A Appendix

A.1 Ethical considerations

DataMUX has the potential to provide great computational efficiency when used in real production systems, where queries across different users can be aggregated to get simultaneous predictions for different users. However, this also present potential risks and scope for misuse. First, the multiplexing layer might 'leak' data between different users, which could potentially lead to privacy concerns. Second, the output of an instance might be influenced by other instances in the multiplexing batch, which may present the possibility of black-box attacks to manipulate information, especially in the multi-user setting. Finally, DataMUX models predictions could be harder to interpret with current techniques since the model's internal representations depend on the set of instances it was multiplexed with.

We believe these are important and interesting research problems to solve technically and may also require careful policy development for deploying these models (e.g. restricting multiplex batches to single users).

A.2 Limitations

Here, we list some of the limitations of this work:

- **Convergence time increases with increasing number of instances:** Training multiplexed models with large number of instances takes larger number of iterations for convergence as increasing number of instances increases the difficulty of multiplexing and demultiplexing. We hope that future work can explore different multiplexing and demultiplexing strategies to improve rate of convergence during training.
- **Multiplexing on CNNs and MLPs not as strong as that of the Transformer:** While we demonstrate non-trivial multiplexing capabilities for the CNN and MLP architectures with our approach, the results are not as strong as that of the Transformer architecture. We hope future work will develop better multiplexing approaches for these architectures.

A.3 Theoretical construction of multiplexing transformer

Following the same notations defined in Section 4.4, assume that the components of a multiplexed input, $\mathbf{u}_t^{(1)}, \ldots, \mathbf{u}_t^{(N)}$, all approximately lie in N linearly independent subspaces $\mathcal{D}^1, \ldots, \mathcal{D}^N$. Equation 6 is realizable when eigenvectors of ${\boldsymbol{W}_i^V}^\top \boldsymbol{W}_i^V$ can be grouped into N non-overlapping subsets $\{\boldsymbol{r}_1^{(1)}, \ldots, \boldsymbol{r}_m^{(1)}\}, \cdots, \{\boldsymbol{r}_1^{(N)}, \ldots, \boldsymbol{r}_m^{(N)}\}$, where $\boldsymbol{r}_\ell^{(k)}$ are orthonormal vectors (since the Gramian is real symmetric), and span the same input subspaces. In this case, the vector after linear transformation $\boldsymbol{W}_i^V$ can be expressed as a superposition of N vectors in N independent subspaces $\mathcal{D}_V^1, \ldots, \mathcal{D}_V^N$. We now verify this statement.

Proof. Since eigenvectors of ${\boldsymbol{W}_i^V}^\top \boldsymbol{W}_i^V$ span the same subspaces, we can write each component of $\mathbf{w}_t^{1:N}$ as a linear combination of the corresponding subset of eigenvectors,

$$\mathbf{w}_t^{1:N} = \sum_k \mathbf{u}_t^{(k)} = \sum_{k,\ell} \alpha_{\ell,t}^{(k)} \boldsymbol{r}_\ell^{(k)}. \tag{9}$$

Let $\boldsymbol{W}_i^V = \boldsymbol{L}\boldsymbol{\Sigma}\boldsymbol{R}^\top$ be the singular decomposition of $\boldsymbol{W}_i^V$, where $\boldsymbol{L} \in \mathbb{R}^{d_V \times d_V}$, $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ are two orthogonal matrices, and $\boldsymbol{\Sigma}$ is a $d_V \times d$ rectangular diagonal matrix with non-negative real numbers on the diagonal. For each column of $\boldsymbol{R}$ that contributes to input subspaces, e.g., $\boldsymbol{r}_\ell^{(k)}$, we denote the dual left singular vector at the corresponding column of $\boldsymbol{L}$ as $\boldsymbol{l}_\ell^{(k)}$, and the singular value at the corresponding column and row $\sigma_\ell^{(k)}$

$$\boldsymbol{v}_{i,t} = \boldsymbol{W}_i^V \mathbf{w}_t^{1:N} = \boldsymbol{L}\boldsymbol{\Sigma}\boldsymbol{R}^\top \sum_{k,\ell} \alpha_{\ell,t}^{(k)} \boldsymbol{r}_\ell^{(k)} \tag{10}$$

$$= \sum_{k=1}^{N} \left(\sum_\ell \alpha_{\ell,t}^{(k)} \sigma_\ell^{(k)} \boldsymbol{l}_\ell^{(k)} \right). \tag{11}$$

