Compared to treating individual convolutional layers or the entire CNN as the encoding unit, partitioning the network into \textit{conv blocks} for distributed coded computation achieves a better trade-off between computational and communication overhead. Due to the local dependency of convolution where each element in the output feature map depends only on a local region of the input, subtask partitioning based on output feature maps naturally introduces overlapping between the corresponding input partitions. If the entire CNN is treated as a single unit for partitioning, such distributed inference method has the minimal communication overhead, the cascading effect of too many stacked convolutional layers leads to significant overlapping between the intermediated results in the shallow layers of CNNs, bringing large redundancy computation overhead between the fused-layer subtasks. On the other hand, decomposing ecah individual convolutional layer independently eliminates redundant computation overhead, but it introduces frequent inter-device synchronizations, leading to high communication overhead.

Therefore, to strike a better balance between computation and communication in distributed inference, we propose partitioning the convolutional layers of CNN into \textit{conv blocks} as the minimal unit for distributed coded computation.

We observe that pooling layers are a major source of the increasing overlapping across input partitions of original split subtasks. \rev{Specifically, pooling operations expands the receptive field of subsequent layers, thereby increasing the dependency range of output partitions on the input.} Based on this insight, we partition CNN into multiple \textit{conv blocks} separated by pooling layers.

Each \textit{conv block} consists of multiple convolution layers, along with their associated activation functions and normalization layers. Notably, BatchNorm is essentially equivalent to a linear transform. Together with activation function, they perform element-wise operations that do not alter the input-output dependency structure. Referring to the previously mentioned notations, we define the input feature map of a \textit{conv block} as $\mathbf{I}\in\mathbb{R}^{B\times C_\text{I}\times H_\text{I}\times W_\text{I}}$. Give the shape of the block input $\mathbf{I}$, we can determine the shape of the block output $\mathbf{O}\in\mathbb{R}^{B\times C_\text{O}\times H_\text{O}\times W_\text{O}}$ based on the configuration of convolutional layers within the block and equation \eqref{eq:input-output}. Following a similar approach as in prior work, we partition the expected output $\mathbf{O}$ along the width dimension into k partitions $\mathbf{Y}*1,\dots,\mathbf{Y}k$, each serves as the target output of an original subtask. Similarly, an output partition $\mathbf{Y}i$ ($i\in[n]$) is defined by a range $(a\text{O}, b\text{O})$ along the width dimension of $\mathbf{O}$, with $a*\text{O}\geq 0$ and $b_\text{O}\leq W_\text{O}$. For an output partition $\mathbf{Y}*i$ with range $(a*\text{O}, b_\text{O})$, we can determined the corresponding partition range (a_I, b_I) associated with the required input partition $\mathbf{X}_i$ within the original block input $\mathbf{I}$ by iteratively computing the input range of the previous layer using equation \eqref{eq:output-input} backwards.

By applying this process to all output partitions, we decompose a \textit{conv block} into k parallelizable subtasks, each associated with a matched input-output pair of equal size. This subtask partitioning strategy enables efficient and balanced distributed execution of CNN inference, while maintaining a nice trade-off for both computational redundancy and communication overhead.

\subsection{Encoder and Decoder}

Encoding and Decoding

To generate redundant subtasks based on the original split subtasks of a \textit{conv blocks}, we introduce specially designed encoder-decoder pair using \textit{sparseMLP}. In general, the encoder takes k equal-size split input partitions

$\mathbf{X}_1,\dots,\mathbf{X}*k$ of the \textit{conv block} as input and generates r encoded input partitions $\Tilde{\mathbf{X}1},\dots,\Tilde{\mathbf{X}r}$, each of them has a shape of $(B,C\text{I},H\text{I},W^p*\text{I})$ due to the partition on the width dimension.

