combinations that perform nearly optimally on all datasets. Both combinations feature weight sharing, no residuals around each block, and no distributional shift.

## 2 Trajectory Prediction

Abstractly, the task is to learn a map from initial conditions to full trajectories of the system. To make the abstraction precise, let $I$ and $K$ denote compact topological spaces representing time and space, respectively. A state is a continuous function $s : K \to \mathbb{R}^q$, and a trajectory is a continuous function $f : I \times K \to \mathbb{R}^q$. We write $\mathcal{S} = C(K, \mathbb{R}^q)$ for the space of states and $\mathcal{F} = C(I \times K, \mathbb{R}^q)$ for the space of trajectories. The learning problem is then to approximate the map $\mathcal{S} \to \mathcal{F}$ that maps the initial conditions $s \in \mathcal{S}$ to the trajectory $f \in \mathcal{F}$.

In practice, we let $I$ and $K$ be finite discretizations of time and space in the discrete topology. Then we can represent elements of $\mathcal{S}$ with elements of $\mathbb{R}^{|K| \times q}$, with each row representing one point in space. Similarly, we can represent elements of $\mathcal{F}$ as elements of $\mathbb{R}^{|I| \times |K| \times q}$.

When the spatial dimensionality exceeds one, it is convenient to decompose $K$ into products. For example, in two spatial dimensions, we let $K = H \times W$, where $H$ and $W$ index height and width. States then become elements of $\mathbb{R}^{|H| \times |W| \times q}$, and trajectories become elements of $\mathbb{R}^{|I| \times |H| \times |W| \times q}$. This pattern generalizes naturally to higher dimensions, but in our experiments the data involve either one or two spatial dimensions.

## 3 Transformers for Trajectory Prediction

As a starting point, we define a baseline algorithm that treats trajectory prediction as a special case of video modeling. This approach, which we refer to as the Video Transformer (VT), is the closest to standard transformer architectures used in video tasks [1, 3]. This approach is also very close to the block causal transformer [18], which was proposed for physical trajectory prediction.

We tokenize the spatio-temporal domain into patches. A token corresponds to one spatial patch at one time step. If $K'$ indexes the patches, then a transformer with hidden dimension $d$ acts as $T : \mathbb{R}^{|I| \times |K'| \times d} \to \mathbb{R}^{|I| \times |K'| \times d}$. If there are two spatial dimensions, then we let $H'$ and $W'$ index the patch height and width and regard $T : \mathbb{R}^{|I| \times |H'| \times |W'| \times d} \to \mathbb{R}^{|I| \times |H'| \times |W'| \times d}$. We apply a block causal mask [18], so that each token can attend to all tokens from its own or earlier time steps, but never to the future.

To interface with the transformer, we define an encoder $\psi : \mathbb{R}^{|I| \times |K| \times q} \to \mathbb{R}^{|I| \times |K'| \times d}$ that chooses a representation for each patch. For example, $\psi$ may be a patch-wise MLP applied independently across space and time. On the output side, we define a decoder $\gamma : \mathbb{R}^{|I| \times |K'| \times d} \to \mathbb{R}^{|I| \times |K| \times q}$ to map back into the original trajectory format.

At training time, the model input is $s || f_{<|I|}$, i.e. the concatenation of the initial condition $s$ with the trajectory up to—but excluding—the last time step. The model prediction is compared against the full ground-truth trajectory under mean squared error.

At evaluation time, however, we cannot provide the ground-truth trajectory as input. Instead, we generate one step at a time, feeding the model's own prediction $\hat{f}_i$ back in as input for the next step. This is directly analogous to autoregressive sampling in language and video models. If training loss on a sample reaches zero, then predictions at every step exactly match the ground truth, so evaluation loss is also zero.

# 4 Backpropagation Through Time

A limitation of VT is that the input distribution at training time differs from the one encountered at evaluation time. During training, the input for time step $i$ is the ground truth state $f_i$. However, during evaluation, the model must instead consume its own predicted state $\hat{f}_i$. Unless the model is perfectly accurate, these two distributions differ. This distributional shift can accumulate across steps, often causing the evaluation loss to be much larger than the training loss.