Since $\boldsymbol{l}_\ell^{(k)}$ are columns of $\boldsymbol{L}$ that are orthogonal to each other, $\boldsymbol{v}_{i,t}$ above is a superposition of N vectors $\boldsymbol{v}_{i,t}^{(k)}$ in independent subspaces, and each subspace is defined by

$$\mathcal{D}_k^V \cong \texttt{span}\{\boldsymbol{l}_\ell^{(k)}\}. \tag{12}$$

■

In addition to decompress-able value vectors, we can set the linear transformations for queries and keys to have the same singular space structures to prevent complicated interference. More specifically, assume that $\boldsymbol{W}_i^Q$ and $\boldsymbol{W}_i^K$ have some subsets of right and left singular vectors such that

$$\mathcal{D}_k \cong \texttt{span}\{\boldsymbol{r}_\ell^{Q(k)}\} \cong \texttt{span}\{\boldsymbol{r}_\ell^{K(k)}\}, \tag{13}$$

and

$$\mathcal{D}_k^K \cong \texttt{span}\{\boldsymbol{l}_\ell^{Q(k)}\} \cong \texttt{span}\{\boldsymbol{l}_\ell^{K(k)}\}. \tag{14}$$

The inner product between the query and keys can be rewritten as

$$\left(\boldsymbol{W}_i^K \mathbf{w}_{t'}^{1:N}\right)^\top \boldsymbol{W}_i^Q \mathbf{w}_t^{1:N} \tag{15}$$

$$= \left(\sum_{k,\ell} \beta_{\ell,t'}^{(k)} \sigma_\ell^{K(k)} \boldsymbol{l}_\ell^{K(k)}\right)^\top \left(\sum_{k,\ell} \gamma_{\ell,t}^{(k)} \sigma_\ell^{Q(k)} \boldsymbol{l}_\ell^{Q(k)}\right) \tag{16}$$

$$= \sum_{k=1}^{N} \left[\sum_{\ell,\ell'} \beta_{\ell',t'}^{(k)} \gamma_{\ell,t}^{(k)} \sigma_{\ell'}^{K(k)} \sigma_\ell^{Q(k)} \left(\boldsymbol{l}_{\ell'}^{K(k)}\right)^\top \boldsymbol{l}_\ell^{Q(k)}\right] \tag{17}$$

$$= \sum_{k=1}^{N} \tau_{t,t'}^{(k)} \tag{18}$$

where $\tau_{t,t'}^{(k)}$ is a scalar only depending on the k-th input sequence (for simplification we omit head index i). Thus, the self-attention operation at each position can be seen as retrieving values based on the average of query-key similarity scores of N sequences.

$$head_i(t) := \sum_{t'} \left[\frac{\exp\left(\sum_k \tau_{i,t,t'}^{(k)} / \sqrt{d_K}\right)}{\sum_{t''} \exp\left(\sum_k \tau_{i,t,t''}^{(k)} / \sqrt{d_K}\right)} \sum_k \boldsymbol{v}_{i,t}^{(k)}\right]. \tag{19}$$

The average retrieval by soft-max could be undesirable. However, this will not affect the quality of decompression at all. If we want perfect non-interference in retrieval, one always has an option to specialize each head to only focus on one input sequence, by setting $\tau_{i,t,t'}^{(k')} = 0$ for all $k' \neq k$. This is easily achievable by controlling singular values of $\boldsymbol{W}_i^Q$ or $\boldsymbol{W}_i^K$.

Finally, we concatenate all heads and linearly project the output with $\boldsymbol{W}^O \in \mathbb{R}^{d \times h d_V}$ to a d-dimensional space again. This step is exactly equivalent to having h projection matrices $\boldsymbol{W}_i^O \in \mathbb{R}^{d \times d_V}$ acting on different heads and aggregating resulting vectors.

$$output(t) = \boldsymbol{W}^O \left(\|_{i=1}^h head_i(t)\right) = \sum_{i=1}^{h} \boldsymbol{W}_i^O head_i(t) \tag{20}$$

Employing a similar structure for right singular vectors as $\boldsymbol{W}_i^V$, we can make each $\boldsymbol{W}_i^O$ also preserves the independence of subspaces.

