SPHYNX: RELU-EFFICIENT NETWORK DESIGN FOR PRIVATE INFERENCE

Minsu Cho[1], Zahra Ghodsi[2], Brandon Reagen[1], Siddharth Garg[1], and Chinmay Hegde[1]
[1] New York University, [2] University of California San Diego
mc8065@nyu.edu,zghodsi@ucsd.edu,bjr5@nyu.edu,sg175@nyu.edu,chinmayh@nyu.edu

ABSTRACT

The emergence of deep learning has been accompanied by privacy concerns surrounding users' data and service providers' models. We focus on private inference (PI), where the goal is to perform inference on a user's data sample using a service provider's model. Existing PI methods for deep networks enable cryptographically secure inference with little drop in functionality; however, they incur severe latency costs, primarily caused by non-linear network operations (such as ReLUs). This paper presents SPHYNX, a ReLU-efficient network design method based on micro-search strategies for convolutional cell design. SPHYNX achieves Pareto dominance over all existing private inference methods on CIFAR-100. We also design large-scale networks that support cryptographically private inference on Tiny-ImageNet and ImageNet.

1 Introduction

Deep learning inference is often outsourced to external cloud services to mitigate the high cost of executing state-of-the-art deep networks on user devices [1]. However, outsourced inference raises essential concerns about data and model privacy: users may not trust a cloud service provider with their data, and cloud service providers may not want to share their models, trained at enormous expense, with users. Private inference (PI) provides a solution to this problem by guaranteeing user and model privacy using cryptographic techniques [2, 3]. Under PI, a user can perform inference using a model hosted in the cloud without the cloud learning anything about her data and, conversely, without the user learning anything about the cloud's model parameters.

PI techniques thus far reported in the literature have leveraged a range of cryptographic protocols, including homomorphic encryption (HE), additive secret sharing (SS), and garbled circuits (GC). However, these all incur heavy computational overheads, resulting in several orders-of-magnitude increase in inference latency compared to standard "plaintext" inference. Prior work has demonstrated that *non-linear* network operations like the Rectified Linear Unit (ReLU) and max-pooling are the key bottlenecks. For example, Ghodsi et al. [4] estimate that ReLU layers in MiniONN [5] are four orders of magnitude more expensive than convolution layers, while in DELPHI [3], ReLUs account for 93% of ResNet32's online private runtime [6]. This is in direct contrast to standard (plaintext) inference where ReLUs and max-pools are effectively free and the dominant runtime costs are due to the floating-point operations (FLOPs) of convolutional and fully-connected layers.

With this in mind, recent efforts have sought to re-think deep neural network architectures that mitigate the dominant costs of ReLUs in PI run-time. Methods like DELPHI [3] and DeepReDuce [6] start with state-of-the-art architectures for standard plaintext inference (such as VGG or ResNets), and replace carefully selected ReLUs with quadratic activations (in the case of DELPHI) or with the identity function (in the case of DeepReDuce). SAFENet [7] extends DELPHI, substituting ReLUs channel-wise with polynomial activations of variable degree.

A parallel line of research, pioneered by CryptoNAS [4], is to *re-design* neural architectures from scratch for "PI-efficiency." This can be accomplished using neural architecture search (NAS) techniques with the objective of minimizing ReLU operations in place of traditional FLOP count optimization. Specifically, CryptoNAS executes a modified version of *macro-search* [8], i.e., over entire convolutional neural network models with arbitrarily many skip connections between layers. However, a long line of standard NAS research [9, 10, 11, 12, 13] has shown that searching over a smaller *micro-search* space of convolutional cells, which are then repeated across layers, yields more accurate networks for standard (plaintext) inference. In this paper we examine if the same is true in the case of private inference: can we outperform CryptoNAS (and other state-of-the-art PI methods including DELPHI, SAFENet, and DeepReDuce) via micro-search principles?

To this end, this paper proposes SPHYNX, a new framework for ReLU-efficient micro-search to design convolutional cells that maximize accuracy under a constraint on the number of ReLUs.

SPHYNX enables cell-based NAS algorithms (such as DARTS [10]) to discover novel *normal* and *reduce* cells for highly-accurate models that use significantly fewer ReLU/max pool operations. Moreover, unlike conventional micro-search techniques (which follow the search space introduced in NASNets [14]), SPHYNX also learns the best location of *reduce* cells, which turns out to be critical for minimizing ReLU costs. We perform extensive ablation studies to analyze the utility of each component of SPHYNX in detail. Overall, the innovations of SPHYNX are threefold:

1. We propose a new ReLU-efficient micro-search space for cell-based NAS. This new search space can be used in conjunction with any of several existing search strategies (such as DARTS, ENAS [9], PC-DARTS [11], GDAS [13], and GAEA [15]) to discover efficient *normal/reduce* convolutional cells.
2. We show that unlike traditional micro-NAS, where *reduce* cells are uniformly spaced, the precise location of *reduce* cells is critical in maximizing accuracy under ReLU constraints. To optimally position the *reduce* cells we develop a novel post-processing method based on the Gumbel-Softmax re-parameterization trick that helps further improve network performance.
3. We demonstrate the effectiveness of SPHYNX using image classification datasets including CIFAR-100 [16]], Tiny-ImageNet [17], and ImageNet [18] and demonstrate that SPHYNX outperforms several state-of-the-art solutions for PI.

2 Background on Private Inference

SPHYNX uses the DELPHI protocol [19] for PI under the same threat model and provides identical security guarantees. Specifically, DELPHI models two parties, a server that holds a trained model for an L layer deep network with parameters $\mathbf{W}_i$ and $\mathbf{b}_i$ for layer i ($i \in [0, L-1]$), and a client holds that holds input $\mathbf{y}_0$. The client seeks to obtain $\mathbf{y}_L$, computed layer-wise as follows: $\mathbf{y}_{i+1} = \sigma_i\left(\mathbf{W}_i \mathbf{y}_i + b_i\right)$, where σ_i is a non-linear activation function, ReLU is used here.

The parties are assumed to be semi-honest, i.e., they follow the protocol faithfully but try to infer information about the other party's input. That is, the client seeks to recover $\mathbf{W}_i$ and $\mathbf{b}_i$ and the server seeks to recover $\mathbf{y}_0$. The goal of PI is to prevent the parties from learning *anything* about the other parties data (other than what the client can learn from output $\mathbf{y}_L$).

DELPHI builds on three cryptographic primitives: additive secret sharing (SS) for linear layers, garbled circuits (GC) for non-linear ReLU layers, and homomorphic encryption (HE) used offline. We begin by briefly reviewing these primitives.

Additive Secret Sharing [20] allows two parties to hold *additive* shares $[x]_1$, $[x]_2$ of a secret value x (defined over a field F_p) such that $[x]_1 + [x]_2 = x$. The shares can be generated by sampling a random value r and setting $[x]_1 = r$ and $[x]_2 = x - r$.

Garbled Circuits [21] is a scheme introduced by Yao that allows two parties to compute a Boolean function f on their private inputs without revealing their inputs to each other. The function f is first represented as a Boolean circuit of two-input logic gates. One of the parties, the garbler, encodes (garbles) the circuit by encrypting the truth table of each gate in the circuit and sends the resulting "garbled circuit" to the other party, the evaluator. The other party, the evaluator, computes (or decrypts) the circuit gate-by-gate using encodings of the garbler's inputs and her own inputs, producing an encoding of the circuit's output. She shares this encoding with the garbler, who then reveals the corresponding plaintext.

Homomorphic Encryption [22] enables operations on encrypted values without a private key or decryption. A cryptosystem supports a homomorphic operation ($*$) if for public key (pk), secret key (sk), and ciphertexts $c_1 = \text{Enc}(pk, m_1)$, $c_2 = \text{Enc}(pk, m_2)$, there exists a function EVAL such that $\text{Dec}(sk, \text{EVAL}(c_1, c_2)) = m_1 * m_2$. A fully homomorphic cryptosystem supports arbitrary homomorphic additions and multiplications.

DELPHI is a hybrid protocol that uses different cryptographic primitives for linear and ReLU layers. Specifically, DELPHI uses HE (offline) and SS (online) for linear layers, and GC for ReLUs. We summarize these protocols below, assuming $\mathbf{b}_i = 0$ for simplicity.

