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 (1) holds. Condition **III** is satisfied.

$$|In(V_i)| \leq |H_{R,i}| + |H_{BR,i}| \leq R(S) + Y - T(S) = X \tag{1}$$

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 (2) holds. Condition **IV** is satisfied.

$$|Out(V_i)| \leq |W_{R,i}| + |W_{RB,i}| \leq R(S) + Y - T(S) = X \tag{2}$$

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

Given that the subcomputation process encompasses multiple steps, the determination of the size of subcomputation $\mathcal{H}_{max}$ in Eq.(2) emerges as a crucial aspect in assessing the algorithm's I/O lower bound. To address this, we have drawn inspiration from the methodologies outlined by Zhang et al. [1] and further refined them. Prior to delving into the size of subcomputation, we introduce the concepts of multi-step segmentation and vertex generation, as defined in Definition 1-2 from [1].

Definition 1. *$G_1(U_1, E_1), \ldots, G_h(U_h, E_h)$ is a multi-step segmentation of $G(V, E)$ if and only if any input node of $G_j(U_j, E_j)$ must be the output nodes of $G_{j-1}(U_{j-1}, E_{j-1})$, and U_i are disjoint.*

Definition 2. *In a DAG $G(V, E)$, a vertex set U can generate another vertex set U' if and only if every path from an input of V to a vertex in U' contains a vertex in U. $\theta(U)$ contains the set of vertices that can be generated by U.*

We define two vertex generation functions as Tan et al. [2]. We assume that $\widetilde{O}_j$ is the output set of U_j. For any integer k and a vertex U with a dominant set D, inequality(3) is satisfied. The dominator set D is the set of vertices where every path entered to any vertex in U contains at least one of those vertices.

$$|D \cap U_j| + |\theta(D) \cap \widetilde{O}_{j-1}| \leq k \tag{3}$$

$\varphi_j(k)$ and $\psi_j(k)$ respectively indicate that in the j-th calculation, when the number of input vertices is k, the maximum number of vertices in U_j and $\tilde{O}_j$ is generated as Eq.(4)-(5) and Fig. 1 (a)-(b).

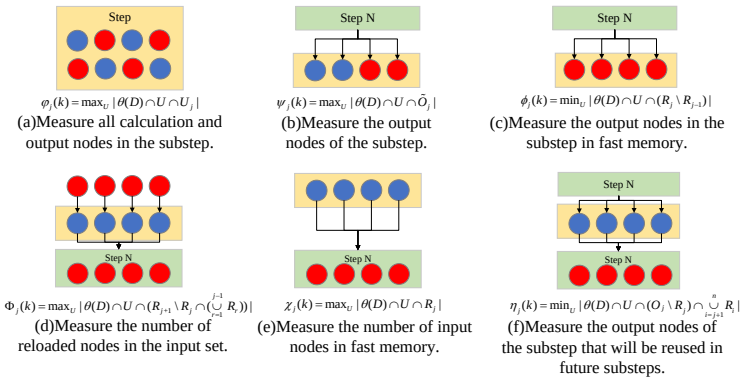
$$\varphi_j(k) = \max_U |\theta(D) \cap U \cap U_j| \tag{4}$$

$$\psi_j(k) = \max_U |\theta(D) \cap U \cap \tilde{O}_j| \tag{5}$$

To facilitate the further introduction of the introduced generation function, we define the reuse set as Definition 3.

Definition 3. *The reuse set R_j is defined as the set of red nodes (retained in fast memory) before the j+1-th step of the subcomputation as Eq.*(6)*. n is defined here as the total number of steps. $maxset(T, S)$ is a function used to find the maximum set that satisfies $|T| < S$.*

$$R_j = \begin{cases} maxset(H_{R,j} \cap (\cup_{r=j+1}^{n} I_r), S) & j = 0 \\ maxset(R_{j-1} \cup \widetilde{O}_j \cap (\cup_{r=j+1}^{n} I_r), S) & j \geq 1 \end{cases} \tag{6}$$



(a)Measure all calculation and output nodes in the substep.
(b)Measure the output nodes of the substep.
(c)Measure the output nodes in the substep in fast memory.
(d)Measure the number of reloaded nodes in the input set.
(e)Measure the number of input nodes in fast memory.
(f)Measure the output nodes of the substep that will be reused in future substeps.

