Appendix A Proof of the *X*-Partition Theorem

Proof. Let $h = \lceil Q/Y \rceil$, $Y = X - R(S) + T(S)$. We first construct a segmentation sequence that satisfies $Q(V_1) = \ldots = Q(V_{h-1}) = Y$, $Q(V_h) \leq Y$. The I/O complexity of V_i can be expressed as the sum of input complexity $Q_{input}(V_i)$ and output complexity $Q_{output}(V_i)$, that is, $Q(V_i) = Q_{input}(V_i) + Q_{output}(V_i)$.

Since the directed acyclic graph represents a program that cannot be recalculated, there is no overlap between node sets, so condition **I** is satisfied. And because in the directed acyclic graph, a node in V_i is colored red only when its parent nodes in $V_j (j \leq i)$ is colored red, condition **II** is satisfied. Since $In(V_i)$ represents the input set, there is $In(V_i) \subseteq H_{R,i} \cup H_{BR,i}$. According to the conditions of the theorem, we can get $|H_{R,i}| \leq R(S)$, $|H_{BR,i}| \leq Q_{input}(V_i) = Q(V_i) - Q_{output}(V_i) \leq Y - |W_{R,i}| = Y - T(S)$. The inequality (18) holds. Condition **III** is satisfied.

$$|In(V_i)| \leq |H_{R,i}| + |H_{BR,i}| \leq R(S) + Y - T(S) = X \tag{18}$$

In the same way, for the output set $|Out(V_i)|$, we can also get $|W_{R,i}| \leq R(S)$, $|W_{RB,i}| \leq Q_{output}(V_i) = Q(V_i) - Q_{input}(V_i) \leq Y - |H_{BR,i}| = Y - T(S)$. so inequality (19) holds. Condition **IV** is satisfied.

$$|Out(V_i)| \leq |W_{R,i}| + |W_{RB,i}| \leq R(S) + Y - T(S) = X \tag{19}$$

Appendix B Method for Determining the Size of Sub-Computational Domain

The content of this appendix partially overlaps with a manuscript presently under consideration by another journal. To avoid potential conflicts, it is temporarily withheld and will be released after the review process has concluded.

Appendix C I/O Theory For Network Architecture Search

We solve the problem by building a graph neural network. Leveraging the characteristics of graph neural networks, we divide the neural network unit structure determination into two subtasks: identifying the node types on the computation graph and predicting the connections between these nodes.

Prediction of Operations on Nodes We focus on CNN unit structures composed of operator nodes with fixed types: Conv3, Conv5, AvgPool, and MaxPool. To simplify the experimental setup, we frame the determination of node types on the computational graph as a graph neural network-based node classification task. Following Hieu Pham et al.'s [1] exploration, CNN unit nodes have two

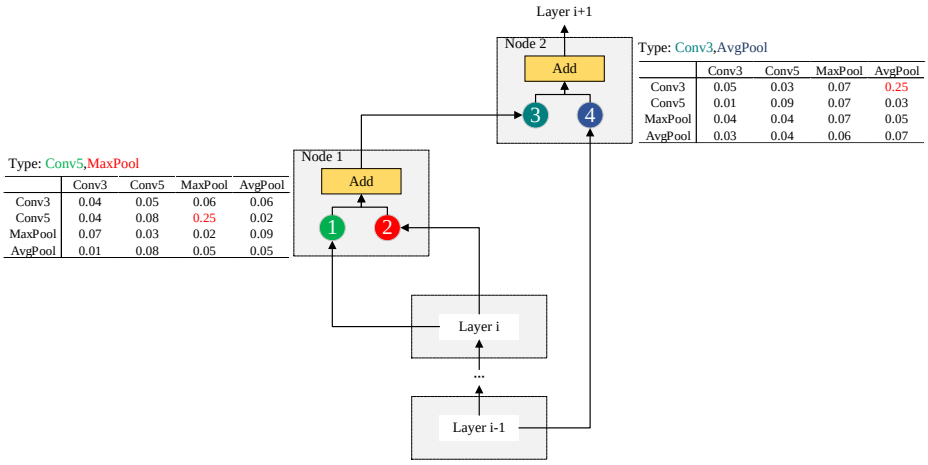

Fig. 3. By selecting the type combination with the highest probability from each node's probability vector, we determine the operation type for each node based on the two inputs and subsequently construct the CNN unit structure.

inputs, necessitating the determination of two operation types per node. Given four possible operation types per input, 16 combinations are possible. Using a graph neural network, we output a probability matrix for each node, selecting the combination with the highest probability to determine the operation types for the two inputs as Fig. 3.

Prediction of Links between Nodes In building neural network units, predicting links on the computational graph is crucial. Using CNNs as an example, we introduce the link prediction process. Through the graph convolutional neural network, nodes obtain a probability vector indicating their correlation with other nodes. Since CNNs have two inputs per node, we must consider both input sources. Similar to node type prediction, we predict the combined probability of the two inputs. As Fig. 4 demonstrates, node 3 has four possible combinations, and we choose the group with the highest probability as the predecessor nodes for the inputs. Notably, if the predecessor nodes and operation types of both inputs are identical, they are merged into a single input. Finally, we eliminate redundant nodes to establish effective connections between all nodes.