Linear layers: During the offline phase, the client and server sample random vectors $\mathbf{r_i}$ and $\mathbf{s_i}$ respectively for linear layer i. The client encrypts and sends $Enc(pk, \mathbf{r_i})$ to the server. The server computes $Enc(\mathbf{W_i}.\mathbf{r_i} - \mathbf{s_i})$ homomorphically, and sends it back to the client. The client decrypts this ciphertext, and obtains $\mathbf{W_i}.\mathbf{r_i} - \mathbf{s_i}$, which will be used later in the online phase.

During the online phase, the client computes and sends $\mathbf{y}_0 - \mathbf{r_0}$ to the server. The client and server now hold secret shares of the client's input $\mathbf{y}_0$, or equivalently, the first layer's inputs. The server then computes $\mathbf{W_0}.(\mathbf{y_0} - \mathbf{r_0}) + \mathbf{s_0}$, its own share of the first layer's output. Correspondingly, the client's share of the first layer's output is $\mathbf{W_0}.\mathbf{r_0} - \mathbf{s_0}$, which was already obtained in the offline phase. It can be confirmed that the shares sum to $\mathbf{W_0}.\mathbf{y_0}$, i.e., the client and server each hold an additive secret share of $\mathbf{W_0}.\mathbf{y_0}$. The same protocol is used in subsequent layers allowing the client and server to obtain shares of each linear layer's outputs from shares of its inputs. Importantly, the only computation performed online is the server's computation of its share, which can be performed at roughly the same speed as the layer's plaintext computations [3].

ReLU layers: During the offline phase, the server garbles the circuit representing the ReLU function and sends it to the client, along with encodings of $\mathbf{r_{i+1}}$ and $\mathbf{W_i}.\mathbf{r_i} - \mathbf{s_i}$. During the online phase, the server sends the encodings of $\mathbf{W_i}.(\mathbf{y_i} - \mathbf{r_i}) + \mathbf{s_i}$ to the client, who is now able to evaluate the garbled circuit and send the encoded output to the server. The server decodes the garbled circuit output, which is $\mathbf{y_{i+1}} - \mathbf{r_{i+1}}$, the server's share for the next linear layer.

From the above discussion, we can observe that linear layers' online computations are effectively the same as plaintext computations and can be accelerated using standard CPU/GPU libraries. On the other hand, operations involving ReLU (or max-pool) layers use GCs that require expensive online crpytographic computations and interaction between parties, resulting in high latency. This motivates the need for a systematic method to design PI-efficient neural architectures that judiciously minimize ReLU computations, which we address next.

3 Limitations of Existing NAS Micro-Search Methods for Private Inference

Before introducing SPHYNX, we first analyze conventional NAS micro-search methods [14, 10, 11, 13, 12, 23, 24] through the lens of private inference. We conclude that networks found using existing micro-cell search are ill-suited for PI, motivating the need for SPHYNX. This is because existing architectures found via NAS micro-search use far too many ReLU operations, making such networks prohibitively slow for private inference.

We start with a NAS primer. Typical NAS micro-search techniques seek two types of *cells*: *normal* and *reduce* cells. Cells are essentially small networks of layers with convolutions, ReLUs, pools, batch norms (BN), and skip connections. *Normal* cells learn finer-scale features and their outputs have the same spatial resolution as their inputs. *Reduce* cells decrease spatial resolution (usually with stride 2).

Both types of cells are defined using a directed acyclic graph representation consisting of N nodes. Each node $x^{(i)}$ represents a feature map, and each directed edge (i, j) is associated with some operation $o^{(i,j)}$ that transforms $x^{(i)}$. For example, the cell representation used in DARTS [10] has N=7 nodes — two input nodes, one output node and four intermediate nodes — with eight operation edges. The output node concatenates features from all intermediate nodes. Each intermediate node is computed by summing all of its parent nodes: $x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$.

For concreteness, let us focus on the micro-search space used in DARTS [10]. Consider, first, the operation set $\mathcal{O}$. Conventionally, $\mathcal{O}$ is described as a set containing eight different operations: 3×3 and 5×5 separable or dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and *zero*. Here, each "convolution" operation is actually a sequence of ReLU-Conv-BN operations. Therefore, a feature map size $\{H_i, W_i, C_i\}$ in the i^{th} cell implies that each convolution operation computes $H_i \times W_i \times C_i$ ReLU operations.

However, additional costs emerge. Each cell takes two previous cells' output as inputs. Recalling that the cell output channel dimension is four times larger than its input, the output channel number requires a dimensionality reduction equivalent to the input dimension. To reconcile this difference, a particular *pre-processing* layer, a sequence of ReLU-Conv1×1-BN, is inserted to downscale the number of channels back to the desired resolution. For the i^{th} cell, this imposes additional $2 \times 4 \times H_i \times W_i \times C_i$ ReLU operations (2 and 4 comes from number of inputs and filters, respectively).

In this manner, we see that the DARTS search space imposes severe costs when viewed from the perspective of ReLU counts (a fact that seems to have been overlooked by traditional NAS methods, which optimize for FLOP counts). If we want to design a network satisfying a ReLU budget of 90K (which roughly translates to PI latency of 2 seconds using the DELPHI protocol) using cells reported by DARTS. Using even a very small network with an initial number of C=5 channels, even the first *normal* cell from this small network contains roughly 95K ReLU operations. Table 1 compares ReLU counts of cells reported by DARTS and PC-DARTS [11]; note that even small networks of depth D=4 and C=2 with PC-DARTS cells require more than 100K ReLU operations.

Network Setup	ReLU Counts	
	DARTS	PC-DARTS
C=36, D=20	7796K	8073K
C=5, D=10	523.5K	591.4K
C=5, D=5	231.6K	314.9K
C=2, D=4	73.2K	107.5K
C=1, D=4	36.6K	53.8K

Table 1: *ReLU count comparisons. 'C' and 'D' stand for initial channels and network depth, respectively;* C=36 *and* D=20 *are default hyperparameters to get good performance on CIFAR-10 with DARTS and PC-DARTS.*

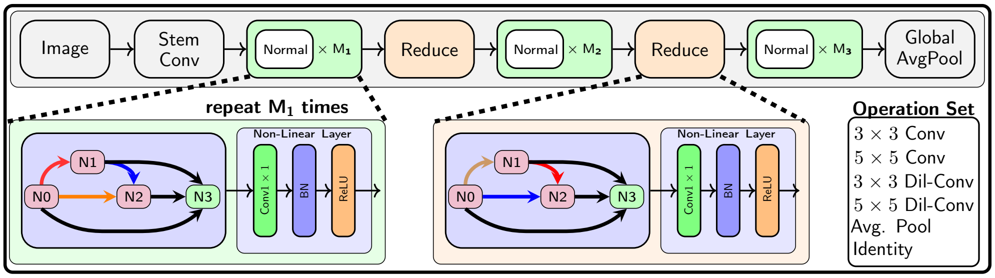


Figure 1: SPHYNX *search space network skeleton contains normal and reduce cells. The numbers of* M_1, M_2, *and* M_3 *represent the number of repeating normal cells, depending on the location of reduce cells. Note that* M_1=M_2=M_3 *in conventional micro-search NAS literature. For example, consider the final architecture with 8 cells and the possible reduce location is* $\{0, 1, 2, \ldots, 7\}$. *If the reduce cells are located at 1 and 3,* M_1=1, M_2=1, *and* M_3=4. *Note that normal and reduce cells do not contain ReLU operations, and 1×1 Conv-BN-ReLU modules follow both normal/reduce cell's end. Operation set does not include max-pooling operation due to its non-linearity.*

4 The SPHYNX Framework

We now propose SPHYNX, a new network design approach for efficient private inference. SPHYNX consists of three components: (i) a new ReLU-efficient search space, which can be combined with any existing search methods to discover promising *normal* and *reduce* cell architectures; (ii) an approach to select the initial number of channels (C) and network depth (D) in order to satisfy a ReLU budget; and (iii) a new stochastic optimization method to optimally discover the locations of *reduce* cells in terms of layer depth. We note that conventional NAS approaches also adopt (ii) (albeit while optimizing for FLOP budgets). To the best of our knowledge, (iii) is novel and could be of independent interest in NAS research.