One proposed remedy is to make the training procedure match the evaluation procedure by backpropagating through the trajectory generation algorithm. This idea, called backpropagation through time (BPTT) has been proposed in various settings [12, 17, 10], even beyond transformers.

# 5 ANIE

An alternative perspective begins with the observation that many physical systems can be described by integral equations. By integral equations, we mean equations of the form:

$$y(i,k) = g(i,k) + \int_{\alpha(i)}^{\beta(i)} \int_K G(y(\tau,\xi),\xi,\tau,i,k) d\xi d\tau$$

From the integral equation, we define an associated integral operator as the right-hand side:

$$\kappa(y)(i,k) = g(i,k) + \int_{\alpha(i)}^{\beta(i)} \int_K G(y(\tau,\xi),\xi,\tau,i,k) d\xi d\tau$$

Then a solution to the system must be a fixed point of $\kappa$.

Attentional neural integral equations (ANIE) [20] leverages this structure by parameterizing an integral operator with a transformer and then searching for a trajectory as the fixed point of that operator. To formalize this, we define a sequence of maps: an encoder $\psi : \mathcal{F} \to \mathcal{F}_1$, a transformer $T : \mathcal{F}_1 \to \mathcal{F}_1$, and a decoder $\gamma : \mathcal{F}_1 \to \mathcal{F}$. These components serve analogous roles to those in VT. The transformer in ANIE does not require any mask, but a mask may be used to enforce integration bounds in the operator.

We approximate the residual $\Delta(y) = \kappa(y) - y$ by the transformer $T$, so we approximate $\kappa$ by $\hat{\kappa}(y) = y + T(y)$. We run $m$ refinement steps, with smoothing parameter $b$: $R(y^i) = y^{i+1} = y^i + bT(y^i)$. When $T = \Delta$, this equals the successive iterations $y^{i+1} = (1-b)y^i + b\kappa(y^i)$. [20] found that the choice of $b$ did not significantly change the results, and found only small performance improvements for values larger than $m = 3$, so we set $b = 1$ and $m = 3$ in all experiments. [20] also terminates the iterations early when the change is sufficiently small. We neglect this because we set $m$ small anyway.

For any $s \in \mathcal{S}$, we define $s\vec{1} \in \mathcal{F}$ as the trajectory with constant state $s$: $s\vec{1}(i,k) = s(k)$. ANIE chooses $s\vec{1}$ as the initialization, so the forward pass of ANIE : $\mathcal{S} \to \mathcal{F}$ is ANIE$(s) = [\gamma \circ R^{(m)} \circ \psi](s\vec{1})$. $R^{(m)}$ denotes $m$ iterations of $R$.

If $\hat{\kappa}$ is a contraction, then the successive iterations converge to a fixed point of $\hat{\kappa}$, equivalently a zero of $T$, by the Banach fixed point theorem. In general, the existence of a fixed point is unknown. But in practice, because $m$ is a fixed hyperparameter, ANIE defines a map $\mathcal{S} \to \mathcal{F}$, regardless of whether the iterations converge as $m \to \infty$.

In this way, ANIE reframes trajectory prediction: instead of autoregressively rolling out a sequence, the model iteratively refines a candidate trajectory with a learned operator.

# 6 Design Grid

ANIE may appear unrelated to VT and BPTT at first glance. But we show that they differ by only a small number of architectural choices. The key insight is that the composition of $m$ transformers with $L$ layers each is a transformer with $L' = Lm$ layers. If $T$ is a transformer with $L'$ layers, we group layers into $m$ blocks of $L$ layers each. Then by varying $T$ along three axes, we can recover VT, BPTT, and ANIE as special cases, while also exploring new hybrids.