Loss Function For shared parameter neural networks, the cross-entropy loss function as Eq. (22) guides the gradient-based optimization of shared network parameters. Concurrently, we aim to optimize the controller predicting node type and link structure. This loss function comprises two parts, with $\mathcal{L}_1$ representing cross-entropy on the verification set as Eq. (20). Here, M denotes the number of categories, while y_{ic} serves as a sign function, assigning 1 when the true category

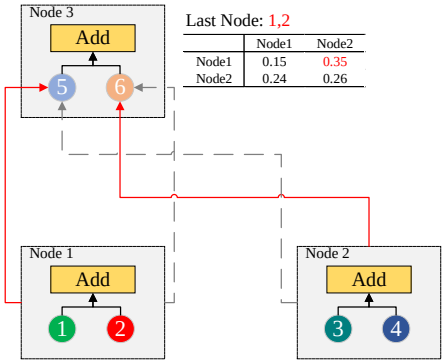

Fig. 4. When selecting a precursor node, CNN must consider the precursor paths of both inputs, necessitating the calculation of combination probabilities for informed decision-making.

of sample i matches c and 0 otherwise. p_{ic} denotes the predicted probability of sample i belonging to category c.

$$\mathcal{L}_1 = -\frac{1}{N}\sum_{i}\sum_{c=1}^{M} y_{ic} log(p_{ic}) \tag{20}$$

The $\mathcal{L}_2$ metric is used to assess the estimated I/O performance. For this evaluation, we utilized the I/O lower bound algorithm presented in Chapter 3, although it exhibits minor variations from the previously employed algorithm.

As the tensor size is constant in our unit structure design, we can directly fix the solution to the constrained optimization problem as $(g_1, g_2, \ldots, g_j) = (\max(X_1, X_2), 0, \ldots, 0)$. This optimal solution represents the size of the external input at each step. In unit structure design, estimating the I/O overhead Q_e within the unit is crucial, calculated as Theorem 2 in the paper. Considering that the input and output of this task are fixed, we only need to consider the possible impacts that may occur in the intermediate process. Leveraging the designed generation function, we can approximately compute Q_e using Eq. (21).

$$Q_e = \sum_{j=2}^{n} \Phi_{j-1}(k) + \sum_{j=1}^{n} (\psi_j(k) - \phi_j(k)) \tag{21}$$

It is noteworthy that, contrary to the algorithm for computing the I/O lower bound, we will randomly generate the initial capacity of the Cache multiple times and calculate the average of the resulting values. This average is given by the formula $\mathcal{L}_2 = avg(\sum Q_e)$.

We introduce the hyperparameter α to establish a balance between the two quantities' order of magnitude. Specifically, α is determined by the reciprocal of the first calculated lower bound of I/O, i.e., $\alpha = \frac{1}{\mathcal{L}_2^0}$. To optimize the con-

troller, we employ the REINFORCE algorithm, an approximate gradient update strategy.

$$\mathcal{L} = \mathcal{L}_1 + \alpha\mathcal{L}_2 \tag{22}$$

Platforms and Hyperparameters Utilizing the NVIDIA GeForce RTX 4090 GPU with 24 GB video memory, the setup uses a graph convolutional neural network with a learning rate of $3.5e{-}4$ and Adam optimizer. A shared-parameter CNN, limited to 12 nodes, operates at a learning rate of $5e-3$ with the same optimizer. During training, the shared-parameter model undergoes 100 epochs with 100 iterations per epoch on CIFAR10 (CIFAR100: 101 epochs), evaluating I/O performance across 20 diverse, randomly generated margin scenarios using a consistent cache setup. The codebase builds upon the work by Pham H. et al. [1].

Limitations and Future Directions Based on our exploratory experiments, we believe that incorporating I/O lower-bound theorems into neural network architecture search may offer clear benefits, particularly in enhancing the I/O efficiency of neural networks. However, due to limited computational resources, methodological constraints, and restricted experimental time, several important issues remain unresolved. Drawing on our experimental experience, we outline these limitations below and offer practical recommendations for future researchers.

- **The NAS code takes too long to run.** In fact, although our experiments were conducted for about 100-101 rounds, it often took 1-2 days to obtain experimental results on an NVIDIA GeForce RTX 4090 GPU. Such a long training time is unbearable.
- **The experimental effect has a certain degree of randomness.** In our experiments on the CIFAR-10 and CIFAR-100 datasets, we identified certain network architectures that outperformed those reported in the original paper, as well as others that performed poorly. However, due to inherent randomness, the search process for these architectures could not be reliably reproduced once the neural network parameters were modified, potentially causing the most promising architectures to be missed. For researchers interested in extending this work, we recommend at least adopting a Top-K selection strategy to prevent overlooking potentially superior architectures during the search process. In addition, exploring new methods to mitigate randomness may further enhance the stability and reproducibility of the results.
- **Perhaps restricting the search space would make more sense than designing a search strategy.** Although we believe that the introduction of the I/O lower bound theorem can bring a new perspective to network structure search to a certain extent, randomness will still affect the performance

of the algorithm. We strongly recommend that subsequent researchers consider conducting research from the perspective of limiting the search space, which may achieve better results than simply designing strategies.

References

1. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International conference on machine learning. pp. 4095–4104. PMLR (2018)