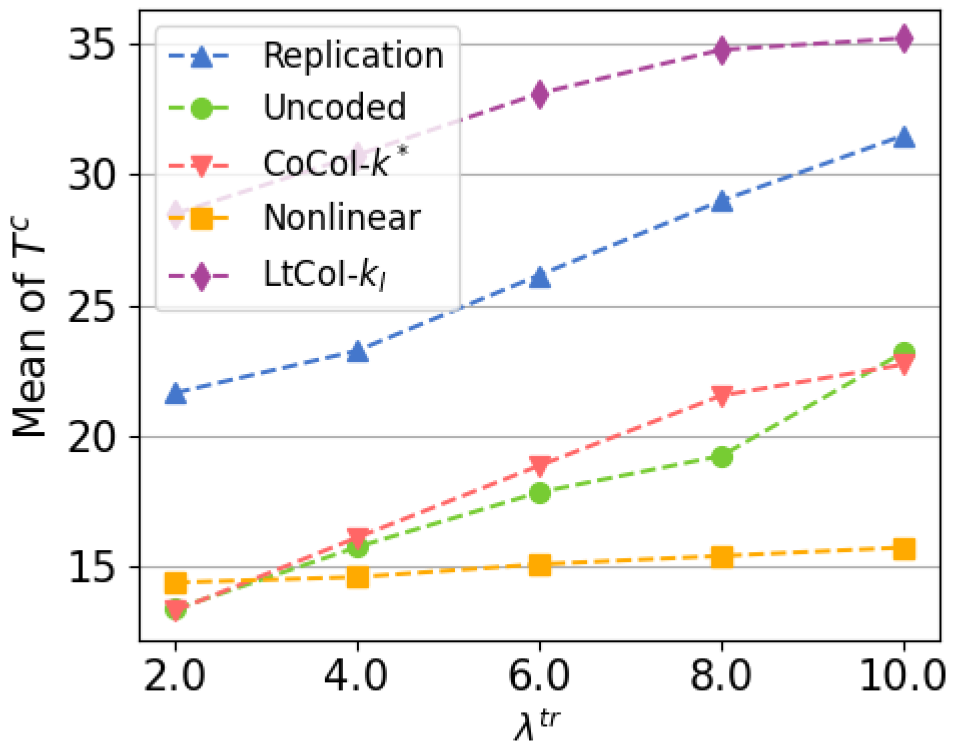
As for the decoder, we need a unified decoding scheme to handle different struggler cases of the subtask execution. Specifically, strugglers may occur to various number and different individual block subtasks. Therefore, the decoder takes $k+r$ encoded output partitions $\mathbf{Y}*1,\dots,\mathbf{Y}k$ and $\Tilde{\mathbf{Y}1},\dots,\Tilde{\mathbf{Y}r}$ as input, each with shape $(B,C\text{O},H\text{O},W^p\text{O})$, and generates the decoded output feature map of size $(B,C*\text{O},H_\text{O},W_\text{O})$, served as the block output $\mathbf{O}$. When some stragglers occur, the corresponding subtask outputs fed to the decoder are transformed to all-zero tensor, therefore the decoder does not obtain any information about the desired output corresponding to the straggler subtasks. Similar settings are also adopted in \cite{Kosaian2020learningbased}.

\textbf{Architecture of Encoder and Decoder.} Encoder composed of multiple convolutional layers (conv encoder) and decoder based on MLP (mlp decoder) have been proposed to approximate a non-linear erasure code to encode multiple CNN inference tasks \cite{Kosaian2020learningbased}. However, such design leads to inefficient coded inference in edge scenarios, especially for block-wise encoding. Specifically, the high FLOPs encoding through conv encoder leads to significant encoding overhead. While the MLP encoder/decoder necessitate a large scale of parameters to encode the intermediate \textit{conv blocks} due to the large intermediate results, even larger than the original CNN model, accounting for too much memory resources.

While linear erasure codes essentially perform simple linear transform with the input symbols, which motivates us that a non-linear erasure code may not need the full mapping from all element of the input symbols. To this end, we propose \textit{sparseMLP} encoder/decoder that incorporates the expressiveness of MLP and the weight-sharing feature of convolutional layers. A \textit{sparseMLP} encoder/decoder consists of multiple \textit{sparse-linear} layers, which specifies the output size of the encoder/decoder. Different from convolutional layers, a \textit{sparse-linear} layer applies different weight to compute the output of different positions (i.e., computing)

\textbf{Training Encoder-Decoder Pairs.} We simulate the straggling effects during the training of encoder-decoder pairs.

