Appendix

A. Gated Identity Initialization Ablation

All applicable gating variants in the main text were trained with the gated identity initialization. We observed in initial Memory Maze results that the gated identity initialization significantly improved optimization stability and learning speed. Figure 7 compares an otherwise identical 4-layer GTrXL (GRU) trained with and without the gated identity initialization. Similarly to the previous sensitivity plots, we plot the ranked mean return of 10 runs at various times during training. As can be seen from Fig. 7, there is a significant gap caused by the bias initialization, suggesting that preconditioning the transformer to be close to Markovian results in large learning speed gains.

B. Environment Details

Numpad: Numpad (Fig. 8, left) has three actions, two of which move the sphere towards some direction in the x,y plane and the third allows the agent to jump in order to get over a pad faster. The observation consists of a variety of proprioceptive information (e.g. position, velocity, acceleration) as well as which pads in the sequence have been correctly activated (these will shut off if an incorrect pad is later hit), and the previous action and reward. Episodes last a fixed 500 steps and the agent can repeat the correct sequence any number of times to receive reward. Observations were processed using a simple 2-layer MLP with tanh activations to produce the transformer's input embedding.

DMLab-30: Example image observations are shown in Fig. 9. Ignoring the "jump" and "crouch" actions which we do not use, an action in the native DMLab action space consists of 5 integers whose meaning and allowed values are given in Table 4. Following previous work on DMLab (Hessel et al., 2018), we used the reduced action set given in Table 5 with an action repeat of 4. Observations are 72×96 RGB images. Some levels require a language input, and for that all models use an additional 64-dimension LSTM to process the sentence.

In (Wayne et al., 2018), the DMLab Arbitrary Visuomotor Mapping task was specifically used to highlight the MERLIN architecture's ability to utilize memory. In Figure 10 we show that, given a similarly reduced action set as used in (Wayne et al., 2018), see Table 6, the GTrXL architecture can also reliably attain human-level performance on this task.

Memory Maze: An action in the native Memory Maze action space consists of 8 continuous actions and a single discrete action whose meaning and allowed values are given in Table 7. Unlike for DMLab, we used a hybrid continuous-discrete distribution (Neunert et al., 2019) to directly output policies in the game's native action space. Observations are 72×96 RGB images. A visual description of the environment is shown in Fig. 8.

Image Encoder: For DMLab-30 and Memory Maze, we used the same image encoder as in (Song et al., 2020) for multitask DMLab-30. The ResNet was adapted from Hessel et al. (2018) and each of its layer blocks consists of a (3×3, stride 1) convolution, followed by (3×3, stride 2) max-pooling, followed by 2 3×3 residual blocks with ReLU non-linearities.

ACTION NAME	RANGE
LOOK_LEFT_RIGHT_PIXELS_PER_FRAME	[-512, 512]
LOOK_DOWN_UP_PIXELS_PER_FRAME	[-512, 512]
STRAFE_LEFT_RIGHT	[-1, 1]
MOVE_BACK_FORWARD	[-1, 1]
FIRE	[0, 1]

Table 4. Native action space for DMLab. See https://github.com/deepmind/lab/blob/master/docs/users/actions.md for more details.

ACTION	NATIVE DMLAB ACTION
Forward (FW)	`[ 0, 0, 0, 1, 0]`
Backward (BW)	`[ 0, 0, 0, -1, 0]`
Strafe left	`[ 0, 0, -1, 0, 0]`
Strafe right	`[ 0, 0, 1, 0, 0]`
Small look left (LL)	`[-10, 0, 0, 0, 0]`
Small look right (LR)	`[ 10, 0, 0, 0, 0]`
Large look left (LL)	`[-60, 0, 0, 0, 0]`
Large look right (LR)	`[ 60, 0, 0, 0, 0]`
Look down	`[ 0, 10, 0, 0, 0]`
Look up	`[ 0, -10, 0, 0, 0]`
FW + small LL	`[-10, 0, 0, 1, 0]`
FW + small LR	`[ 10, 0, 0, 1, 0]`
FW + large LL	`[-60, 0, 0, 1, 0]`
FW + large LR	`[ 60, 0, 0, 1, 0]`
Fire	`[ 0, 0, 0, 0, 1]`

Table 5. Simplified action set for DMLab from Hessel et al. (2018).