We define this space as a $2 \times 2 \times 3$ grid:

| Axis 1 (Weight Sharing) | Axis 2 (Residuals) | Axis 3 (Temporal Dependencies) |
|---|---|---|
| **Shared Weights**: shared weights across $m$ blocks ($B_i \equiv B$) | **Block Residuals**: Add residual connections around each block | **Broadcasted**: unmasked; input $s\vec{1}$ |
| **Unshared Weights**: unshared weights across $m$ blocks ($B_i$ all distinct) | **No Block Residuals**: do not add residual connections around each block | **Teacher Forcing**: block-causal mask; input $s\|\|f_{<i}$ at training time and input $s\|\|\hat{f}_{<i}$ at evaluation time |
| | | **Recursive**: block-causal mask; input $s\|\|\hat{f}_{<i}$ |

Table 1: Each config is coded on three axes (Weight Sharing, Residuals, Temporal Dependencies). "input $s||f_{<i}$" denotes that for time-step $i$, the transformer has access to all ground truth tokens for timesteps less than $i$, including the initial conditions. "input $s||\hat{f}_{<i}$" denotes that for time-step $i$ the transformer has access to the initial conditions and its own predictions for time steps less than $i$.

In this space, we recover VT as (*Unshared Weights*, *Block Residuals*, *Teacher Forcing*), BPTT as (*Unshared Weights*, *No Block Residuals*, *Recursive*), and ANIE as (*Shared Weights*, *Block Residuals*, *Broadcasted*). Further, with this framing, by choosing values along each axis, we obtain twelve possible configurations, which we evaluate systematically in our experiments.

# 7 Design Choice Strengths

In the discussion below, we propose plausible mechanisms by which each choice could affect inductive biases, optimization dynamics, or the flow of information in the model and consequently influence validation loss. Then our results can test these hypotheses and quantify their relative importance. Our list is not exhaustive.

## 7.1 *Shared Weights* vs. *Unshared Weights*

### 7.1.1 Potential Advantages of *Shared Weights*

With *Shared Weights*, the model repeatedly applies the same operator, which may provide an inductive bias towards stable iterative refinement of proposal trajectories. An analogy to convolutional neural networks is useful for understanding the inductive biases of weight sharing: weight sharing across strides provides a powerful inductive bias for image processing; weight sharing across blocks may play a similar role for trajectory prediction.

### 7.1.2 Potential Advantages of *Unshared Weights* over *Shared Weights*

*Shared Weights* restricts the parameter space to a low-dimensional subspace of the unshared model. With enough data, optimization over the larger unshared space is more likely to find a

better solution. Thus, while *Shared Weights* may effectively regularize in low-data regimes, it may also limit expressivity in high-data regimes.

## 7.2 *Block Residuals* vs. *No Block Residuals*

### 7.2.1 Potential Advantages of *No Block Residuals* Over *Block Residuals*

For consistency with [20], we use Post-LN transformer layers instead of Pre-LN transformer layers[19]. For Post-LN transformer layers, when the weights equal 0, a transformer layer simply implements layer normalization. Then the composition of multiple layers implements the final layer normalization, recalling that layer normalization has learnable parameters for the scale and bias.

With *No Block Residuals*, when the weights equal 0, the composition of multiple blocks similarly implements the final layer normalization in the final block. But with *Block Residuals*, when the weights equal 0, the outputs tend to grow with the number of blocks, preventing iterative refinement. Since weight decay keeps weights close to 0, *No Block Residuals* may have a stronger inductive bias towards iterative refinement.

### 7.2.2 Potential Advantages of *Block Residuals* Over *No Block Residuals*

For Post_Norm transformer layers, the layer normalization interferes with the stream of residuals inside the transformer layers. *Block Residuals* features a clean residual stream across blocks, which may better propagate signal through the network.

## 7.3 *Recursive* vs. *Teacher Forcing* vs. *Broadcasted*

### 7.3.1 Potential Advantages of *Recursive* Over *Broadcasted* and *Teacher Forcing*

*Recursive* allocates increasingly deep computational graphs to the prediction of later time-steps, which may be beneficial since later predictions are typically more difficult. *Teacher Forcing* shares this property during evaluation, but only *Recursively* has this property during training, so only *Recursively* receives gradient updates to take advantage of this property.