Latency & Robustness

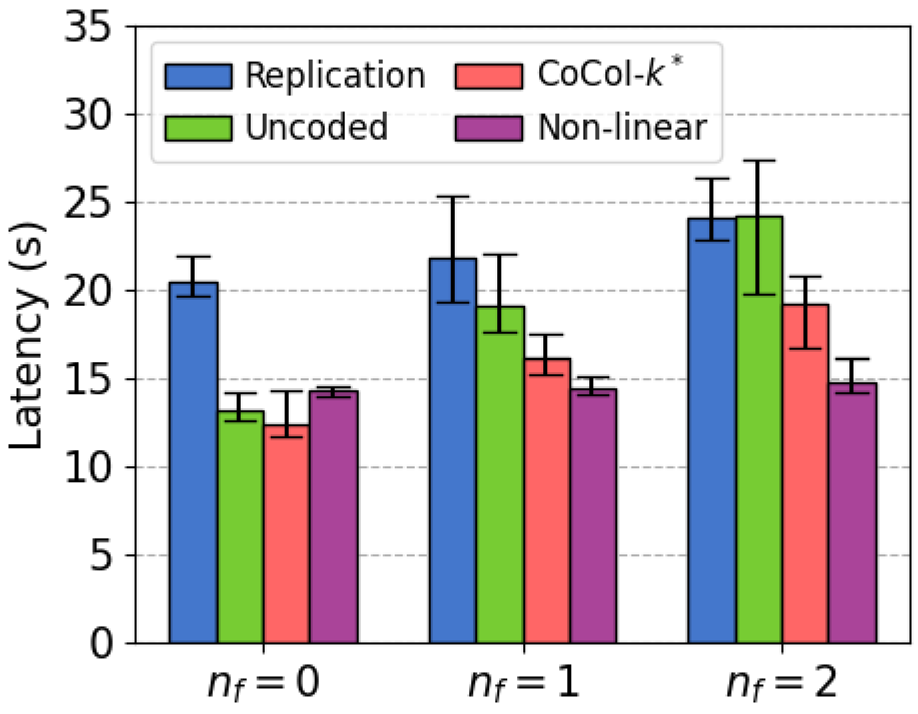


x

Fig. 8 shows the CNN inference latency in scenario-1

When λ^{tr} is small (e.g., $\lambda^{tr} = 2$), "Uncoded" and CoCol-k^* are slightly faster than Nonlinear due to the additional computational overhead of the neural encoder/decoder at each worker. However, under a moderate degree of straggling effect (e.g., $\lambda^{tr} \ge 4$), the proposed Nonlinear scheme significantly outperforms the linear benchmarks due to its high stability against transmission delays. The latency reduction can be up to 32.6% when $\lambda^{tr} = 10$ compared to the Uncoded scheme (and 50% compared to Replication).

we evaluate the performance of our proposed non-linear coded inference framework. Unlike linear approaches that ensure perfect recovery, the non-linear scheme trades off a marginal drop in inference accuracy for superior latency stability in highly volatile environments.