ACTION	NATIVE DMLAB ACTION
Small look left (LL)	`[-10, 0, 0, 0, 0]`
Small look right (LR)	`[ 10, 0, 0, 0, 0]`
Look down	`[ 0, 10, 0, 0, 0]`
Look up	`[ 0, -10, 0, 0, 0]`
No-op	`[ 0, 0, 0, 0, 0]`

Table 6. Simplified action set for DMLab Arbitrary Visuomotor Mapping (AVM). This action set is the same as the one used for AVM in (Wayne et al., 2018) but with an additional no-op, which may also be replaced with the *Fire* action.

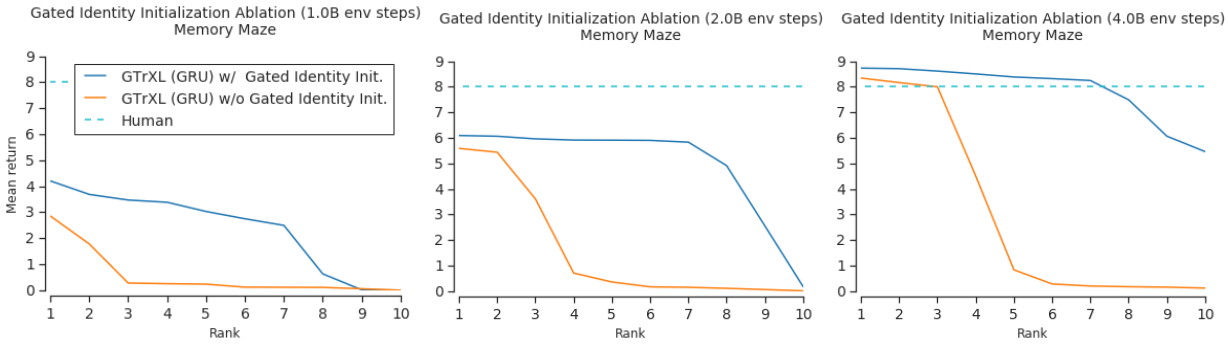

Figure 7. Ablation of the gated identity initialization on Memory Maze by comparing 10 runs of a model run with the bias initialization and 10 runs of a model without. Every run has independently sampled hyperparameters from a distribution. We plot the ranked mean return of the 10 runs of each model at 1, 2, and 4 billion environment steps. Each mean return is the average of the past 200 episodes at the point of the model snapshot. We plot human performance as a dotted line.

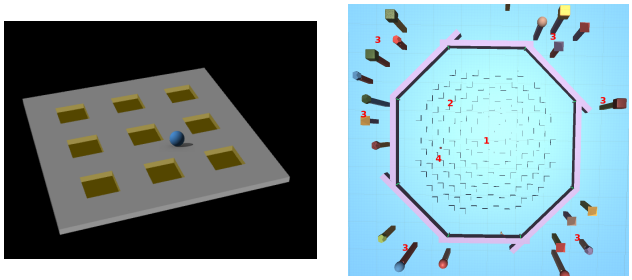

Figure 8. **Left:** The Numpad environment, showing the controllable "sphere" robot and a full 3x3 pad. Pads are activated when the robot collides with their center. The robot can move on the plane as well as jump to avoid pressing numbers. **Right:** Top down view of "Memory Maze": (1) Central chamber, (2) blocks among which the apple is placed, (3) landmarks the agent can use to locate the apple, (4) one of the possible location of the apple.

ACTION NAME	RANGE
LOOK_LEFT_RIGHT	[-1.0, 1.0]
LOOK_DOWN_UP	[-1.0, 1.0]
STRAFE_LEFT_RIGHT	[-1.0, 1.0]
MOVE_BACK_FORWARD	[-1.0, 1.0]
HAND_ROTATE_AROUND_RIGHT	[-1.0, 1.0]
HAND_ROTATE_AROUND_UP	[-1.0, 1.0]
HAND_ROTATE_AROUND_FORWARD	[-1.0, 1.0]
HAND_PUSH_PULL	[-10.0, 10.0]
HAND_GRIP	{0, 1}

Table 7. Hybrid action set for Memory Maze, consisting of 8 continuous actions and a single discrete action.