A.4 Multiplexing with Hadamard + Ortho strategy trains consistently for different runs

In Figure 7b we show MNLI performance for the Hadamard + Ortho multiplexing strategy with error bars across 3 random seeds. We observe that the value of N has no effect on variance in performance across seeds, with all values showing minimal variance. Since we use the same retrieval pre-trained weights, the random seed only affects the demultiplexer and classification head initializations.

A.5 Alternate multiplexing strategies

We experiment with different multiplexing strategies and use index embeddings for demultiplexing. For the Hadamard product, we try unfreezing the random Gaussian vectors and update through optimization ("Learned"). We also experiment with binary masking, where the i^{th} binary vector selects the i^{th} chunk of size d/N from the input representation of instance i ("Binary"). We first look

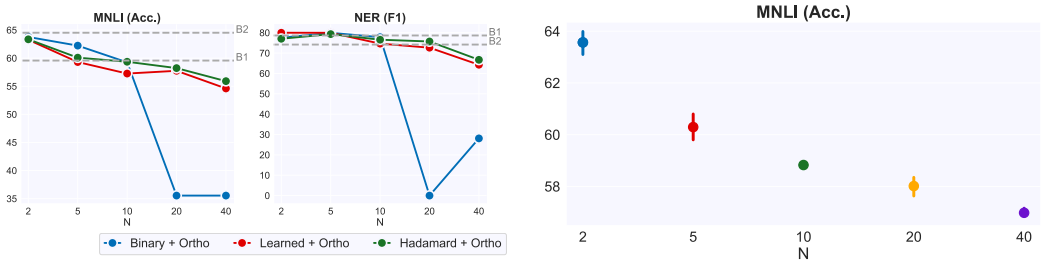

Figure 7: (Left) Evaluation on alternative superposition methods for multiplexing. Binary vectors fail to multiplex beyond 10 instances, while unfreezing Gaussian vectors for multiplexing does not help. **(Right)** MNLI accuracy results with error bar for the Hadamard + Ortho strategy across 3 different random seeds. Variance in performance is minimal to none.

at the performance on the retrieval warm-up task. Figure 4b shows that the unfreezing the vectors does not significantly change performance. We also observe that binary vectors fails to multiplex for large N, suggesting the multiplexing is more capable than just concatenating multiple downsampled inputs of d/N dimension when N is large. We see similar trends for the MNLI and NER (illustrated in Figure 7a) with unfrozen vectors not impacting performance and binary vectors failing to multiplex for large N.

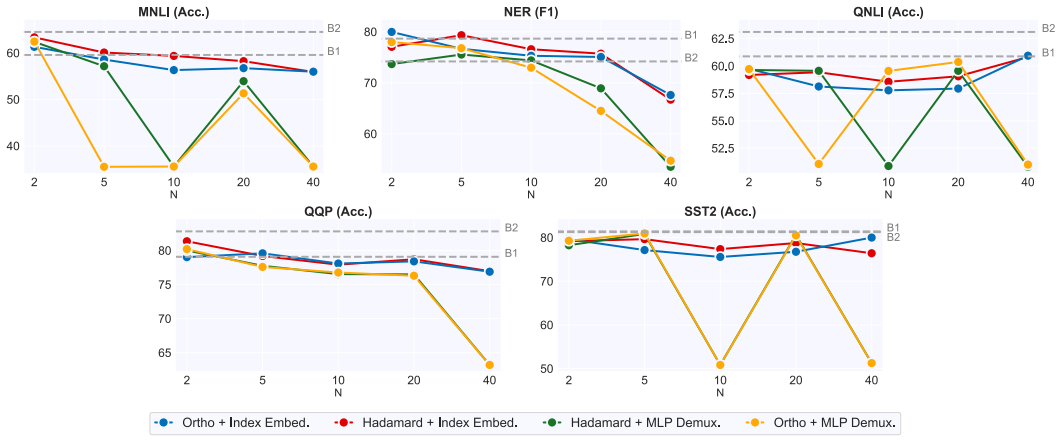

Figure 8: Primary results for NLP tasks mirroring those shown in Figure 3. We additionally show results for the MLP Demux. strategy. While MLP Demux. demultiplexing very works well for retrieval, shown in Figure 4b, we find that it typically performs slightly worse than the Index Embedding method and it leads to unstable optimization. This method also leads to an increase in parameter size proportional to N, since we must now train N independent MLPs for each input index.