### 7.3.2 Potential Advantages of *Broadcasted* and *Teacher Forcing* Over *Recursive*

*Recursive* requires backpropagation through a deeper computational graph, which may exacerbate vanishing or exploding gradients. Although modern techniques like residual connections [7], normalization [2], adaptive optimizers [9], and careful weight initialization [5] help stabilize deep neural networks, the recursive setting could still be more fragile to architectural or optimization choices than the others.

### 7.3.3 Potential Advantages of *Teacher Forcing* Over *Broadcasted* and *Recursive*

*Teacher Forcing* has access to $s||f_{<|I|}$ during training, which is information computed by the system itself, which may be useful to predict $f$. $s||\hat{f}_{<|I|}$ instead contains information computed by the model itself, which is error-prone and probably less useful than the information from the system. Meanwhile, $s\vec{1}$ contains no information other than $s$. So, the model may require a greater capacity to accurately compute $f$ from $s\vec{1}$ or $s||\hat{f}_{<|I|}$ than from $s||f_{<|I|}$.

For difficult datasets, this may allow *Teacher Forcing* to at least decrease its training loss, with perhaps some generalization, while the others struggle to even decrease their training loss.

#### 7.3.4 Potential Advantages of *Broadcasted* and *Recursive* Over *Teacher Forcing*

*Teacher Forcing* receives input $s||f_{<|I|}$ during training, but receives input $s||\hat{f}_{<|I|}$ during evaluation. This distributional shift may accumulate over time as the model's errors accumulate, leading to much greater validation loss than training loss.

#### 7.3.5 Potential Advantages of *Broadcasted* Over *Teacher Forcing* and *Recursive*

*Broadcasted* is unmasked, which permits early time-steps to use the keys and values from later time-steps. This may allow the model to adjust its predictions for global consistency across time.

#### 7.3.6 Potential Advantages of *Teacher Forcing* and *Recursive* Over *Broadcasted*

*Broadcasted* receives input $s\vec{1}$, while *Teacher Forcing* and *Recursive* receives inputs $s||f_{<|I|}$ and $s||\hat{f}_{<|I|}$, respectively. As explained in section 7.3.3, $s||f_{<|I|}$ likely contains more useful information than $s||\hat{f}_{<|I|}$, which in turn contains more information $s\vec{1}$. So, the model may require greater capacity to accurately compute $f$ from $s\vec{1}$ than from $s||\hat{f}_{<|I|}$ or $s||f_{<|I|}$.

# 8 Methods

## 8.1 Data and tokenization

### 8.1.1 Navier-Stokes

The 2D incompressible Navier–Stokes equations describe the motion of a 2D, viscous, incompressible fluid. In vorticity form, the system evolves according to

$$\partial_t w + u \cdot \nabla w = \nu \Delta w + f$$
$$\nabla \cdot u = 0$$

where $u : I \times H \times W \to \mathbb{R}^2$ is the velocity field, $w : I \times H \times W \to \mathbb{R}$ is the vorticity defined by $\nabla \times u$, $\nu \in \mathbb{R}_{\geq 0}$ is the viscosity, and $f : I \times H \times W \to \mathbb{R}$ is a forcing term. The velocity can be recovered from the vorticity. In 2D, $\nabla \times u = \partial_x u_y - \partial_y u_x$ so the system can be represented by a scalar field for the vorticity.

We use the Navier–Stokes dataset used in [20] and generated by a script from Fourier Neural Operator [11] with viscosity $10^{-3}$. Additionally, we generate a new dataset with viscosity $10^{-3.5}$. We use temporal resolution $|I| = 9$ and the full spatial resolution $|H| \times |W| = 64 \times 64$, with $q = 1$ channel for vorticity. Thus, each sample consists of initial conditions in $\mathbb{R}^{64\times64\times1}$ and a trajectory in $\mathbb{R}^{9\times64\times64\times1}$. We split each dataset into 4000 training and 1000 evaluation samples. To tokenize the input, we partition space into non-overlapping $4 \times 4$ patches, yielding $|H'| = 16$ and $|W'| = 16$.