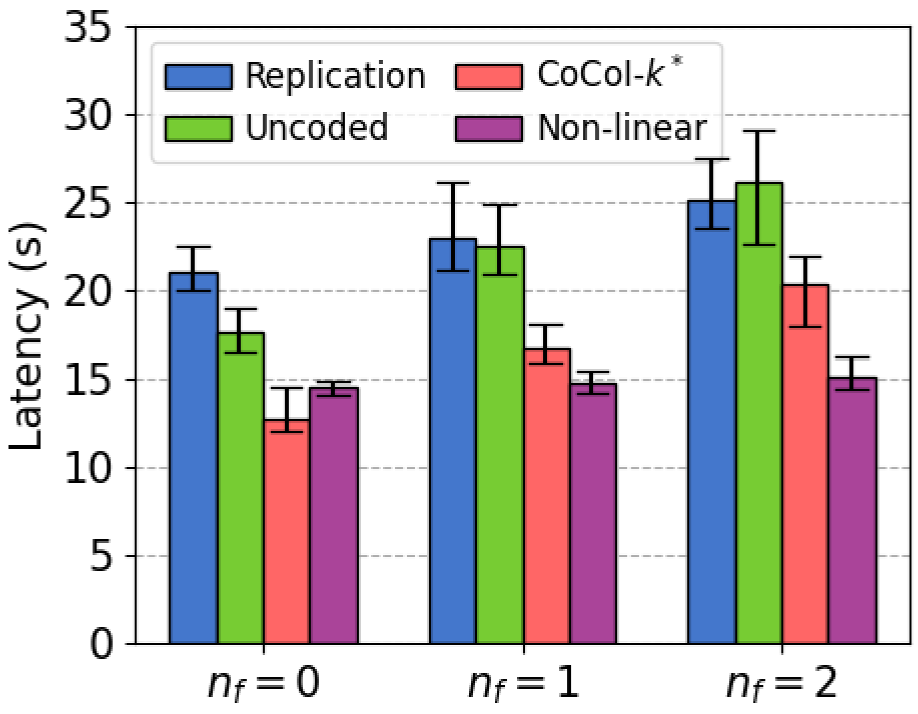
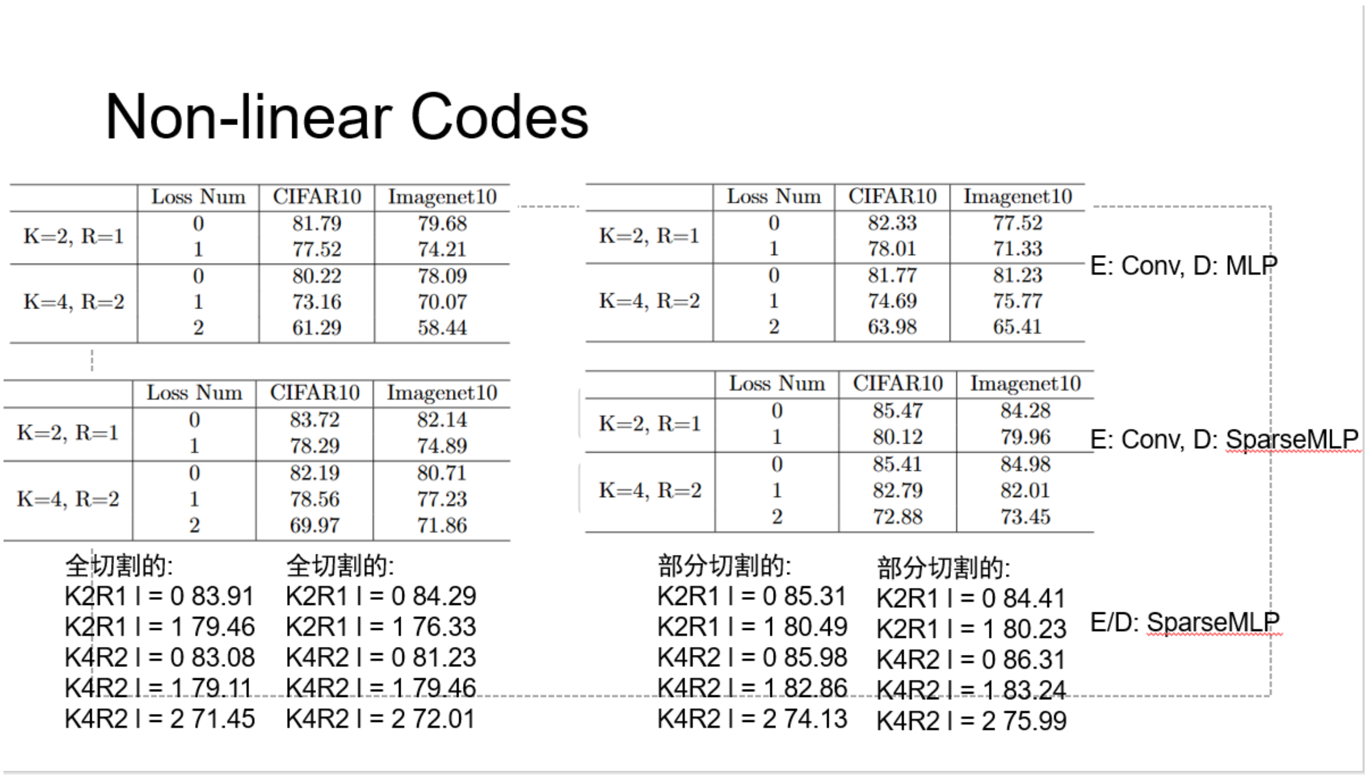