4.1 SPHYNX: Search Space

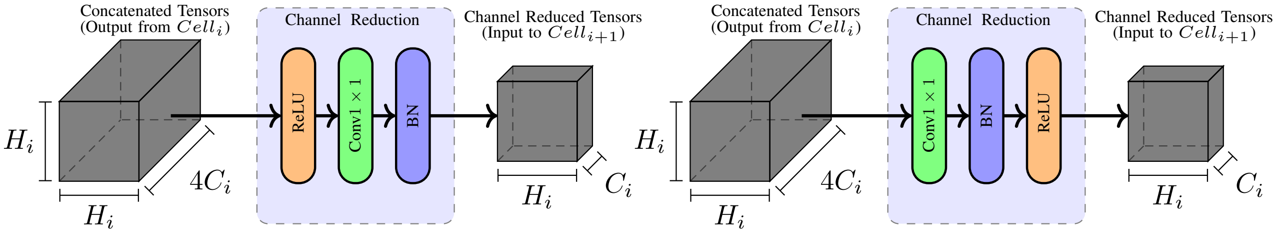
We first describe a new ReLU-efficient search space, where we redefine the skeleton of the network; see Figure 1. Compared to the traditional (DARTS) search space, we propose the following changes:

1. We eliminate the ReLU layer from convolution operations so each ReLU-Conv-BN sequence is now simply a Conv-BN sequence. We also replace separable convolutions with vanilla convolutions. Vanilla convolutions are more expressive than separable convolutions, at the expense of more FLOPs.[1]
2. We remove all max-pooling operations since these also require expensive GCs to compute. However, we retain average-pooling operations since they are linear and can be efficiently computed in DELPHI.
3. Instead of reducing the dimensionality of a cell's inputs via pre-processing, we instead reduce the dimensionality of a cell's outputs in a single *post-processing* step. Further, as shown in Figure 2, post-processing is performed using a Conv1×1-BN-ReLU sequence instead of ReLU-Conv1×1-BN so that the ReLU operate on a tensor with 4× few channels. The cell's outputs are forwarded the subsequent two cells.

Steps 1 and 2 remove all non-linear operations from the DARTS search space. The only non-linearity in our new SPHYNX search space is at the output of each cell, as described in Step 3.

In addition, SPHYNX follows the ReLU balancing rule introduced in CryptoNAS [4]. Unlike conventional FLOP balancing methods which doubles the channel size when the spatial resolution is halved, ReLU balancing quadruples channel size in order to distribute ReLU equally across layers. This strategy has been shown empirically to achieve higher accuracy for low ReLU budgets [4].

[1] In private inference, FLOP count is not a concern compared to ReLU counts.



(a) Module Sequence: ReLU-1×1 Conv-BN

(b) Module Sequence: 1×1 Conv-BN-ReLU

Figure 2: *Visualization of ReLU counts on two different module sequence. (a) The original convolution module sequence requires the ReLU count $4C_i \times H_i \times H_i$. (b) Our proposed search space applies ReLU layers, saving ReLU counts by a factor of four.*

Algorithm 1 Pseudocode: Finding location of reduce cells

```
**Inputs:** Split train dataset to $D_T$ and $D_V$.
**while** not $\hat{\boldsymbol{\beta}}$ not converged **do**
    Sample the minibatch $(\mathbf{x}_T, \mathbf{y}_T)$ from $D_T$.
    Sample the candidate network from $\boldsymbol{\beta}$ using Eq. 4.1.
    Calculate $\sum_{(x,y)\in(\mathbf{x}_T,\mathbf{y}_T)} \mathcal{L}(F_{\mathbf{w}^*,\boldsymbol{\beta}}(x), y)$ and update $\mathbf{w}$.
    Sample the minibatch $(\mathbf{x}_V, \mathbf{y}_V)$ from $D_V$.
    Sample the candidate network from $\boldsymbol{\beta}$ using Eq. 4.1.
    Calculate $\sum_{(x,y)\in(\mathbf{x}_V,\mathbf{y}_V)} \mathcal{L}(F_{\mathbf{w}^*,\boldsymbol{\beta}}(x), y)$ and update $\boldsymbol{\beta}$ via Eq. 4.2
**end while**
Pick the architecture candidate via Eq.4.4
```

4.2 SPHYNX: Search phase

Finding cells. Having defined the search space, we now discover *normal/reduce* cells using the DARTS micro-cell search algorithm; for complete details refer Appendix A. The choice of DARTS here is entirely for convenience, and SPHYNX can be used in conjunction with any other search method such as ENAS [9], GDAS [13], PC-DARTS [11], and GAEA [15].

Choosing initial number of channels and network depth. Once we have discovered *normal/reduce* cells, we need to determine the initial channels (C) and depths (D) of the network to satisfy the ReLU budget constraint. Intuitively, the total number of ReLU operations scales with both C and D. Furthermore, due to the ReLU balancing rule, the ReLU counts on each *post-processing* layer are equally distributed. Therefore, the total number of ReLU in the network is simply $H_0 \times W_0 \times C \times D$, where H_0 and W_0 are height and width of initial feature map, respectively. We empirically observed that the network performance deviates very little from C and D selections given a ReLU budget. We support our observation with results from ablation experiments conducted with various C and D selections given 50K ReLU budget in Appendix F.

Choosing the location of reduce cells. Conventional cell-based NAS approaches, including DARTS, fix the position of *reduce* cells at $D/3$ and $2D/3$ cell-depth index. In contrast, we propose a method to find the optimal position of *reduce* cells to improve network performance[2].

We focus on the case with two *reduce* cells; extending this to multiple cells is straightforward. Given a network with D cells, let $\boldsymbol{\beta} \in \mathbb{R}^K$ be a position indicator vector where $K = \binom{D}{2}$ is number of all possible candidates of *reduce* cell locations, and let $\hat{\boldsymbol{\beta}} = \text{softmax}(\boldsymbol{\beta}) = \frac{\exp \beta_i}{\sum_k \exp \beta_k}$ be a probability distribution over K elements.

Define a categorical random variable with distribution $\hat{\boldsymbol{\beta}}$ and encoded by random one-hot vectors $\mathbf{g} \in \{0,1\}^K$. We construct a "super" network as shown in Figure 3. Let f_i, where $i \in \{1, 2, \ldots, K\}$ be a function parameterized by weights $\mathbf{w}_i$; we can imagine f_i to represent candidate networks with different locations of *reduce* cells. The output F computes the linear combination $F = g_1 f_1 + \ldots + g_k f_k$. Intuitively, F samples one branch according to $\mathbf{g}$.

One can imagine learning the optimal indicator vector $\boldsymbol{\beta}$ via gradient descent; unfortunately, the sampling operation is not differentiable. Therefore, we leverage the Gumbel-softmax trick [25]. During the forward pass, we sample a

[2] In practice, due to channel scaling effects we find that re-locating *reduce* cells increases overall parameter count, but again this is not a bottleneck for private inference since we are focused on optimizing for ReLUs.

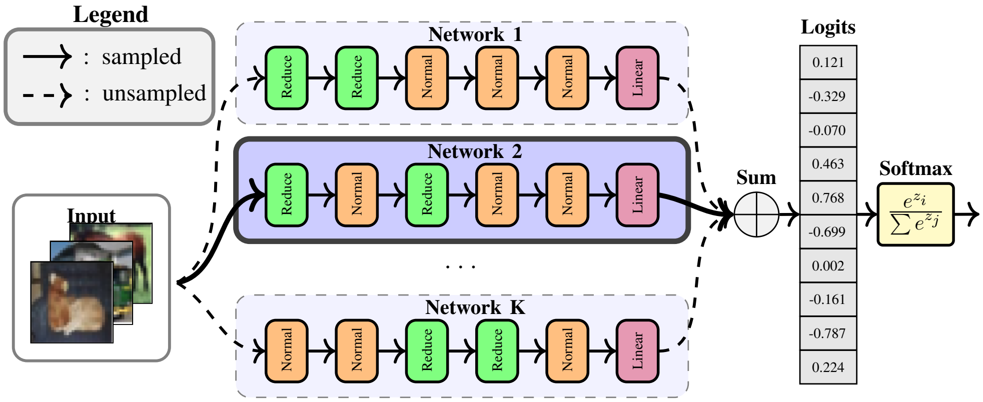


Figure 3: *The Gumbel-softmax trick applied to searching cell location. We randomly sample a candidate network from categorical variable $\boldsymbol{\beta}$ and only train sampled network for a given batch. In this example, our algorithm samples Network 2 and update parameters in sampled network and categorical variable $\boldsymbol{\beta}$.*

one-hot vector according to the formula:

$$\mathbf{g} = \text{one-hot}(\operatorname*{arg\,max}_{i\in\{1,2,\ldots,K\}} G_i + \log(\hat{\beta}_i)) \tag{4.1}$$

where $G_i \sim \text{Gumbel}(0,1)$ i.i.d samples drawn from the standard Gumbel distribution. During the backward pass, we use the straight-through Gumbel softmax estimator which replaces $\mathbf{g}$ with $\tilde{\mathbf{g}}$ during the gradient update:

$$\tilde{g}_i = \frac{\exp{(\log g_i + G_i)/\tau}}{\sum_k (\exp{(\log g_k + G_k)})/\tau} \tag{4.2}$$

where τ is a temperature parameter. The parameter τ controls the sharpness of the softmax approximation; $\tilde{\mathbf{g}} = \mathbf{g}$ as $\tau \to 0$, whereas $\tilde{\mathbf{g}}$ becomes an uniform distribution as $\tau \to \infty$.