### 8.1.2 Burgers

The 1D Burgers equation is a common PDE across multiple areas of physics, with the interesting property that for small viscosity values, the system nearly forms discontinuities. The system evolves according to

$$\partial_t u + u\partial_x u = \nu \partial_{xx} u$$

where $u : I \times K \to \mathbb{R}$ is the scalar field and $\nu \in \mathbb{R}_{\geq 0}$ is the viscosity.

We also use the Burgers dataset used in [20] and generated by a script from [11]. We use temporal resolution $|I| = 9$, and the full spatial resolution $|K| = 1024$. The spatial grid is dense and results from [20, 11] suggest that performance is unaffected by downsampling by a factor of 2 or 4. However, downsampling would limit the information in the initial conditions, making the trajectories non-deterministic given the initial conditions, violating our problem setting. We split the dataset into 800 training and 200 evaluation samples. To tokenize the input, we partition space into non-overlapping patches of length 16, yielding 64 patches.

#### 8.1.3 Shallow-Water

The shallow-water equations are derived from the Navier-Stokes equations. They model the motion of incompressible fluids under the assumption that the height of the fluid above a surface is small, and consequently the vertical velocity is negligible compared to the horizontal velocity. The system evolves according to

$$\partial_t h + \partial_x(hu) + \partial_y(hv) = 0 \tag{1}$$

$$\partial_t(hu) + \partial_x\left(u^2 h + \tfrac{1}{2} g_r h^2\right) + \partial_y(uvh) = -g_r h\, \partial_x b \tag{2}$$

$$\partial_t(hv) + \partial_y\left(v^2 h + \tfrac{1}{2} g_r h^2\right) + \partial_x(uvh) = -g_r h\, \partial_y b \tag{3}$$

where $h \in I \times H \times W \to \mathbb{R}$ is the depth, $u \in I \times H \times W \to \mathbb{R}$ is the horizontal velocity, $v \in I \times H \times W \to \mathbb{R}$ is the vertical velocity, and $b \in I \times H \times W \to \mathbb{R}$ is the time-invariant elevation of the bottom surface.

We use the Shallow-Water dataset from [16]. We use temporal resolution $|I| = 9$ and the full spatial resolution $|H| \times |W| = 128 \times 128$, with $q = 1$ channel for the height. The velocity is not specified directly in the state, but can be estimated from the trajectory. Thus, each sample consists of initial conditions in $\mathbb{R}^{128\times128\times1}$ and a trajectory in $\mathbb{R}^{9\times128\times128\times1}$. We split the dataset into 800 training samples and 200 evaluation samples. To tokenize the input, we partition space into non-overlapping $8 \times 8$ patches, yielding $|H'| = 16$ and $|W'| = 16$.

### 8.2 Optimization

We use the Adam optimizer[9] with weight decay 1e-4 and otherwise use Pytorch's default parameters[14]. The learning rate follows a cosine annealing schedule[13] with 101 epochs per half-cycle. We additionally introduce 10 epochs of linear learning rate warmup[6] to account for the possibility that some configurations rely on warmup to reach a well-conditioned region of the loss landscape[8]. We train for 4959 epochs, which terminates at the 25th local minimum of the cosine annealing schedule (excluding the warmup minimum at the first step). We chose the half-cycle length 101 for consistency with [20] and we chose 4959 epochs because it is the closest local minimum of the learning rate to 5000.

### 8.3 Architectures

- **Transformer** $T$: We use a Galerkin transformer [4], which has shown empirical success in operator learning tasks and has linear complexity in the number of tokens. This is a

transformer with multi-linear attention, in addition to layer normalization for the keys and values, scaling attention by 1/(number of tokens), and a special initialization for the attention projection matrices. We set the model dimension $d = 60$, feedforward dimension $4 \cdot d$, and ReLU activations. Layers are grouped into $m = 3$ sequential blocks of $L = 4$ layers each ($L' = 12$ total layers).