Fig. 9 presents the inference latency comparison under device failure scenarios ($n_f = 0, 1, 2$). The error bars indicate the variance of the inference latency. When there is no device failure ($n_f = 0$), the proposed Non-linear scheme is slightly slower than CoCoI-$k^$ *and "Uncoded" benchmarks. This is attributed to the additional computational overhead introduced by the neural encoder and decoder inference. However, as n_f increases from 0 to 2, the latency of "Uncoded" and Replication significantly deteriorates, increasing noticeable due to the necessary re-transmission or waiting for replicas. While CoCoI-k^* mitigates this rise better than the baselines, it still exhibits a noticeable latency increase as it waits for the k-th fastest result. In contrast, the Non-linear scheme demonstrates superior robustness; its latency remains around 15s regardless of the number of failures. Unlike linear codes which rely on exact reconstruction, the Non-linear approach effectively approximates the output using available partitions, eliminating the need for re-execution. Consequently, the Non-linear scheme achieves the most stable performance with negligible variance (indicated by the minimal error bars). When compared to the “Uncoded” and Replication baselines under severe failure conditions ($n_f=2$), the Non-linear approach reduces inference latency by up to 37.5%.

Accuracy



Non-linear Codes

	Loss Num	CIFAR10	Imagenet10
K=2, R=1	0	81.79	79.68
	1	77.52	74.21
K=4, R=2	0	80.22	78.09
	1	73.16	70.07
	2	61.29	58.44

	Loss Num	CIFAR10	Imagenet10
K=2, R=1	0	82.33	77.52
	1	78.01	71.33
K=4, R=2	0	81.77	81.23
	1	74.69	75.77
	2	63.98	65.41

E: Conv, D: MLP

	Loss Num	CIFAR10	Imagenet10
K=2, R=1	0	83.72	82.14
	1	78.29	74.89
K=4, R=2	0	82.19	80.71
	1	78.56	77.23
	2	69.97	71.86