Equipped with a fully differentiable technique for learning the parameter $\boldsymbol{\beta}$, we now train both the network parameters $\mathbf{w}$ and categorical parameter $\boldsymbol{\beta}$. Given a train (T) and validation (V) dataset of labeled pairs (x, y) drawn from a joint distribution (X, Y), Algorithm 1 update $\mathbf{w}$ and $\boldsymbol{\beta}$ in alternative fashion to estimate following bilevel optimization problem:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^K} \sum_{(x,y)\in V} \mathcal{L}(F_{\mathbf{w}^*,\boldsymbol{\beta}}(x), y) \quad \text{s.t.} \quad \mathbf{w}^* = \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \sum_{(x,y)\in T} \mathcal{L}(F_{\mathbf{w},\boldsymbol{\beta}}(x), y) \tag{4.3}$$

Once the training terminates, we select the positions of the *reduce* cells in the final network looking at the peak achieved by the categorical distribution (without any Gumbel sampling):

$$\mathbf{g} = \text{one-hot}(\operatorname*{arg\,max}_{i\in\{1,2,\ldots K\}} \log(\hat{\beta}_i)) \tag{4.4}$$

5 Evaluations

We now present a series of evaluation studies showing (i) superior performance of Sphynx with state-of-the-art methods for private inference, (ii) transferability of Sphynx cells to more complex datasets (Tiny-ImageNet and ImageNet), (iii) thorough ablation studies that describe the impact of each component in Sphynx.

5.1 Experimental setup

Sphynx adopts the Delphi PI protocol described in Section 2. Our reported online runtime includes total computation and communication costs from both the client and server. We evaluate Sphynx on CIFAR-100 [16], Tiny-ImageNet [17], and ImageNet [18]. Datasets are preprocessed only with image normalization, random horizontal flips, padding, and random crops for the fair comparison. We use NVIDIA Quadro RTX8000 for network training and an Intel i9-10900X CPU running at 3.70GHz with 64GB of memory for benchmarking PI runtime. We include details on the exact hyperparameter setup for searching cell structure and location in Appendix D.

5.2 Experimental results and ablation studies

The Sphynx search space is very efficient in terms of ReLUs. We compare the network performance between our Sphynx space with (a variant of) the regular DARTS search space given similar ReLU budgets. To save ReLUs for both methods, we use a technique called *ReLU sharing* inspired from the ReLU shuffling method in [4]. If a node connects to two or more convolution modules, convolution modules share a ReLU pre-computed feature map from the input. This reduce the ReLU operations without changing the functionality. We defer a visual explanation of ReLU sharing in Figure 9 from Appendix E.1. With all this, the micro-search space still faces the limitation of scaling down to a deficient ReLU budget network. The smallest network we could design using the micro-search space involved 78K ReLU counts with channel count C=1 and depth D=4. Furthermore, SPHYNX achieves 69.57% test accuracy with 50K ReLUs (C=5, D=10) while the network discovered using the micro-search space only achieves 47.24% test accuracy with 78K ReLUs. We include more detailed comparisons for various ReLU budgets in Appendix E.1

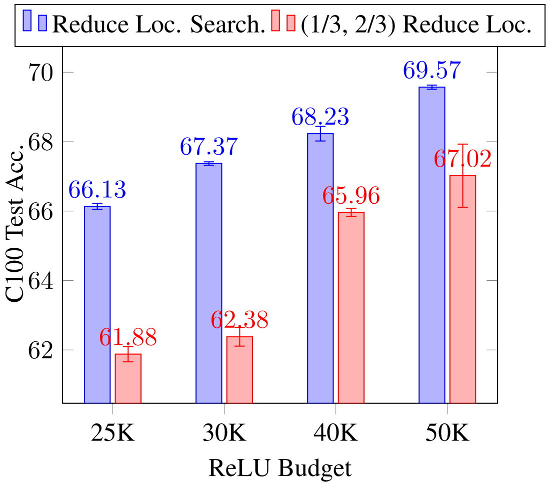


Figure 4: *Networks with Algorithm 1 enjoys the performance boost over the conventional* $D/3$ *and* $2D/3$ *reduce cells approaches over all range of ReLU budget. Each plot includes mean and standard deviation from three different random seed.*

Optimizing locations of *reduce* cells is necessary. We first compare between *reduce* cell location optimized network and $D/3$ and $2D/3$ fixed networks. We collect four different ReLU budget networks with three different random seeds: 25K, 30K, 40K, and 50K ReLU. We observe the network performance, especially at lower ReLU budgets, enjoys the benefit of Algorithm 1 with huge gains in test accuracy. The 25K ReLU network achieves 66.13% test accuracy from Algorithm 1 while fixing locations to $D/3$ and $2D/3$ only gives 61.88%. As shown in Figure 4, the networks uniformly enjoy boosts in accuracy via Algorithm 1 in various ReLU budget networks.

The natural follow-up question is whether Algorithm 1 is capable of finding optimal locations of *reduce* cells. We conduct experiments on CIFAR-100 with three different ReLU budgets: 25K, 30K, and 40K, where initial channels fixed at C=5 and depths chosen as D=5, 6, and 8, respectively. We directly compare the output $\hat{\boldsymbol{\beta}}$ from Algorithm 1 with test accuracy from all possible candidates through a grid search. Our empirical results show that our approach capable of finding the best *reduce* cell locations, which maximizes the test accuracy. We note that our methods can save computational resources over the grid search by a factor of $\frac{\text{\# of candidates}}{2}$ under the assumption that Algorithm 1 and final network training spends equivalent computational budgets (location searching + final network training). We note that we use the same number of epochs in Algorithm 1 for the final network training.

SPHYNX achieves the accuracy-latency Pareto frontier for CIFAR-100. Using SPHYNX, we design a range of ReLU-budgeted networks to understand the impact of accuracy-latency tradeoffs in PI. We plot Pareto curves in Figure 6 (SPHYNX in blue) comparing with prior PI methods: CryptoNAS [4], DELPHI [3], DeepReDuce [6], and SAFENet [7]. SPHYNX strictly outperforms CryptoNAS, DELPHI, and SAFENet for the entire range of latency models. SPHYNX achieves 66.13% test accuracy with 2.3× faster online latency than CryptoNAS models with 63.60%, and 1.7× faster than DELPHI with 66.00%. We approximate the online latency from SAFENet (67.50% test acc.) due to different specs in hardware. We leverage the our latency (4.4s) from DELPHI model with 67.00% and the Delphi latency (14.4s) with 67.3% model from SAFENet and estimate SAFENet online latency to be 2.2s with our hardware setup. Moreover, SPHYNX achieves the par test accuracy to SAFENet with 2.6× faster PI latency.