A.6 MLP Demux Results

We show our primary results for NLP tasks in Figure 3, though we've excluded models using the MLP Demux. demultiplexing method. We provide these results instead in Figure 8 to highlight the optimization instability encountered during training of models using the MLP Demux. method. Especially curious is the failure to converge at apparently arbitrary points for N, such as converging for $N = 20$ for MNLI yet not converging for $N = 10$ despite $N = 10$ being a presumably simpler setting.

A.7 Smaller models achieve good performance across tasks

For our evaluation tasks, our base model might be over-parametrized and smaller models might perform equally well. Figures 9 shows performance on MNLI and NER as we vary the hidden size and the number of layers in the transformer. We observe that smaller models are competitive on both tasks and in the following section explore the possibility of getting higher throughput by multiplexing smaller models.

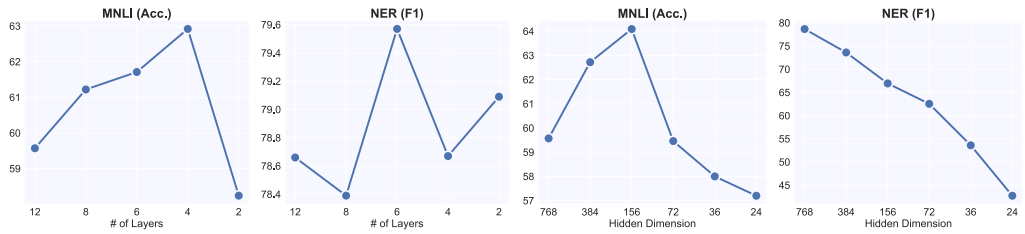

Figure 9: Model performance for different hidden dimension sizes and number of layers.

A.8 Experiment details for computing throughput

We compute throughput over 20000 samples on the MNLI task on a single Nvidia RTX 2080 machine. We use 4 batch sizes and take the maximum value.

A.9 Implementation details for multiplexing Transformers

We train all models to convergence. We use a learning rate of $2e-5$ and $5e-5$ for baselines and report the best performance. For multiplexed models, we use a learning rate of $5e-5$. However, for large N, we use $2e-5$ in case learning rate of $5e-5$ does not converge. We use a batch size of 32 for the baselines and use slightly smaller batch sizes for multiplexing as multiplexing effectively increase our size of the batch and therefore we need to keep more input instances in memory, leading to a drop in batch size. For the language tasks, we report numbers on the validation split and do not perform any extensive hyper-parameter sweeps.

A.10 Experiment Design details for CNNs and MLPs

Each image is cropped as 20×20 pixels at the center and trained with standard stochastic gradient decent. We also do not apply weight decay or other regularization as these are orthogonal to the multiplexing setting.

Principle component analysis on all 60000 training images indicates that 86.54% variance can be explained by top 50 principle components, suggesting that if we only keep the top 50 dimensions for each input image and project them into linearly independent subspaces, ideally we can multiplex $8 (d = 400/50)$ inputs as just one input without much information loss. We test the model performances with $N = \{1, 2, 4, 8, 16\}$ multiplexed inputs.

Our MLP consists of a hidden layer with 100 neurons, a demultiplexing layer that maps the 100 hidden neurons to $20 \times N$ neurons, and a shared linear readout layer that maps each group of 20 neurons to 10 categories for classification. Our CNN is similar to the classic LeNet LeCun et al. (1989), consisting of three convolutional layers with 10 activation maps from 3x3 kernels, 16 activation maps from 4x4 kernels and 120 activation maps from 3x3 kernels, one linear layer maps to 84 hidden neurons, a demultiplexing layer that maps them to $84 \times N$ neurons, and a shared linear readout layer that maps each group of 84 neurons to 10 categories for classification. The first two convolutional layers are followed by 2x2 max pooling. Every linear layer or convolution layer in our models is followed by a $\tanh$ activation. Labels are $+1$ for the correct digit and -1 for the incorrect digits. And we use the mean squared error (MSE) loss to train all our models.

We train all our models with the standard stochastic gradient decent with fixed learning rates, and the batch size is 32.

To generate the low-rank approximations ("LowRank" in Figure 6a), we divide d random orthogonal row vectors into N groups, and multiplying them by another $d \times d$ orthogonal matrix.