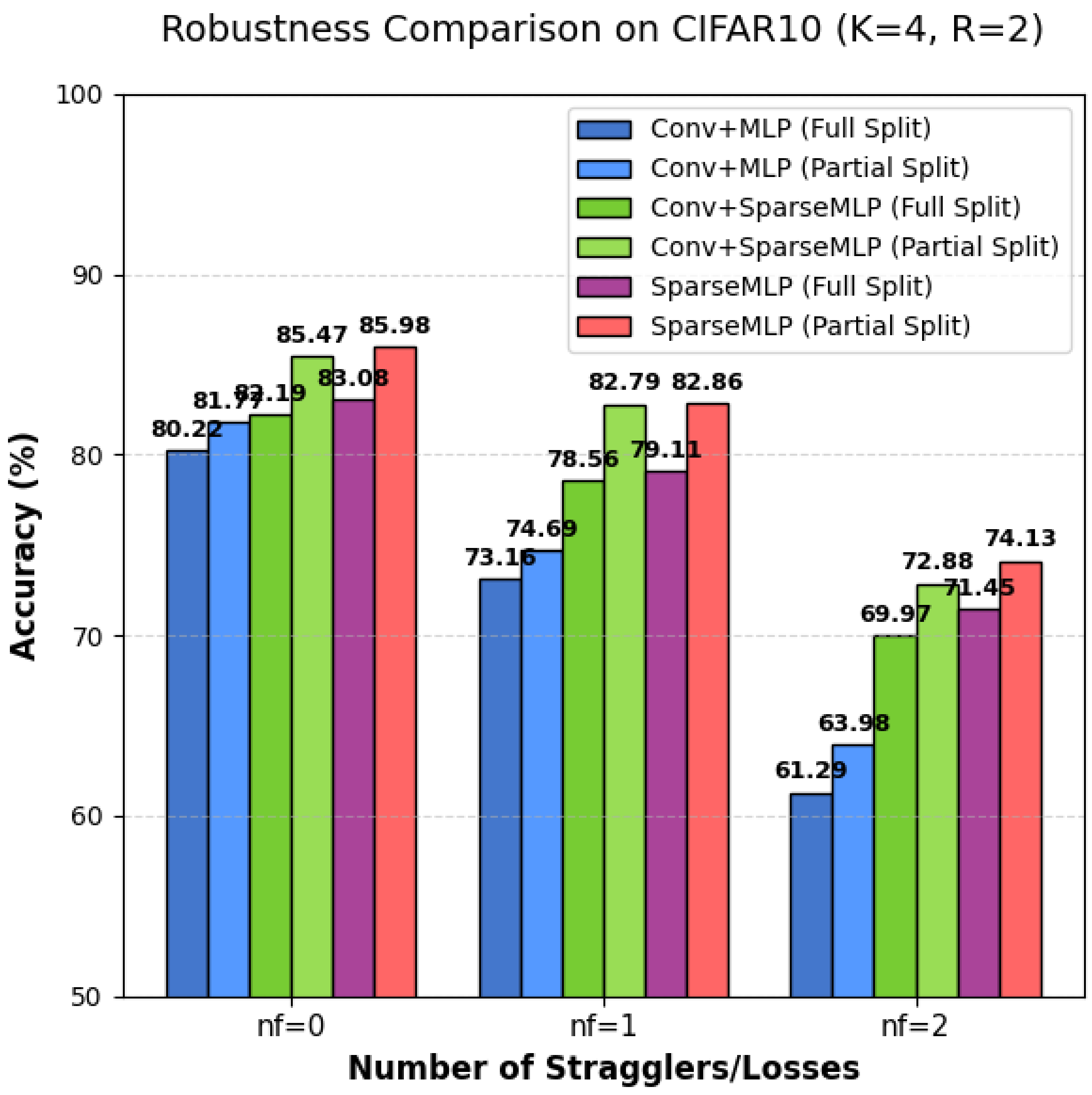
	Loss Num	CIFAR10	Imagenet10
K=2, R=1	0	85.47	84.28
	1	80.12	79.96
K=4, R=2	0	85.41	84.98
	1	82.79	82.01
	2	72.88	73.45

E: Conv, D: SparseMLP

全切割的:	全切割的:	部分切割的:	部分切割的:
K2R1 l = 0 83.91	K2R1 l = 0 84.29	K2R1 l = 0 85.31	K2R1 l = 0 84.41
K2R1 l = 1 79.46	K2R1 l = 1 76.33	K2R1 l = 1 80.49	K2R1 l = 1 80.23
K4R2 l = 0 83.08	K4R2 l = 0 81.23	K4R2 l = 0 85.98	K4R2 l = 0 86.31
K4R2 l = 1 79.11	K4R2 l = 1 79.46	K4R2 l = 1 82.86	K4R2 l = 1 83.24
K4R2 l = 2 71.45	K4R2 l = 2 72.01	K4R2 l = 2 74.13	K4R2 l = 2 75.99

E/D: SparseMLP

部分切割部分?


Robustness Comparison on CIFAR10 (K=4, R=2)
Conv+MLP (Full Split)
Conv+MLP (Partial Split)
Conv+SparseMLP (Full Split)
Conv+SparseMLP (Partial Split)
SparseMLP (Full Split)
SparseMLP (Partial Split)
80.22
81.77
82.19
85.47
83.08
85.98
73.16
74.69
78.56
82.79
79.11
82.86
61.29
63.98
69.97
72.88
71.45
74.13
100
90
80
70
60
50
Accuracy (%)
nf=0
nf=1
nf=2
Number of Stragglers/Losses