SPHYNX achieves better results over DeepReDuce in all ranges of online runtime, except at very low-ReLU budgets (24.6K). Our smallest network (25.6K ReLU) is on par with DeepReduce in test accuracy but 25%× slower in online runtime. This is probably due to the fact that SPHYNX contains more linear layers than 24.6K DeepReDuce network

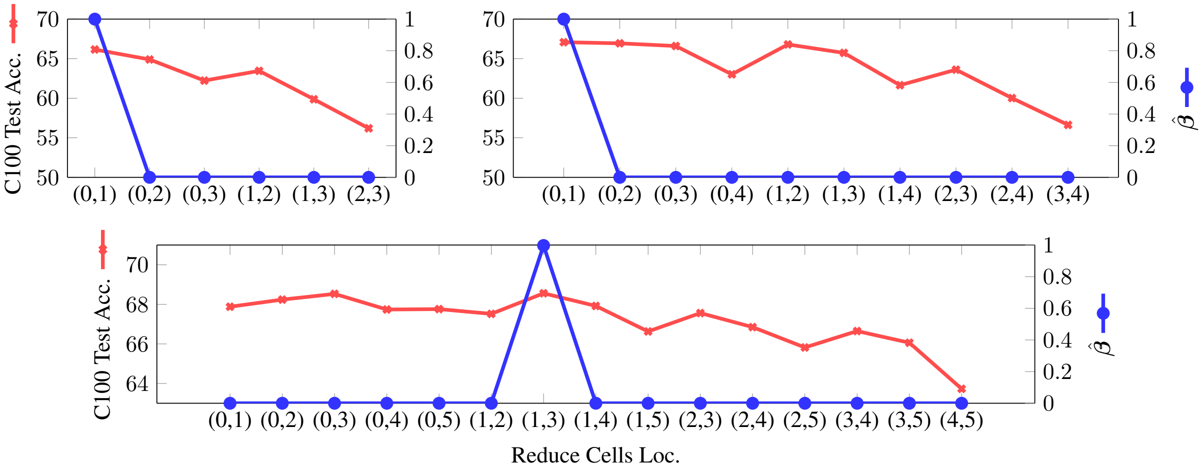


Figure 5: Comparison on learned $\hat{\boldsymbol{\beta}}$ and a grid search on all possible locations of reduce cells' test accuracy. The x-axis represents the locations of two reduce cells. Left and right y-axis represents the CIFAR-100 test accuracy and $\hat{\boldsymbol{\beta}}$ respectively. Categorical parameter $\boldsymbol{\beta}$ matches to the best performing network candidate from the grid search. *Upper-Left*: 25K ReLU budget network with depth 5. *Upper-Right*: 30K ReLU budget network with depth 6. *Bottom*: 40K ReLU budget network with depth 8.

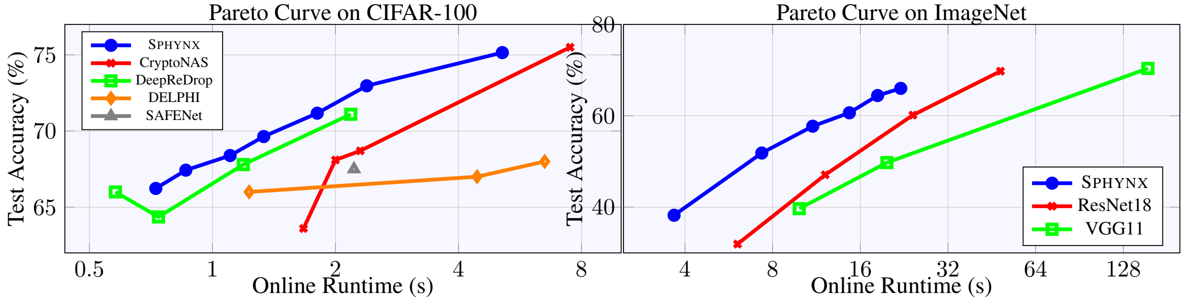


Figure 6: *Left:* Pareto curve on SPHYNX compare to state-of-the-art PI methods on CIFAR-100. *Right:* Pareto curve on SPHYNX compare to VGG11 and ResNet18 via DELPHI protocol. Online runtimes include both client and server online cost.

SPHYNX	**Tiny-ImageNet** (ResNet18 (2228K), Acc=61.28%)				
ReLU #	102.4K	204.8K	286.7K	491.5K	614.4K
PI Lat.	2.35s	4.40s	6.14s	10.2s	12.5s
Test Acc.	48.44%	53.51%	56.72%	59.18%	60.76%

Table 2: SPHYNX *Tiny-ImageNet results via transfer learning from cells found from CIFAR-100.* SPHYNX *with 614K ReLU achieves roughly par test accuracy to ResNet18 with 2228K ReLU counts.*

resulting a viable difference on PI latency, especially in an extreme low-ReLU regime. Apart from this, SPHYNX dominates as a Pareto frontier on all range of DeepReDuce case[3].

Transfer learning from CIFAR-100 cells to Tiny-ImageNet and ImageNet Next, we show that the SPHYNX cells learned from CIFAR-100 transfer to more complicated datasets such as Tiny-ImageNet and ImageNet. In case of Tiny-ImageNet, we found that an apples-to-apples comparison with DeepReDuce is difficult since only KD results were reported. In particular, the teacher model in DeepReDuce (ResNet18 with 2228K ReLUs) achieves 61.28% test accuracy while the student model surpasses it (e.g., 917K ReLU student network achieves 64.66% w/ KD). Under the assumption that the student models without KD are less potent than the teacher model, SPHYNX with 614K ReLUs achieves competitive test accuracy 60.57%, roughly par with ResNet18 with 2224K ReLUs, with over 0.25× fewer ReLUs.

[3] We note that our comparison here includes results without knowledge distillation (KD). The original DeepReduce paper empirically showed performance boosts in test accuracy using KD with a high-ReLU network as teacher. For brevity, we defer additional SPHYNX results with KD in Appendix D and have observed similar trends to the Pareto curve in Figure 6 between SPHYNX and DeepReDuce.

We also transfer SPHYNX CIFAR-100 cells to ImageNet to verify the promise of SPHYNX for large-scale complex datasets. We adopt a slightly different network architecture for ImageNet following DARTS; details are in Appendix H. Since the DELPHI protocol fails in the large-scale dataset, we estimate the PI latency by summing average GC latency per ReLU from our Tiny-ImageNet models, and plaintext inference time as an approximation to linear layer computation. SPHYNX strictly dominates VGG11 and ResNet18 for the entire range of latency values.

Additional ablation studies. We further support our approach by providing several other ablation studies. Due to space constraints, we only summarize these results and defer details to Appendix F. (i) The network performance deviates minuscule from various selections in initial channels and depths given a ReLU budget. (ii) ReLU balancing outperforms FLOP balancing. (iii) Cells with $N{=}7$ (more linear operations) found from SPHYNX achieve superior performance over cells with $N{=}5$ (less linear operations). (iv) Using cells reported from DARTS/PC-DARTS but modify as described in Section 4.1 is significantly worse compare to using cells found from the SPHYNX framework.

6 Related Work and Discussion

CryptoNets [26] is one of the earliest works on PI, fully leveraging homomorphic encryption (FHE) to guarantee data (but not model) privacy. However, CryptoNets only allows polynomial activations due to reliance on fully homomorphic encryption. Subsequent works, including MiniONN [5], SecureML [27], Gazelle [2], and DELPHI [3], have focused on providing both data and model privacy and support standard nonlinear activation functions such as ReLUs. These approaches isolate linear and non-linear operations and apply different protocols to each. Several state-of-the-art PI protocols leverage expensive Garbled Circuits for ReLU operations. Consequently, subsequent works have concentrated on reducing ReLU operations, such as ReLU approximation [3, 7], ReLU-efficient network design via NAS [4], and pruning ReLU layers [6]. A separate line of PI literature, including DeepSecure [28] and XONN [29], leverages binarized neural networks but these approaches underperform in test accuracy than conventional networks.

Early NAS approaches leveraged RL-based controllers [8] to design entire convolutional networks. Macro-search methods adopt a simple chain-structure skeleton with skip-connections between layers since the search complexity tends to exponentially increase depending on the number of design components. NASNet [14] proposes transforming macro-search tasks into finding block modules, which can be repeatedly stacked up to form the final architecture. NASNet further improves the performance over [8]; however, both approaches consumed substantial computational resources, running into thousands of GPU-days. Subsequent NAS works on micro-search space have focused on reducing the search time with techniques such as gradient-based optimization [30, 10, 12, 11, 13], weight-sharing super-network [9, 31, 23], evolutionary algorithms [32, 33], and hyperparameter optimization techniques [34, 23]. We envision that SPHYNX extends to any of these methods.

Numerous follow-up avenues still persist. SPHYNX finds the network in three stages that are somewhat decoupled: (i) find cells; (ii) choose the initial channels and depth given a ReLU budget, and (iii) optimize for the location of the reduce cells. Integrating these three phases into a single technique might improve final performance by reducing the gap between the search and evaluation phase [35]. Moreover, the objective function in SPHYNX is the standard empirical risk used in training. The impact of incorporating additional terms (such as ReLU counts) to the objective function will be an interesting direction for future work.

References

[1] Yang Li, Zhenhua Han, Quanlu Zhang, Zhenhua Li, and Haisheng Tan. Automating cloud deployment for deep learning inference of real-time online services. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1668–1677. IEEE, 2020.

[2] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[3] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522. USENIX Association, Aug. 2020.

[4] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16961–16971. Curran Associates, Inc., 2020.

[5] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications*

Security, pages 619–631, 2017.

[6] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. DeepReDuce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, 2021.