- **Positional encoding**: We compare two positional encoding choices: rotary positional embeddings (RoPE) [15] and the coordinate positional encoding used in [20], where the space-time coordinates, normalized to $[0, 1]$ in each dimension, are concatenated as extra channels. These extra channels must be removed later to maintain the number of channels. For consistency with ANIE, for *Block Residuals*, we remove the extra channels after each block, before adding them again. For *No Block Residuals* we do not.
- **Encoder/Decoder** $\psi/\gamma$: The encoder is a patch-wise MLP with two hidden layers of width $4 \cdot d$ and ReLU activations. In the encoder, we experiment with and without layer-normalization. We also vary the decoder in two ways: [20]'s MLP which takes as input the outputs for all tokens in a timestep with two hidden layers of size 64 and 256, and a patch-wise MLP mirroring the encoder but with the input and output dimensions swapped. We make one change in the first decoder design: we switch from ELU to ReLU.

In total, we vary positional encoding, encoder, and decoder choices in two ways each. Preliminary experiments showed that some grid configurations performed significantly better with particular architectural choices. To account for this, we perform a dense search over these choices for each configuration. Using the Navier-Stokes dataset with viscosity $10^{-3}$, we tune for 313 epochs, corresponding to the second learning-rate local minimum (excluding the warmup minimum at the first step). For the full 4959-epoch training runs on all datasets, we use the best-performing architecture identified in this search.

# 9 Results

| Temporal Dependencies | Weight Sharing | Residuals | NS ($10^{-3}$) | NS ($10^{-3.5}$) | Burgers | SWE |
|---|---|---|---|---|---|---|
| Broadcasted | Shared Weights | No Block Residuals | 2.521e-3 | 9.992e-2 | **2.741e-4** | **2.898e-4** |
| Broadcasted | Shared Weights | Block Residuals | 2.632e-3 | 1.017e-1 | 3.270e-4 | 3.420e-4 |
| Broadcasted | Unshared Weights | No Block Residuals | 2.908e-3 | 8.578e-2 | 4.256e-4 | 5.975e-4 |
| Broadcasted | Unshared Weights | Block Residuals | 3.640e-3 | 1.445e-1 | 3.486e-4 | 2.944e-4 |
| Teacher Forcing | Shared Weights | No Block Residuals | 1.659e-2 | 4.744e-1 | 1.202e-3 | 4.017e-4 |
| Teacher Forcing | Shared Weights | Block Residuals | 1.479e-2 | 4.854e-1 | 1.108e-3 | 3.281e-4 |
| Teacher Forcing | Unshared Weights | No Block Residuals | 1.644e-2 | 5.908e-1 | 1.285e-3 | 6.052e-4 |
| Teacher Forcing | Unshared Weights | Block Residuals | 1.683e-2 | 5.195e-1 | 1.077e-3 | 4.119e-4 |
| Recursive | Shared Weights | No Block Residuals | **1.929e-3** | 1.000e-1 | 3.377e-4 | 3.468e-4 |
| Recursive | Shared Weights | Block Residuals | 2.213e-3 | **8.560e-2** | 4.494e-4 | 4.906e-4 |
| Recursive | Unshared Weights | No Block Residuals | 2.686e-3 | 1.153e-1 | 4.043e-4 | 3.754e-4 |
| Recursive | Unshared Weights | Block Residuals | 2.931e-3 | 1.013e-1 | 4.333e-4 | 5.235e-4 |

Table 2: Validation losses across datasets. Best results per dataset are in bold.