直接比较全切割的

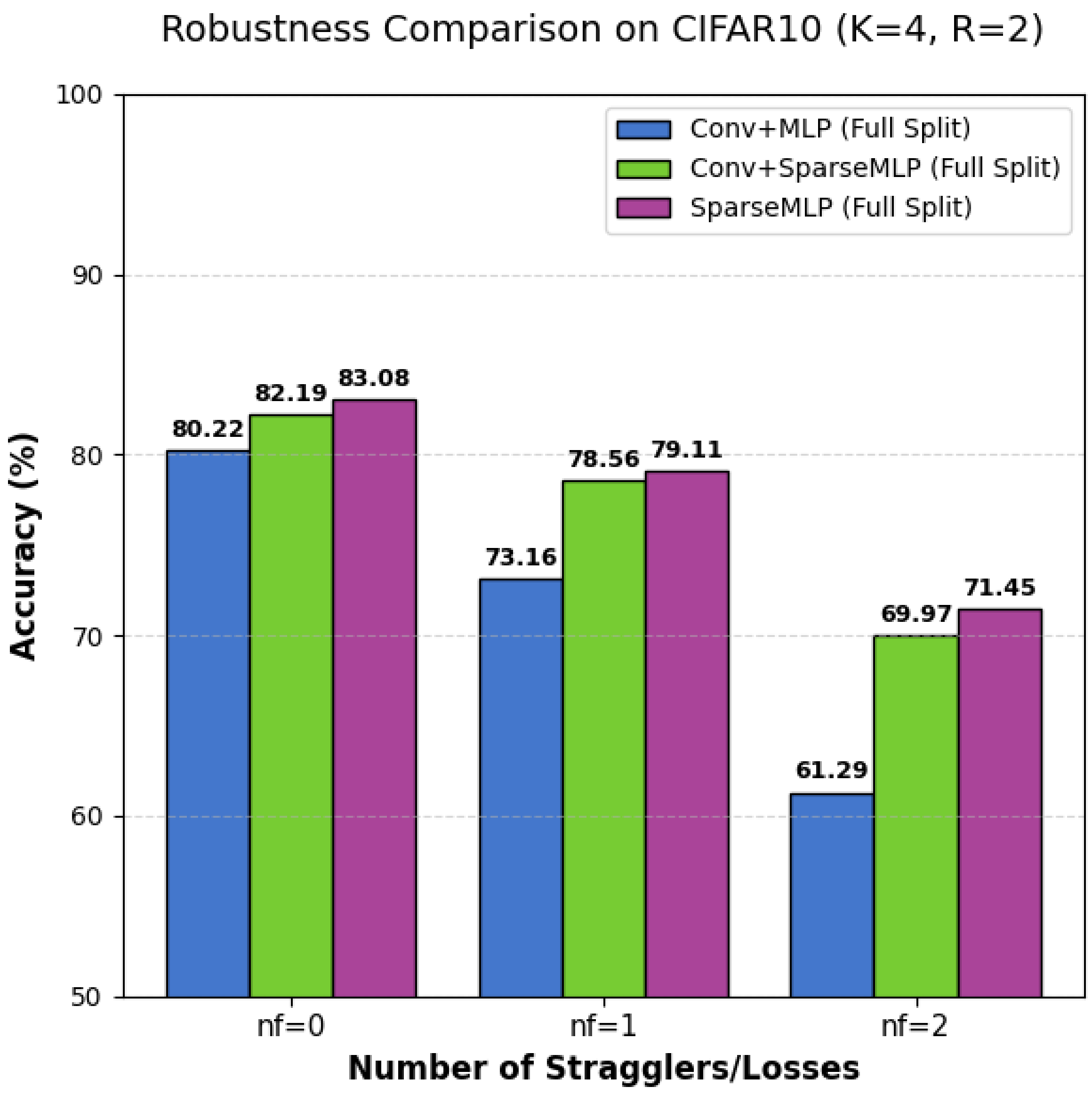


Fig. 9 evaluated three architectures (i) Convolutional Encoder with MLP Decoder, (ii) Convolutional Encoder with SparseMLP Decoder, and (iii) SparseMLP for both. Experimental results show that the SparseMLP-based E/D pair achieves the highest reconstruction quality.