Fig. 1. Schematic diagram of the six generating functions

$\phi_j(k)$ represents the minimum number of set nodes with red pebbles at the end of the j-th process of the subcomputation and are used in subsequent steps of that subcomputation as Eq.(7) and Fig. 1(c).

$$\phi_j(k) = \min_U |\theta(D) \cap U \cap (R_j \backslash R_{j-1})| \tag{7}$$

The generating functions $\Phi_j(k)$ and $\chi_j(k)$ are utilized to characterize specific memory-related metrics. Specifically, $\Phi_j(k)$ denotes the maximum number of nodes reloaded into memory during step $j+1$, while $\chi_j(k)$ represents the maximum number of nodes residing in the fast memory prior to step $j+1$. The mathematical expressions of these functions are detailed in Eq. (8) and Eq. (9), respectively. Additionally, their graphical representations are presented in Fig. 1(d) and (e).

$$\Phi_j(k) = \max_U |\theta(D) \cap U \cap (R_{j+1} \backslash R_j \cap (\bigcup_{r=1}^{j-1} R_r))| \tag{8}$$

$$\chi_j(k) = \max_U |\theta(D) \cap U \cap R_j| \tag{9}$$

To facilitate further modeling, we introduce Definition 4 to specify the number of input set nodes for each step, in conjunction with the definitions of the generating functions $\Phi_j(k)$ and $\chi_j(k)$.

Definition 4. *The input set for each step of a subcomputation with a composite process can be defined as a set satisfying Eq.*(10). *g_j is the number of subcomputation input nodes used in step j and will not be used in previous steps, i.e. $g_j = |G_j| = |I_j \cap I_0 \backslash (\cup_{0<i<j} I_i)|$ and $\forall i \neq j, G_i \cap G_j = \varnothing$.$\sum_{j=1}^{n} g_j \leq X$ is obviously established. I_0 represents the set of all input nodes of the subcomputation.*

$$|I_j| = \begin{cases} g_j & j = 1 \\ g_j + \Phi_{j-1}(|I_{j-1}|) + \chi_{j-1}(|I_{j-1}|) & j \geq 2 \end{cases} \tag{10}$$

We further strengthen the definition of k from inequality(3) to inequality(11).

$$|D \cap U_j| + |\theta(D) \cap I_j| \leq k \tag{11}$$

We introduce the vertex generation function $\eta_j(k)$ to represent the minimum number of output elements that will be reloaded into fast memory in the future, as defined in Eq. (12) and Figure 1(f). This function facilitates the analysis of potential I/O patterns in subcomputations.

$$\eta_j(k) = \min_U |\theta(D) \cap U \cap (\widetilde{O}_j \backslash R_j) \cap \bigcup_{i=j+1}^{n} R_i| \tag{12}$$

Using prior definitions and generation functions, we convert the I/O lower bound of the subcomputation involving composite processes into a constrained optimization problem, as outlined in Theorem 1, leveraging Definition 1.

Theorem 1. *Assume that $\{G_1(U_1, E_1), \ldots, G_n(U_n, E_n)\}$ is a multi-step partition of a DAG $G(V, E)$. For any modified (X_1, X_2)-Partition of $G(V, E)$, $|V_i|$ has an upper bound as Eq.*(13).

$$|\mathcal{H}_{max}| = \max(X_1, X_2) + \max \sum_{j=1}^{n} (\varphi_j(|I_j|)) \tag{13}$$

Inequality(14)-(16) *are satisfied during the calculation.*

$$s.t. \sum_{j=1}^{n} g_j \leq \max(X_1, X_2) \tag{14}$$

$$\max_{j=1,\ldots,n} |R_j| \leq S \tag{15}$$

$$|R_n| + |\tilde{O}_n \backslash R_n| + \sum_{j=1}^{n-1} (\psi_j(|I_j|) - \phi_j(|I_j|) - \eta_j(|I_j|)) \leq \max(X_1, X_2) \tag{16}$$

Utilizing Theorem 1, we can achieve a compacter sub-computational domain size by solving a constrained optimization problem. By applying this result, we can determine the calculation scale and further estimate the I/O lower bound Q of the composite algorithm based on Theorem 2.