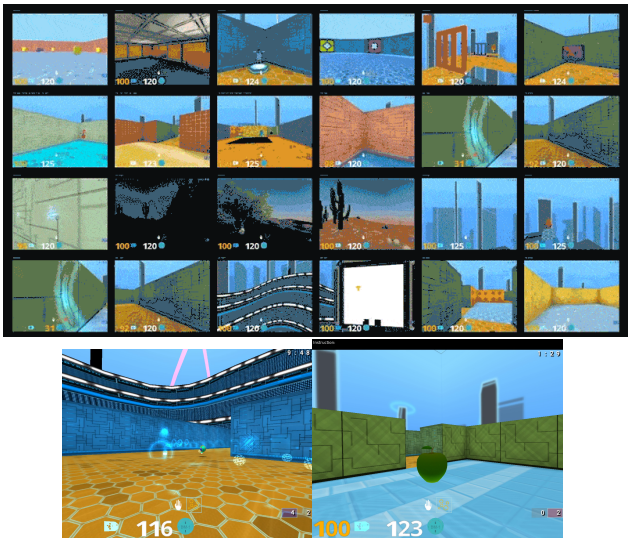

Figure 9. A set of example observations taken from DMLab-30 levels.

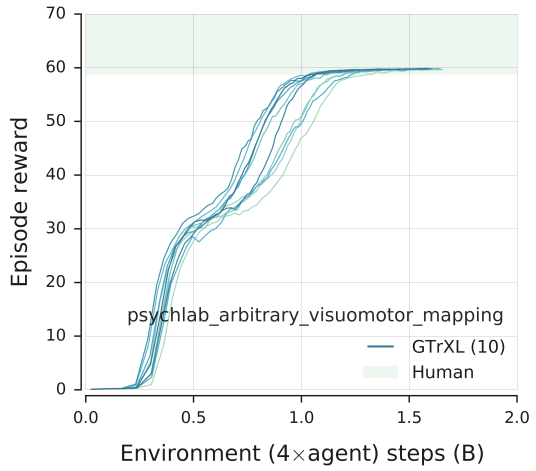

Figure 10. Learning curves for the DMLab Arbitrary Visuomotor Mapping task using a reduced action set.

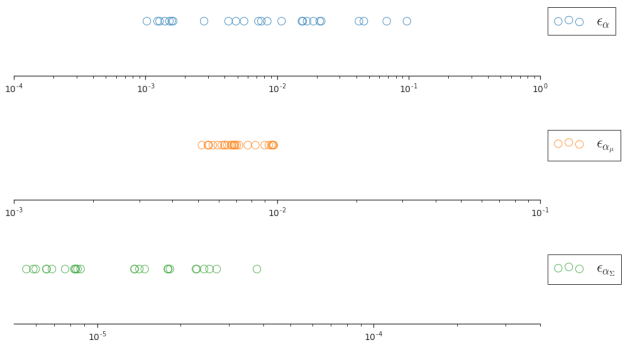

Figure 11. The 25 hyperparameter settings sampled for the sensitivity ablation (Sec. 4.3.2). X-axis is in log scale and values are sampled from the corresponding ranges given in Table 8.

Agent Output: As in (Song et al., 2020), in all cases we use a 256-unit MLP with a linear output to get the policy logits (for discrete actions), Gaussian distribution parameters (for continuous actions) or value function estimates.

C. Experimental details

For all experiments, beyond sampling independent random seeds, each run also has V-MPO hyperparameters sampled from a distribution (see Table 8). The sampled hyperparameters are kept fixed across all models for a specific experiment, meaning that if one of the ϵ_α sampled is 0.002, then all models will have 1 run with $\epsilon_\alpha = 0.002$ and so on for the rest of the samples. The exception is for the DMLab-30 LSTM, where a more constrained range was found to perform better in preliminary experiments. Each model had 8 seeds started, but not all runs ran to completion due to compute issues. These hyperparameter settings were dropped randomly and not due to poor environment performance. We report how many seeds ran to completion for all models. At least 6 seeds finished for every model tested. We list architecture details by section below. All LSTM models have residual skip connections in depth. We used normal xavier initialization (Glorot & Bengio, 2010) for all submodules except the gating layers, which used truncated normal initialization with standard error $1/\sqrt{D}$ where D is the input dimension of the weight matrix.

C.1. Training setup

All experiments in this work were carried out in an actor-learner framework (Espeholt et al., 2018) that utilizes TF-Replicator (Buchlovsky et al., 2019) for distributed training on TPUs in the 16-core configuration (Google, 2018). "Actors" running on CPUs performed network inference and interactions with the environment, and transmitted the resulting trajectories to the centralised "learner".