| Temporal Dependencies | Weight Sharing | Residuals | NS ($10^{-3}$) | NS ($10^{-3.5}$) | Burgers | SWE |
|---|---|---|---|---|---|---|
| Broadcasted | Shared Weights | No Block Residuals | 1.3071 | 1.1672 | **1.0000** | **1.0000** |
| Broadcasted | Shared Weights | Block Residuals | 1.3645 | 1.1883 | 1.1933 | 1.1803 |
| Broadcasted | Unshared Weights | No Block Residuals | 1.5075 | 1.0021 | 1.5529 | 2.0620 |
| Broadcasted | Unshared Weights | Block Residuals | 1.8869 | 1.6884 | 1.2722 | 1.0160 |
| Teacher Forcing | Shared Weights | No Block Residuals | 8.6010 | 5.5413 | 4.3855 | 1.3864 |
| Teacher Forcing | Shared Weights | Block Residuals | 7.6663 | 5.6702 | 4.0431 | 1.1321 |
| Teacher Forcing | Unshared Weights | No Block Residuals | 8.5233 | 6.9020 | 4.6888 | 2.0886 |
| Teacher Forcing | Unshared Weights | Block Residuals | 8.7233 | 6.0686 | 3.9301 | 1.4214 |
| Recursive | Shared Weights | No Block Residuals | **1.0000** | 1.1686 | 1.2323 | 1.1968 |
| Recursive | Shared Weights | Block Residuals | 1.1474 | **1.0000** | 1.6397 | 1.6930 |
| Recursive | Unshared Weights | No Block Residuals | 1.3926 | 1.3465 | 1.4751 | 1.2953 |
| Recursive | Unshared Weights | Block Residuals | 1.5196 | 1.1838 | 1.5811 | 1.8066 |

Table 3: Normalized validation loss ratios across datasets (validation loss ÷ per-dataset minimum). Best results per dataset are in bold.

| Temporal Dependencies | Weight Sharing | Residuals | Geometric Mean Normalized Validation Loss | Worst-Case Normalized Validation Loss |
|---|---|---|---|---|
| Broadcasted | Shared Weights | No Block Residuals | **1.1114** | 1.3071 |
| Broadcasted | Shared Weights | Block Residuals | 1.2293 | 1.3645 |
| Broadcasted | Unshared Weights | No Block Residuals | 1.4830 | 2.0620 |
| Broadcasted | Unshared Weights | Block Residuals | 1.4245 | 1.8869 |
| Teacher Forcing | Shared Weights | No Block Residuals | 4.1259 | 8.6010 |
| Teacher Forcing | Shared Weights | Block Residuals | 3.7558 | 7.6663 |
| Teacher Forcing | Unshared Weights | No Block Residuals | 4.8992 | 8.5233 |
| Teacher Forcing | Unshared Weights | Block Residuals | 4.1469 | 8.7233 |
| Recursive | Shared Weights | No Block Residuals | 1.1458 | **1.2323** |
| Recursive | Shared Weights | Block Residuals | 1.3359 | 1.6930 |
| Recursive | Unshared Weights | No Block Residuals | 1.3758 | 1.4751 |
| Recursive | Unshared Weights | Block Residuals | 1.5056 | 1.8066 |

Table 4: Summary across datasets: geometric mean and maximum normalized validation loss per configuration. Best results per summary statistic in bold.

**Two configurations are nearly optimal across datasets.** In table 3, we show summary statistics for each configuration, after dividing the validation losses by the minimum loss on each dataset. (*Broadcasted*, *Shared Weights*, *No Block Residuals*) minimizes the geometric mean normalized validation loss across datasets at 1.1114 followed closely by (*Recursive*, *Shared Weights*,

*No Block Residuals*) at 1.1458. We also consider the worst-case normalized validation loss by taking a maximum over datasets. (*Recursive*, *Shared Weights*, *No Block Residuals*) minimizes the worst-case normalized validation loss at 1.2323, followed by (*Broadcasted*, *Shared Weights*, *No Block Residuals*) at 1.3071. With the same two configurations leading on both summary statistics, we suggest these two configurations as the default best choices.

***Teacher Forcing* performs poorly.** All *Broadcasted* and *Recursive* configurations have worst-case normalized validation loss less than or equal to 2.0620. Meanwhile, on datasets except SWE, all *Teacher Forcing* configurations have normalized validation loss ranging from 3.9301 to 8.7233. On SWE, *Teacher Forcing* configurations are only moderately worse, with normalized validation loss between 1.1321 and 2.0886. So we find that while *Teacher Forcing* configurations are somewhat competitive on some datasets, overall, *Teacher Forcing* configurations perform significantly worse than other configurations. Because only *Teacher Forcing* configurations have a distributional shift between training and evaluation time, the inferior performance of *Teacher Forcing* configurations is well explained by this distributional shift.