[7] Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. Safenet: Asecure, accurate and fast neu-ral network inference. *Proc. Int. Conf. Learning Representations*, 2021.

[8] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *Proc. Int. Conf. Learning Representations*, 2017.

[9] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *Proc. Int. Conf. Machine Learning*, 2018.

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *Proc. Int. Conf. Machine Learning*, 2018.

[11] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[12] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[13] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019.

[14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *IEEE Conf. Comp. Vision and Pattern Recog*, 2018.

[15] Liam Li, Mikhail Khodak, Nina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *International Conference on Learning Representations*, 2021.

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

[17] Ya Le and X. Yang. Tiny imagenet visual recognition challenge. 2015.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[20] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[21] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[22] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[23] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[24] Minsu Cho, Mohammadreza Soltani, and Chinmay Hegde. One-shot neural architecture search via compressive sensing. *arXiv preprint arXiv:1906.02869*, 2019.

[25] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017.

[26] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.

[27] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[28] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[29] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. {XONN}: Xnor-based oblivious deep neural network inference. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1501–1518, 2019.

[30] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *Proc. Int. Conf. Learning Representations*, 2019.

[31] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. *Proc. Int. Conf. Machine Learning*, 2018.

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *Proc. Assoc. Adv. Art. Intell. (AAAI)*, 2019.

[33] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

[34] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[35] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.

[36] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

A Background on DARTS: Differentiable Architecture Search

This section describes the micro-search NAS optimization problem, and briefly discusses DARTS details.

In the micro-search space, the neural architecture has a skeleton of repeatedly stacked structures or modules denoted as cells. Following the pioneer works from NASNet [14], searching the architecture candidate is commensurate to search two types of cells: normal cells and reduce cells. Normal cells learn the the high-level features returning the exact spatial resolution to the input. Conversely, reduce cells extract the high-level representations and reduce the spatial resolution (commonly reduced by half on height and width). Since the final architecture is defined as stacked normal and reduce cells, constructing the architecture is equivalent to selecting normal/reduce cells in the micro-search space based NAS perspective.

In detail, consider a supervised learning NAS setup. We have a train (T) and validation (V) dataset of labeled pairs (x, y) drawn from a joint distribution (X, Y). Let $f_{\mathbf{w},a}$ be the candidate network parameterized by $\mathbf{w}$ given a network constructed with normal/reduce cells from discrete architecture space $a \in A$. Finding the normal/reduce cells $a \in A$ is equivalent to solving following combinatorial optimization problem:

$$\min_{a\in A} \sum_{(x,y)\in V} \mathcal{L}(f_{\mathbf{w}^*,a}(x), y) \quad \text{s.t.} \quad \mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{(x,y)\in T} \mathcal{L}(f_{\mathbf{w},a}(x), y) \tag{A.1}$$

Gradient-based NAS approaches including DARTS require continuous relaxation step of discrete architecture space. Let Θ be some continuous relaxation of the discrete architecture space A. Then the gradient-based approaches solve the following objectives:

$$\min_{\theta\in\Theta} \sum_{(x,y)\in V} \mathcal{L}(f_{\mathbf{w}^*,\theta}(x), y) \quad \text{s.t.} \quad \mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{(x,y)\in T} \mathcal{L}(f_{\mathbf{w},\theta}(x), y) \tag{A.2}$$

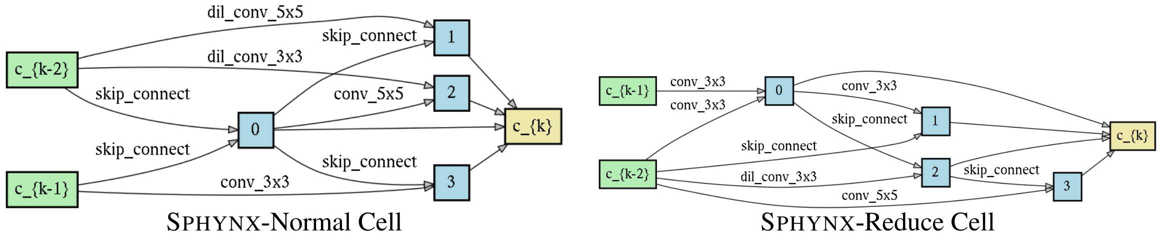
and derive a final architecture by mapping back to discrete space: $\texttt{Sample}:\Theta \to A$ (e.g., magnitude based selection).

During the search phase, DARTS involves a continuous relaxation of the discrete cell space via weighted summation of all possible operation between node i and node j by leveraging softmax. From an operation set $\mathcal{O}$, the continuous relaxed operation $\bar{o}$ is expressed with learnable architecture parameter θ:

$$\bar{o}^{(i,j)}(z^{(i)}) = \sum_{o\in\mathcal{O}} \frac{\exp(\theta_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\theta_{o'}^{(i,j)})} o(z^{(i)}) \tag{A.3}$$

where $\boldsymbol{\theta} = \{\theta^{(i,j)}\}$ and $\theta^{(i,j)} \in \mathbb{R}^{|\mathcal{O}|}$. During the search phase, DARTS alternatively update the network parameter weights $\mathbf{w}$ and architecture parameter $\boldsymbol{\theta}$ via gradient descent algorithms. Finally, DARTS selects the final normal/reduce cells with a function `Sample` selecting two strongest predecessors for each intermediate node from architecture parameter $\boldsymbol{\theta}$ by $\max_{o\in\mathcal{O}} \frac{\exp(\theta_o^{(i,j)})}{\sum_{o'\in\mathcal{O},o\neq zero} \exp(\theta_{o'}^{(i,j)})}$.

B Found cells from SPHYNX



SPHYNX-Normal Cell SPHYNX-Reduce Cell

Figure 7: *Normal* and *reduce* cells found by SPHYNX from CIFAR-100.

C CIFAR-100, Tiny-ImageNet, and ImageNet results in Table

Table 3: Comparison of SPHYNX and existing state-of-the-art methods in private inference on CIFAR-100. Higher the test accuracy and lower the PI latency the better. Our empirical results outperforms existing PI methods in various ReLU budget networks on test accuracy and private inference latency. We run SPHYNX with three different random seeds and report mean and standard deviation.

	Methods	**ReLUs**	**Test Acc.**	**PI Lat.**		**Methods**	**ReLUs**	**Test Acc.**	**PI Lat.**
ReLU $\leq 55K$	SPHYNX	25.6K	**66.13±0.09%**	727ms	ReLU $\leq 350K$	SPHYNX	71.7K	**71.06±0.15%**	**1805**ms
	DeepReDuce	24.6K	66.00%	**579**ms		CryptoNAS	86.0K	68.13%	2000ms
	SPHYNX	30.2K	67.37±0.05%	861ms		SPHYNX	102.4K	**72.90±0.06%**	**2385**ms
	SPHYNX	41.0K	**68.23±0.21%**	**1105**ms		CryptoNAS	100.0K	68.30%	2300ms
	SPHYNX	51.2K	**69.57±0.06%**	1335ms		DELPHI	180.0K	67.00%	4440ms
	DeepReDuce	49.2K	67.80%	1190ms		SPHYNX	230.0K	74.93 ± 0.16%	**5120**ms
	DELPHI	50.0K	66.00%	1230ms		DELPHI	300.0K	68.00%	6500ms
	CryptoNAS	50.0K	63.60%	1670ms		CryptoNAS	344.0K	75.64%	7500ms

Table 4: **Tiny-ImageNet and ImageNet results**. No prior art on PI experiment on Tiny-ImageNet and ImageNet except DeepReDuce [6] only on Tiny-ImageNet. We note that apple-to-apple comparison between SPHYNX and DeepReDuce is challenging due to KD incorporated in DeepReDuce training. SPHYNX strictly outperforms scaled down VGG11 and ResNet18 with less ReLU budget. Our ImageNet training methods on SPHYNX, VGG11, and ResNet18 are equivalent as described in Appendix D.