D. Multi-Head Attention Details

D.1. Multi-Head Attention

The Multi-Head Attention (MHA) submodule computes in parallel H soft-attention operations for every time step, producing an output tensor $Y^{(l)} \in \mathbb{R}^{T\times D}$. MHA operates by first calculating the query $Q^{(l)} \in \mathbb{R}^{H\times T\times d}$, keys $K^{(l)} \in \mathbb{R}^{H\times T\times d}$, and values $V^{(l)} \in \mathbb{R}^{H\times T\times d}$ (where $d = D/H$) through trainable linear projections $W_Q^{(l)}$, $W_K^{(l)}$, and $W_V^{(l)}$, respectively, and then using the combined Q, K, V, tensors to compute the soft attention. A residual connection (He et al., 2016a) to the resulting embedding $E^{(l)}$ is then applied and finally layer normalization (Ba et al., 2016).

MHA$(E^{(l-1)})$:

$$
\begin{aligned}
Q^{(l)}, K^{(l)}, V^{(l)} &= W_Q^{(l)} E^{(l-1)}, W_K^{(l)} E^{(l-1)}, W_V^{(l)} E^{(l-1)} \\
\alpha_{htm}^{(l)} &= Q_{htd} K_{hmd} \\
W_{htm}^{(l)} &= \text{MaskedSoftmax}(\alpha^{(l)}, \text{axis}{=}m) \\
\overline{Y}_{htd}^{(l)} &= W_{htm}^{(l)} V_{hmd}^{(l)} \\
\hat{Y}^{(l)} &= E^{(l-1)} + \text{Linear}(\overline{Y}^{(l)}) \\
Y^{(l)} &= \text{LN}(\hat{Y}^{(l)})
\end{aligned}
$$

where we used Einstein summation notation to denote the tensor multiplications, MaskedSoftmax is a causally-masked softmax to prevent addressing future information, Linear is a linear layer applied per time-step and we omit reshaping operations for simplicity.

Hyperparameter	Environment		
	DMLab-30	Numpad	Memory Maze
Batch Size	128	128	128
Unroll Length	95	95	95
Discount	0.99	0.99	0.99
Action Repeat	4	1	4
Pixel Control Cost	2×10^{-3}	-	-
Target Update Period	10	10	10
Initial η	1.0	10.0	1.0
Initial α	5.0	-	5.0
Initial α_μ	-	1.0	1.0
Initial α_Σ	-	1.0	1.0
ϵ_η	0.1	0.1	0.1
ϵ_α (log-uniform)	LSTM [0.001, 0.025) TrXL Variants [0.001, 0.1)	-	[0.001, 0.1)
ϵ_{α_μ} (log-uniform)	-	[0.005, 0.01)	[0.005,0.01)
ϵ_{α_Σ} (log-uniform)	-	$[5 \times 10^{-6}, 4 \times 10^{-4})$	$[5 \times 10^{-6}, 4 \times 10^{-5})$

Table 8. V-MPO hyperparameters per environment.

Model	# Layers	Head Dim.	# Heads	Hidden Dim.	Memory Size	Runs Completed
LSTM	3	-	-	256	-	8
Large LSTM	12	-	-	512	-	6
TrXL	12	64	8	512	512	6
TrXL-I	12	64	8	512	512	6
GTrXL (GRU)	12	64	8	512	512	8
GTrXL (Input)	12	64	8	512	512	6
GTrXL (Output)	12	64	8	512	512	7
GTrXL (Highway)	12	64	8	512	512	7
GTrXL (SigTanh)	12	64	8	512	512	6
Thin GTrXL (GRU)	12	64	4	256	512	8

Table 9. DMLab-30 Ablation Architecture Details. We report the number of runs per model that ran to completion (i.e. 10 billion environment steps). We follow the standard convention that the hidden/embedding dimension of transformers is equal to the head dimension multiplied by the number of heads. (Sec. 4.1 & Sec. 4.3).

Model	# Layers	Head Dim.	# Heads	Hidden Dim.	Memory Size	Runs Completed
LSTM	3	-	-	256	-	5
GTrXL (GRU)	12	64	8	256	512	5

Table 10. Numpad Architecture Details. (Sec. 4.2).