A.11 Other multiplexing strategies for CNNs

To make the multiplexing method more compatible with CNNs, we first tried some simple separation functions for images. 2D rotations (SO(2)) work much better than SO(d) when $N \leq 2$. When $N = 1$, rotating inputs certainly does no harm to the performance. When $N = 2$, there is some unavoidable interference between two rotated images, while CNN can still easily distinguish inputs with decent accuracy. However, when $N > 2$, inputs are heavily overlapped, and CNN fails to

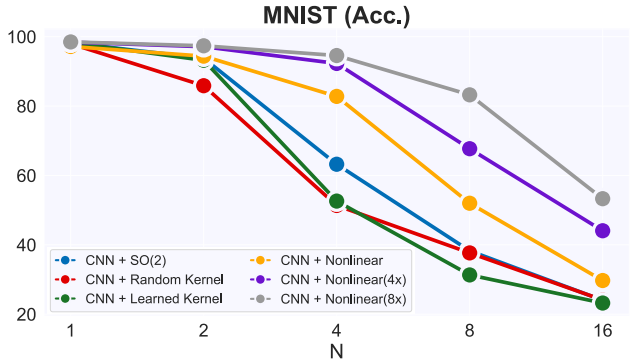

Figure 10: Averaged test accuracy for multiplexing CNN for $N = \{1, 2, 4, 8, 16\}$ inputs on the MNIST classification task, with different multiplexing strategies. Results are stable across multiple runs.

distinguish many digits. The accuracy among different inputs also varies largely, with std. at 10-24% when $N \geq 8$, indicating the permutation symmetry of inputs is hard to preserve during optimization. We also tried other simple 2D transformations like mosaic and downsampling so that all inputs are perfectly separated but get blurred. CNN can only answer one of the inputs correctly for this mosaic transformation, since we are only testing the vanilla convolutional architecture, which is not suitable for object detection without proposing bounding boxes.

Figure 10 summarizes the performance of other separation functions we used for multiplexing CNN that can preserve spacial locality of images. During multiplexing, we can slide a 3x3 kernel over each input images before summing them up. This make CNN able to distinguish different inputs even with random initialized weights from $\mathcal{N}(0, 1)$ (CNN+Random Kernel). We also make the weights of all N multiplexing kernels learned but the difference is subtle (CNN+Learned Kernel), and the its performance is worse than a multiplexing CNN with 2D rotations. Also, we find that these multiplexing CNN can always answer at most two inputs correctly each time. This is because sliding a small kernel over image is a very constrained linear transformation that cannot do much to separate images, and the permutation symmetry has to be broken during the training dynamics to improve accuracy.

To increase the expressibility of separation functions while keep it compatible with CNN, we apply N small convolutional nets with two layers of 16 3x3 kernels and tanh activation to input images, and sum up their activation maps (CNN+Nonlinear). A multiplexing CNN with this nonlinear separation function is similar to the MIMO approach in the previous study Ramé et al. (2021); Havasi et al. (2021), and we observe the performance changes consistent with the literature. When $N \leq 4$, its test average test accuracy is above 80%, which is also better than multiplexing CNN with SO(d). However, when $N > 4$, the performance drops rapidly. The accuracy across inputs becomes stabler with std. at about 7% when $N \geq 8$. If we allow higher dimensionality of the multiplexed input over a single input, that is use 4 activation maps (CNN+Nonlinear(4x)) or 8 activation maps (CNN+Nonlinear(8x)) instead of using a single activation map, we can keep improving the accuracy for larger N while still providing improved throughput.

A.12 Empirical analysis of interference among multiplexed instances

To understand how the representation of an instance changes with respect to the other samples it is multiplexed with, we randomly select 10 samples, $x_1, x_2, \cdots x_{10}$ from the MNLI dataset and multiplex the instance with 30 different sets of instances $y_1^i, y_2^i, \cdots, y_{N-1}^i$ for $i \in [1 \cdots 30]$. This generates 30 different demultiplexed representations for each of the 10 selected samples. We then visualize the resulting 10 clusters by reducing the dimension of the resulting 768 dimensional demultiplexed representation to a 200 dimensional vector with PCA (F.R.S. (1901)) and then further reducing the dimension to 2 with t-SNE (van der Maaten and Hinton (2008)). Figure 11 visualizes these clusters for different value of N. We find that across different values of N, all the points in the clusters are very close to each other, which suggests that the representation of an instance is not significantly influenced by the set of instances it is multiplexed with. Interestingly, contrary to our prior intuition, the clusters for N = 20, 40 are very separable. Investigating further, we find that the pairwise cosine similarity between vectors in different clusters is close to 0 for high values of