	Methods	**ReLUs**	**Test Acc.**	**PI Lat.**		**Methods**	**ReLUs**	**Test Acc.**	**PI Lat.**
Tiny-ImageNet	SPHYNX	102.4K	48.44%	2350ms	**ImageNet**	ResNet18	288.4K	31.93%	6060ms
	SPHYNX	204.8K	53.51%	4401ms		SPHYNX	345K	**51.85**%	7340ms
	SPHYNX	286.7K	56.72%	6140ms		VGG11	470K	39.68%	9870ms
	SPHYNX	491.5K	59.12%	10205ms		SPHYNX	517K	**57.72**%	10980ms
	SPHYNX	614.4K	**60.76**%	12548ms		ResNet18	577K	47.09%	12120ms
	ResNet18	2228.2K	61.28%	N/A		SPHYNX	862K	**64.43**%	18340ms
	-	-	-	-		VGG11	940K	49.76%	19740ms
	-	-	-	-		SPHYNX	1034K	**66.00**%	22020ms
	-	-	-	-		ResNet18	1154K	60.15%	24225ms

D Search/Evaluation Protocols and Hyperparameter Setups

We provide the search/evaluation training protocol to reproduce the results. This section includes the following hyperparameters setup:

- Table 5: Hyperparameter setup to find the *normal/reduce* cells from SPHYNX search space.
- Table 6: Hyperparameter setup to find the location of reduce cells.
- Table 7: Hyperparameter setup to train the final network on CIFAR-100 and Tiny-ImageNet.
- Table 8: Hyperparameter setup to train the final network on ImageNet.

Table 5: *Normal/Reduce* cells searching hyperparameters. $\mathbf{w}$ and $\boldsymbol{\theta}$ stand for network and architecture parameter, respectively.

$\mathbf{w}$ optimizer	SGD	initial LR	0.025
Nesterov	Yes	ending LR	0.001
momentum	0.9	LR schedule	cosine
$\mathbf{w}$ weight decay	0.0003	epoch	50
batch size	64	initial channel	5
cells #	8	cutout	No
ops #	6	nodes #	7
random flip	p=0.5	random crop	Yes
normalization	Yes	grad clip	5.0
$\boldsymbol{\theta}$ Optim.	Adam	$\boldsymbol{\theta}$ init. LR	0.0003
$\boldsymbol{\theta}$ weight decay	0.001	reduce loc.	$[D/3, 2D/3]$

Table 6: *Reduce* cells location search hyperparameters. $\mathbf{w}$ and $\boldsymbol{\beta}$ stand for network and categorical parameter, respectively.

$\mathbf{w}$ optimizer	SGD	initial LR	0.025
Nesterov	Yes	ending LR	0.001
momentum	0.9	LR schedule	cosine
$\mathbf{w}$ weight decay	0.0003	epoch	600
batch size	64	initial τ	1000
ending τ	0.1	τ schedule	Linear
random flip	p=0.5	random crop	Yes
normalization	Yes	grad clip	5.0
$\boldsymbol{\beta}$ Optim.	Adam	$\boldsymbol{\beta}$ init. LR	0.0003
$\boldsymbol{\beta}$ weight decay	0.001	-	-

Table 7: Final network training hyperparameter protocol on CIFAR-100 and Tiny-ImageNet. D stands for the total number of cells in the final network.

optimizer	SGD	initial LR	0.025
Nesterov	Yes	ending LR	0
momentum	0.9	LR schedule	cosine
weight decay	0.0003	epoch (C100/Tiny)	600/250
batch size	96	parallel training	No
random flip	p=0.5	random crop	Yes
normalization	Yes	cutout	No
drop-path prob	0.2	aux loss loc.	$2D/3$
grad clip	5.0	aux weight	0.4

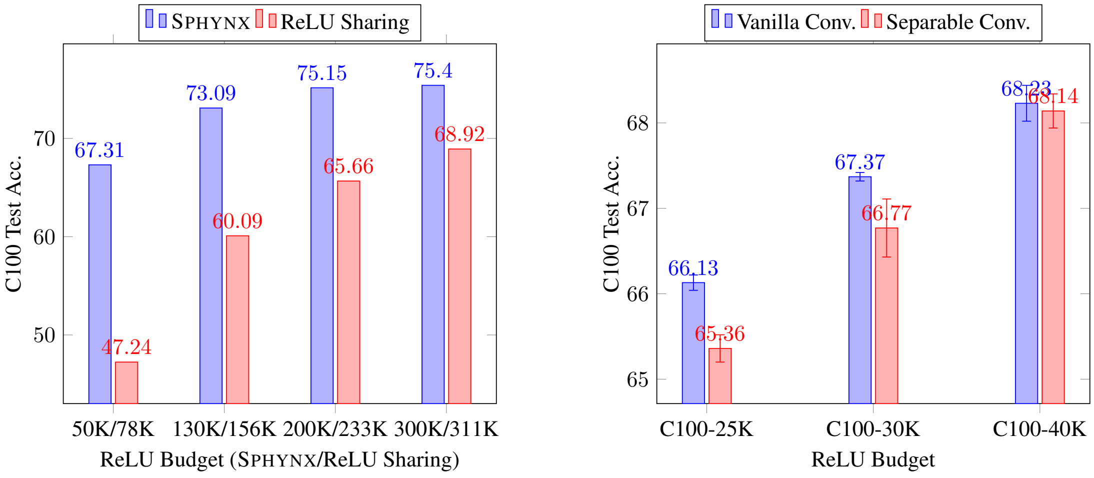
E Detail descriptions in Ablation studies

E.1 SPHYNX vs. Micro-Search Space in ReLU perspective

We provide the more comparison between the networks from SPHYNX search space and micro-search space evaluating in ReLU perspective. We introduced a ReLU saving technique denoted as ReLU sharing motivated from ReLU shuffling

Table 8: Final Network Training Hyperparameter Protocol on ImageNet. D stands for the total number of cells in the final network.

optimizer	SGD	initial LR	0.1
Nesterov	Yes	Lr schedule	step
momentum	0.9	lr decay	30, 60, 90
lr decay mult. factor	0.1	epoch	120
weight decay	0.0003	# of GPUs	4
batch size	768	parallel training	Yes
random flip	p=0.5	random crop	Yes
normalization	Yes	cutout	No
drop-path prob	0.0	aux loss loc.	$2D/3$
label smoothing	No	aux weight	0.4
grad clip	5.0	-	-



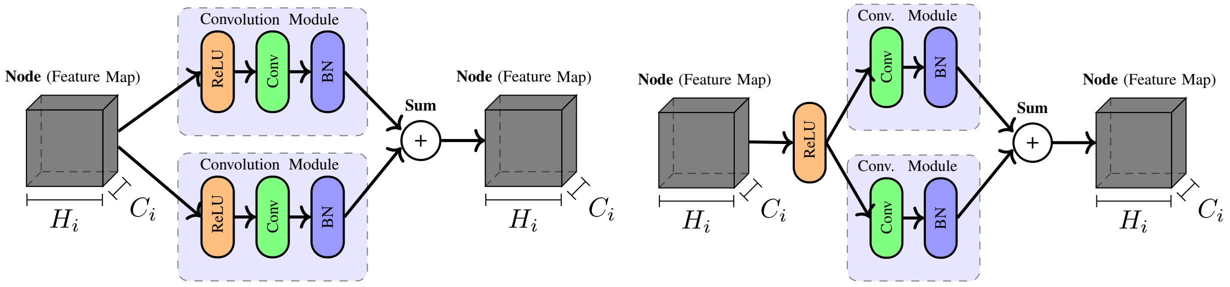
(a) SPHYNX vs. micro-search space

(b) Vanilla Conv vs. Separable Conv

Figure 8: Ablation studies: (a) Compares the difference in performance between SPHYNX and micro-search space. We leverage the ReLU sharing technique inspired from [4] on the micro-search space to reduce the ReLU operations without further functionality changes. Furthermore, we remove ReLU layers in the preprocessing layers (which each layer ReLU cost is equivalent to $4{\times}H_i{\times}W_i$). The experiment results show that SPHYNX superior to modified micro-search space with above techniques. (b) Compares the operation candidates between vanilla convolution and separable convolution. C100 and Image stand for CIFAR-100, respectively. Vanilla convolution achieves a high test accuracy than separable convolutions given the same network with equivalent *normal*/*reduce* cells except the operation set.

in [4]. ReLU sharing reduces the number of ReLU operations without changing the functionality. If the input node (or intermediate nodes) connects to two or more convolution modules (ReLU-Conv-BN), ReLU sharing compute the ReLU of intput node once and shares ReLU computed feature map to convolution modules. ReLU sharing reduces ReLU operations more efficiently as more convolution modules attached to the node. However, ReLU sharing acts useless when every node has at most one convolution modules attached in the worst case. Figure 9 demonstrates the visual representation of ReLU sharing. If three convolution modules connected to the input node in Figure 9 (b), ReLU sharing saves ReLU cost by factor of three. To further reduce the ReLU budget in micro-search space, we remove ReLU layers in the preprocessing layers, 1×1 convolution modules with ReLU-Conv-BN. We apply ReLU balancing on both methods: ReLU balancing increases channel by factor of 4 when the spatial resolution is halved.