Model	# Layers	Head Dim.	# Heads	Hidden Dim.	Memory Size
LSTM	3	-	-	256	-
TrXL	12	64	8	256	512
TrXL-I	12	64	8	256	512
GTrXL (GRU)	12	64	8	256	512
GTrXL (Output)	12	64	8	256	512

Table 11. Sensitivity ablation architecture details (Sec. 4.3.2).

Model	# Layers	Head Dim.	# Heads	Hidden Dim.	Memory Size	Runs Completed
GTrXL (GRU)	4	64	4	256	512	8

Table 12. Gated identity initialization ablation architecture details (Sec. A).

D.2. Relative Multi-Head Attention

The basic MHA operation does not take sequence order into account explicitly because it is permutation invariant, so positional encodings are a widely used solution in domains like language where order is an important semantic cue, appearing in the original transformer architecture (Vaswani et al., 2017). To enable a much larger contextual horizon than would otherwise be possible, we use the relative position encodings and memory scheme described in (Dai et al., 2019). In this setting, there is an additional $\mathcal{T}$-step memory tensor $M^{(l)} \in \mathbb{R}^{\mathcal{T} \times D}$, which is treated as constant during weight updates.

RMHA$(M^{(l-1)}, E^{(l-1)})$:

$$\begin{aligned}
\widetilde{E}^{(l-1)} &= [M^{(l-1)}, E^{(l-1)}] \\
Q^{(l)}, K^{(l)}, V^{(l)} &= W_Q^{(l)} E^{(l-1)}, W_K^{(l)} \widetilde{E}^{(l-1)}, W_V^{(l)} \widetilde{E}^{(l-1)} \\
R &= W_R^{(l)} \Phi \\
\alpha_{htm}^{(l)} &= Q_{htd} K_{hmd} + Q_{htd} R_{hmd} \\
&\quad + u_{h*d} K_{htm} + v_{h*d} R_{hmd} \\
W_{htm}^{(l)} &= \text{MaskedSoftmax}(\alpha^{(l)}, \text{axis}{=}m) \\
\overline{Y}_{htd}^{(l)} &= W_{htm}^{(l)} V_{hmd}^{(l)} \\
\hat{Y}^{(l)} &= E^{(l-1)} + \text{Linear}(\overline{Y}^{(l)}) \\
Y^{(l)} &= \text{LN}(\hat{Y}^{(l)})
\end{aligned}$$

where Φ is the standard sinusoid encoding matrix, $u^{(l)}, v^{(l)} \in \mathbb{R}^{H \times d}$ are trainable parameters, the $*$ represents the broadcast operation, and W_R is a linear projection used to produce the relative location-based keys (see (Dai et al., 2019) for a detailed derivation).

D.3. Identity Map Reordering

The Identity Map Reordering modifies the standard transformer formulation as follows: the layer norm operations are applied only to the input of the sub-module and a non-linear ReLU activation is applied to the output stream.

$$\overline{Y}^{(l)} = \text{RMHA}(\text{LN}([\text{SG}(M^{(l-1)}), E^{(l-1)}])) \quad (4)$$

$$Y^{(l)} = E^{(l-1)} + \text{ReLU}(\overline{Y}^{(l)}) \quad (5)$$

$$\overline{E}^{(l)} = f^{(l)}(\text{LN}(Y^{(l)})) \quad (6)$$

$$E^{(l)} = Y^{(l)} + \text{ReLU}(\overline{E}^{(l)}) \quad (7)$$

See Figure 1 (Center) for a visual depiction of the TrXL-I.

Model	Median HNR
LSTM	136.6 ± 3.4
GTrXL	137.1 ± 5.0

Table 13. Final human-normalized median return across all 57 Atari levels for LSTM and GTrXL at 11.4 billion environment steps (equivalent to 200 million per individual game). Both models are 256 dimensions in width. We include standard error over runs.

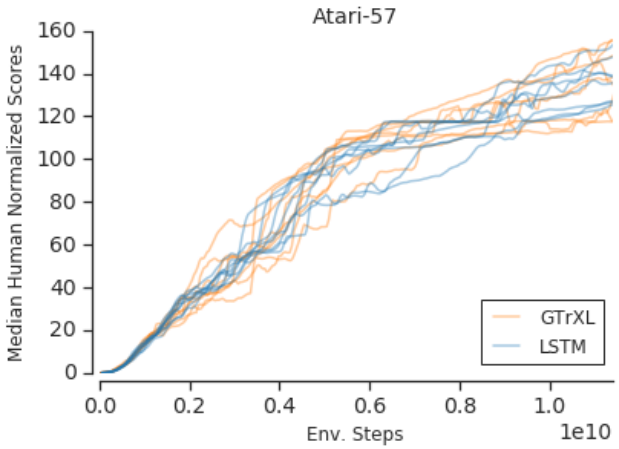

Figure 12. Median human-normalized returns as training progresses for both GTrXL and LSTM models. We run 8 hyperparameter settings per model.