***Broadcasted* and *Recursive* are competitive.** Two optimal configurations are *Broadcasted* and two optimal configurations are *Recursive*. *Broadcasted* beats *Recursive* in 8/16 matched comparisons, with a geometric mean improvement of $1.3344/1.3034 = 1.0238$. With mixed and close results, we do not claim either *Broadcasted* or *Recursive* is generally better than the other.

***Shared Weights* moderately outperforms *Unshared Weights*.** All optimal configurations have *Shared Weights*. Excluding *Teacher Forcing* configurations, *Shared Weights* beats *Unshared Weights* in 13/16 matched comparisons, with a geometric mean improvment of $1.4463/1.2026 = 1.2026$. This can be explained by the hypothesis that weight sharing provides an inductive bias towards iterative refinement of proposal trajectories.

***No Block Residuals* slightly outperforms *Block Residuals*.** Three out of four optimal configurations have *No Block Residuals*. *No Block Residuals* beats *Block Residuals* in 12/16 matched comparisons, with a geometric mean improvement of $1.3610/1.2696 = 1.0720$. This can also be explained by the hypothesis that *No Block Residuals* provides an inductive bias towards iterative refinement.

# 10 Conclusion

We introduced a unifying $2\times2\times3$ design grid for transformer-based trajectory predictors and showed that VT, BPTT, and ANIE are simply different coordinates within this unified space. This perspective makes implementation differences explicit, exposes a family of hybrid methods, and allows us to perform an ablation study.

We find that no single configuration is optimal across datasets, but (*Broadcasted*, *Shared Weights*, *No Block Residuals*) and (*Recursive*, *Shared Weights*, *No Block Residuals*) are close to optimal on all datasets. So we recommend these two choices as the best default options.

We find that while *Teacher Forcing* configurations are sometimes competitive, overall, *Teacher Forcing* configurations perform significantly worse than other configurations. Because only *Teacher Forcing* configurations have a distributional shift from training to evaluation time, our results are well explained by this distribution shift.

Within the set of *Broadcasted* and *Recursive* and configurations, we find all configurations have validation loss ratios less than or equal to 2.0620 on all datasets. Within that variation, we find that *Shared Weights* configurations overall outperform *Unshared Weights* configurations, *No Block Residuals* configurations overall outperform *Block Residuals* configurations, and *Broadcasted* and *Recursive* configurations are competitive with each other. These results are consistent with our finding that (*Broadcasted*, *Shared Weights*, *No Block Residuals*) and (*Recur-*

*sive*, *Shared Weights*, *No Block Residuals*) are near optimal on all datasets, further supporting these two configurations as the best default options. Additionally, we hypothesize both *Shared Weights* and *No Block Residuals* provide an inductive bias towards stable iterative refinement, which may underlie the performance gains along both dimensions.

We hope our conceptual framework, results, and code provide a clear substrate to build stronger trajectory predictors and to disentangle the source of performance gains.

# References


[1] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer, 2021.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding?, 2021.

[4] Shuhao Cao. Choose a transformer: Fourier or galerkin, 2021.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Dayal Singh Kalra and Maissam Barkeshli. Why warmup the learning rate? underlying mechanisms and improvements, 2024.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[10] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), May 2021.

[11] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[12] Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. Differentiability in unrolled training of neural physics simulators on transient dynamics, 2024.

[13] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[15] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[16] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning, 2024.

[17] Kiwon Um, Robert Brand, Yun, Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers, 2021.

[18] Thomas Winterbottom, G. Thomas Hudson, Daniel Kluvanec, Dean Slack, Jamie Sterling, Junjie Shentu, Chenghao Xiao, Zheming Zhou, and Noura Al Moubayed. The power of next-frame prediction for learning physical laws, 2024.

[19] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020.

[20] Emanuele Zappala, Antonio Henrique de Oliveira Fonseca, Josue Ortega Caro, Andrew Henry Moberly, Michael James Higley, Jessica Cardin, and David van Dijk. Learning integral operators via neural integral equations. *Nature Machine Intelligence*, 6(9):1046–1062, August 2024.