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 [3] exploration, CNN unit nodes have two 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. 2.

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. 3 demonstrates, node 3 has four possible combinations,

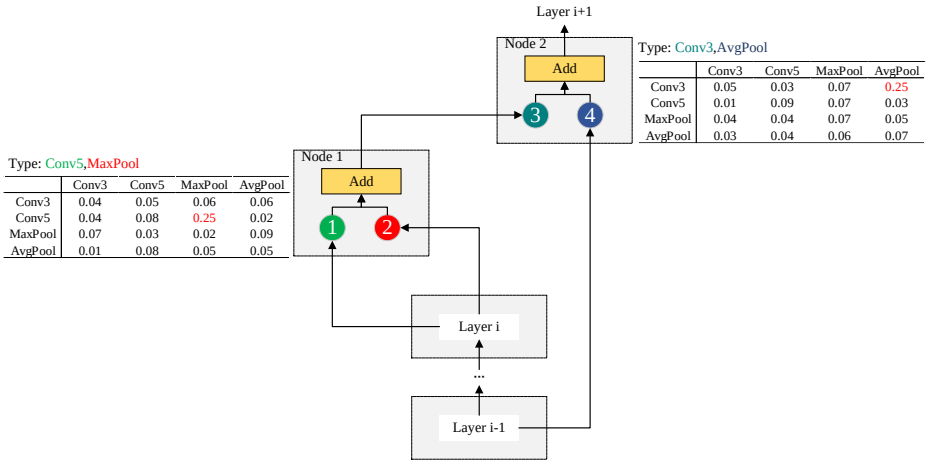


Fig. 2. By selecting the type combination with the highest probability from each node's probability vector, we determine the operation type for each node based on the two inputs and subsequently construct the CNN unit structure.

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. (19) 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. (17). Here, M denotes the number of categories, while y_{ic} serves as a sign function, assigning 1 when the true category of sample i matches c and 0 otherwise. p_{ic} denotes the predicted probability of sample i belonging to category c.

$$\mathcal{L}_1 = -\frac{1}{N}\sum_{i}\sum_{c=1}^{M} y_{ic} log(p_{ic}) \tag{17}$$

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, which is calculated as $\max(X_1, X_2) + \min\{-\mathcal{R}_1(S) +$

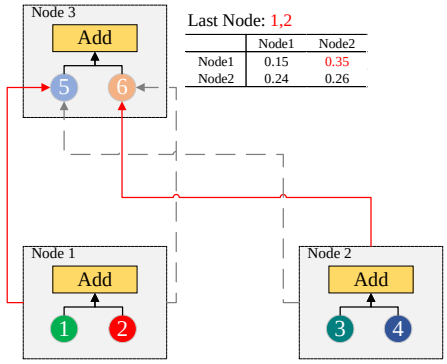


Fig. 3. When selecting a precursor node, CNN must consider the precursor paths of both inputs, necessitating the calculation of combination probabilities for informed decision-making.

$\mathcal{T}_2(S), -\mathcal{R}_2(S) + \mathcal{T}_1(S) + \mathcal{T}_3(S)\}$. Leveraging the designed generation function, we can approximately compute Q_e using Eq. (18).

$$Q_e = \max(X_1, X_2) + \sum_{j=2}^{n} \Phi_{j-1}(k) + \sum_{j=1}^{n} (\phi_j(k) - \psi_j(k)) \tag{18}$$

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, denoted as $\mathcal{L}_2$, 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 ratio between the accuracy achieved in the first epoch and the computed I/O lower bound, as $\alpha = \frac{\mathcal{L}_1^0}{\mathcal{L}_2^0}$. To optimize the controller, we employ the REINFORCE algorithm, an approximate gradient update strategy.

$$\mathcal{L} = \mathcal{L}_1 + \alpha \mathcal{L}_2 \tag{19}$$

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, 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. [3].

References

1. Zhang, X., Xiao, J., Tan, G.: I/o lower bounds for auto-tuning of convolutions in cnns. In: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 247—261. PPoPP '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3437801.3441609
2. Olivry, A., Langou, J., Pouchet, L.N., Sadayappan, P., Rastello, F.: Automated derivation of parametric data movement lower bounds for affine programs. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 808—-822. PLDI 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3385412.3385989
3. 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)