E. Atari-57 Results

In this section, we run the GTrXL on the multitask Atari-57 benchmark (see Fig. 12 and Tab. 13). Although Atari-57 was not designed specifically to test an agent's memory capabilities, we include these results here to demonstrate that we suffer no performance regression on a popular environment suite, providing further evidence that GTrXL can be used as an architectural replacement to the LSTM.

The LSTM and GTrXL are matched in width at 256 dimensions. The GTrXL is 12 layers deep to show our model's learning stability even at large capacity. The LSTM architecture matches the one reported in (Song et al., 2020). We train for 11.4 billion environment steps, equivalent to 200 million per environment. We run 8 hyperparameter settings per model.

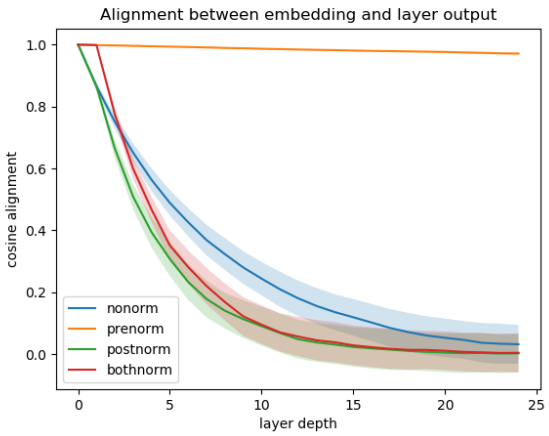

Figure 13. Alignment between embedding (network input) and layer features for a randomly-initialized deep linear residual network. We can see that the only layer norm placement which maintains high alignment is the identity map re-ordering (prenorm for short in the legend). Averaged over 1000 random 256-dimensional embeddings sampled from a standard Gaussian. Shaded regions represents standard error.

F. Motivating the Identity Map Re-ordering

In this section, we present further evidence corroborating the hypothesis that the identity map re-odering of the layer normalization operation can enable an initial reactive policy compared to other placements of the layer norm.

To show this, we sampled a randomly-initialized deep residual network. For simplicity, we used a single fully-connected layer in each residual stream in place of the MLP or MHA sub-module in the transformer blocks. Each layer is randomly initialized using the PyTorch (Paszke et al., 2019) "torch.nn.Linear" default initializer and does not use non-linear activations. Let x be the input to a residual block, $f(x)$ the residual module, and y the output of the residual block. We consider 4 different placements of layer normalization within these randomly-initialized networks.

1. **nonorm** ($y = x + f(x)$): represents a standard residual network without any normalization operations.
2. **prenorm** ($y = x + f(LN(x))$): is shorthand for the identity map re-ordering, where layer normalization is applied only before input to the residual stream fully-connected layer.
3. **postnorm** ($y = LN(x + f(x))$): is the original transformer layer norm placement, applied after residual recombination.
4. **bothnorm** ($y = LN(x + f(LN(x))$): combines prenorm and postnorm, i.e. applies 2 layer norms per residual block: one applied on the input to the residual stream, and the other applied after residual recombination.

We design an experiment to measure how easily an input state embedding will pass unchanged to the policy output layer for each layer norm placement described above. To do this, we sample 1000 256-dimensional embeddings from a standard Gaussian and pass it through a 24-layer linear residual network, constructed as described earlier. We then evaluate the cosine alignment of the input embedding to each layer's feature embedding, to see how much noise is injected as the embedding progresses through the residual blocks. We show in Fig. 13 that the only layer norm placement that maintains high alignment to the input embedding as a function of depth is the prenorm placement, or identity map re-ordering. In particular, we can see that the postnorm, bothnorm and to a lesser extent nonorm placements exhibit a very rapid decay of initial feature alignment, meaning by the 12th layer extracting information directly from the state becomes much more difficult. In contrast, prenorm is almost always aligned to the initial embedding, with barely any degradation throughout the entire 24-layers.