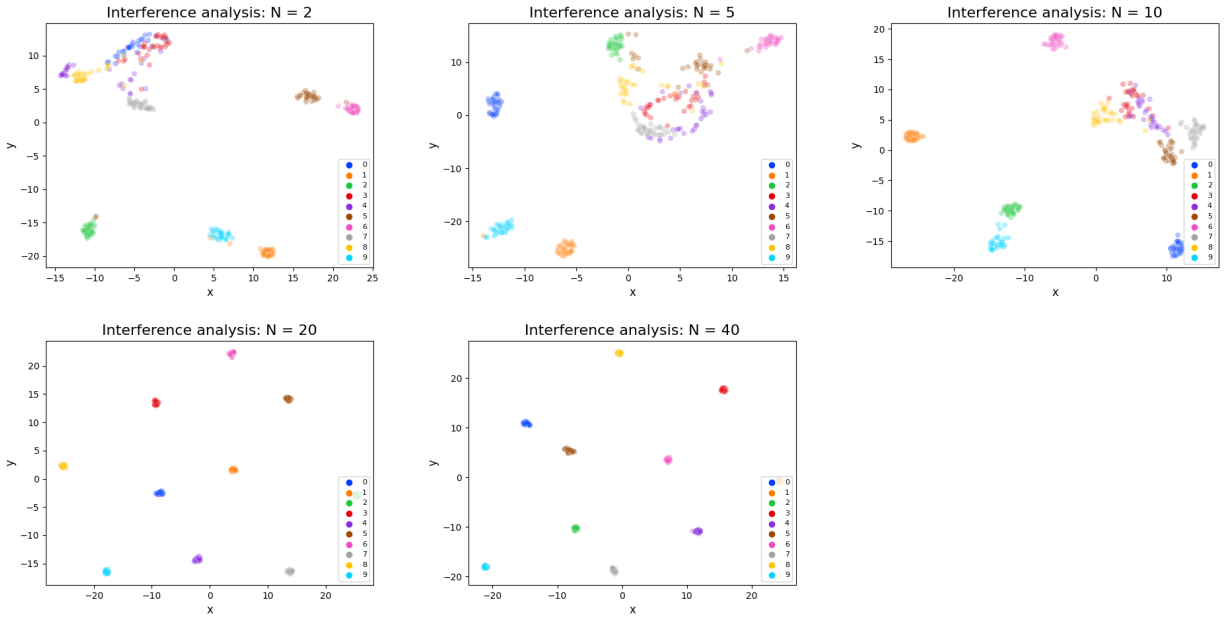

Figure 11: T-SNE plots to understand how the demultiplexed representation of an instance changes with respect to the set of instances it is multiplexed with. Across different N, we find that the demultiplexed representation of an instance is not significantly impacted by the set of instances it is multiplexed with.

N, suggesting that the network is forcing the learned representations of different instances to be orthogonal to each other.

A.13 Memory overhead for multiplexed models

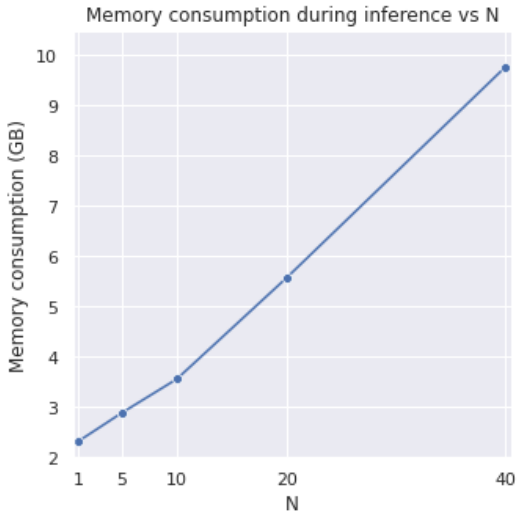

Figure 12: Memory overhead of multiplexed models during inference increases linearly with increasing N, with a very gentle slope (Memory overhead for N = 40 is $\sim$ 4x than N = 1).

We measure the memory overhead of various multiplexed models in Figure 12. We use a fixed minibatch size of 60 for all N and measure GPU memory during inference. We use the index demultiplexing strategy along with the Hadamard multiplexing strategy. We note that the memory increases linearly with increasing N as the number of inputs to the demultiplexing layers increases linearly with increasing N. However, the rate of growth is very gentle and the memory for N = 40 grows only by a mere $\sim$ 4x compared to the baseline model.