Figure 8 (a) shows additional comparison to SPHYNX space and micro-search space with ReLU sharing technique and ReLU pruning on preprocessing layers. The smallest ReLU budget network from the micro-search space has 78K ReLU counts with initial channel C=1 and D=4. In case of 154K, 233K, and 311K ReLU networks from micro-search space, we select C=2, C=3, and C=4, respectively while fixing the depth to D=4. The network skeleton from SPHYNX strictly outperforms all range of ReLU budget network to micro-search space based network skeleton, which supports the necessity of SPHYNX search space in the lens of ReLU counts.



(a) Default connection between the Nodes (b) Saving ReLU Operations using ReLU Sharing

Figure 9: *Visualization of saving ReLU counts via ReLU Sharing. Unlike* SPHYNX *space, applying ReLU Sharing technique to existing NAS search space cannot scale down the network to satisfy low-ReLU budget (e.g., < 70K). (a) Default connection between the nodes in micro-search space. The input feature map needs to go through two ReLU layers for each convolution module. (b) Pre-computing ReLU once and split to each convolution module saves the ReLU count without changing a functionality.*

F Additional ablation studies

F.1 Minuscule deviation in performance in selecting initial channels and depths given a ReLU budget

We empirically observe that the network performance deviates minuscule with the possible choice of initial channels and depth to ReLU constraints. We conduct the CIFAR-100 experiments building the networks with the ReLU budget close to 50K with different initial channels (depth changes accordingly). We train five various networks with various initial channels: $C = 5, 6, 7, 8, 10$. Our experiment excludes the $C = 9$ case due to no available setup for a network close to the 50K ReLU budget. We observe that the network performance from various initial channels achieves a small deviation from the mean with a test accuracy of 69.38 ± 0.23 with the best (resp. worst) accuracy of 69.68% (resp. 69.11%). Table 9 provides a detailed network setup including initial channel, depth, and ReLU counts.

Method	**Init Channel (C) and Depth (D)**				
	C=5, D=10	C=6, D=8	C=7, D=7	C=8, D=6	C=10, D=5
ReLU #	51.2K	49.2K	50.2K	49.2K	51.2K
Test Accuracy	69.64%	69.11%	69.68%	69.27%	69.22%

Table 9: *Detailed experiment setup for selecting channels and depth. Given various initial channels* $C = 5, 6, 7, 8, 10$, *we choose the depth such that the networks' ReLU count close to 50K as possible. In this case, the ReLU count formula is* $32 \times 32 \times C \times D$ *due to ReLU balancing technique.* 32 *comes from the original image size of CIFAR-100. We observe that the network performs consistently on various C and D selections with small standard deviations. (*69.38 ± 0.23*)*

F.2 ReLU Balancing vs. FLOP Balancing

We compare the performance between ReLU balancing and FLOP balancing. As a recall, FLOP balancing is the typical rule that many existing convolutional neural networks follow, which increase channel size by a factor of two when the spatial resolution reduces by half. We compare four different ReLU budget networks: 25K, 30K, 40K, and 50K. First, we leverage the same *normal/reduce* cells in Figure B to construct the network. Second, we match the equivalent depth for the FLOP balancing network and reduce cells' locations to the ReLU balancing network. Finally, we adjust initial channels on the FLOP balancing network to satisfy a given ReLU budget. As shown in Table 10 ReLU balancing applied networks outperforms the FLOP balancing networks in test accuracy. Furthermore, FLOP balanced networks struggle with inconsistency in the performance of the given ReLU budget. Table 11 includes the network configurations conducted in this experiment. Table 11 shows the experiment setup, including the network initial channels, depth, and reduce cell locations. We note that we carefully chose C for a ReLU budget while fixing D such that reducing cell cases takes the same advantage of reducing cell locations.

F.3 Vanilla Convolution vs. Separable Convolution

This section compares the performance difference between vanilla convolution (which we adopt in our search space) and separable depthwise convolution. We leverage the same cells from our methods and convert the vanilla convolutions with separable convolutions without ReLU layers. Our empirical results on CIFAR-100 in Figure 8 (b) show that vanilla convolution achieves a slightly higher test accuracy compare to the separable convolutions. Further, we test on a small network with 188K ReLU counts on ImageNet, in which we observe the more significant performance difference as

Method	**ReLU Budget** (ReLU Bal. Net. / FLOP Bal. Net.)			
	25.6K/26.1K	30.2K/30.4K	41.0K/41.4K	51.2K/53.7K
ReLU Bal. Test Acc.	**66.16**%	**67.31**%	**68.39**%	**69.64**%
FLOP Bal. Test Acc.	65.14%	66.16%	65.55%	69.32%

Table 10: *Comparison between the network based on ReLU balancing and FLOP balancing on CIFAR-100. Bal. stands for balancing. Applying ReLU balancing on a cell-based network achieves a higher test accuracy over FLOP balancing. Furthermore, ReLU balancing enjoys consistency on test accuracy improvement given the ReLU budget, while FLOP balancing does not. (e.g., FLOP balancing network on 41.4K ReLU budget)*

Method	**ReLU Budgets** (ReLU Balancing/FLOP Balancing)			
	25.6K/26.1K	30.2K/30.4K	41.0K/41.4K	51.2K/53.7K
ReLU Balancing	C=5, D=5	C=5, D=6	C=5, D=8	C=5, D=10
FLOP Balancing	C=17, D=5	C=17, D=6	C=12, D=8	C=14, D=10
Reduce Cell Loc.	(0, 1)	(0, 1)	(1, 3)	(0, 5)

Table 11: *Experiment setup for ReLU balancing and FLOP balancing comparison. We use equivalent cell structures in Figure 7 to construct the network. We use the same reduce cells location for both network. Note that the number of ReLU operations in FLOP balancing affects by* reduce *cells locations.*

38.11% and 35.95% with vanilla and separable convolution, respectively. Table 12 includes the details in formations, including test performance and model parameters. We observe that separable convolution-based networks struggle with small network parameter size, especially in the small-ReLU budget network.

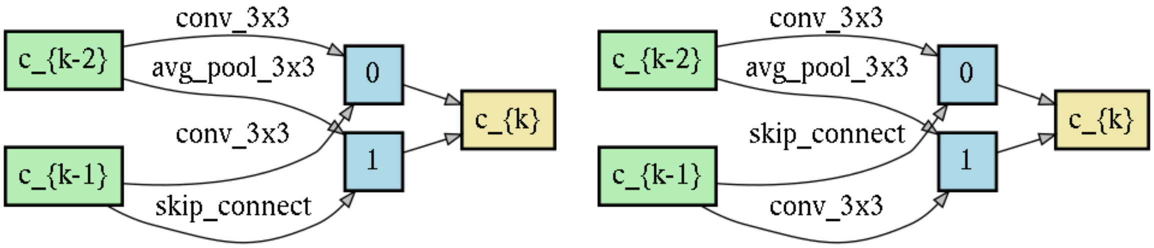
Method	**ReLU Budgets**		
	25.6K	30.2K	41.0K
Vanilla Conv.	$66.13 \pm 0.09\%$	$67.37 \pm 0.05\%$	$68.23 \pm 0.21\%$
Separable Conv.	$65.36 \pm 0.16\%$	$66.77 \pm 0.34\%$	$68.14 \pm 0.20\%$
Param. (Vanilla/Separable).	2.75M/0.35M	3.41M/0.43M	3.45M/0.45M

Table 12: *Comparison between the network with vanilla convolutions and separable convolutions.*

F.4 *Normal* and *Reduce* cells with smaller number of nodes

We recall our cell structures. We define the operation set, eliminating the ReLU layer in convolution modules into a sequence of Conv-BN from ReLU-Conv-BN. We also remove the ReLU layers in 1×1 preprocessing layers. In other words, only linear operations exist inside the cells. We ask whether to reduce the linear operations since the operations inside the cells are equivalent to linear combinations of linear functions.

We conduct the experiments by leveraging DARTs designing the cells $N = 5$. We first search the cells with DARTs with identical training protocols to Section. Figure 10 shows the cells we found with $N = 5$ nodes. Then, conducting apple-to-apple comparisons between $N = 7$ nodes and $N = 5$ nodes with identical training hyperparameters, we observe the superior performance in test accuracy with $N = 7$ nodes cells over $N = 5$ nodes.



Sphynx-$N = 5$ Normal Cell

Sphynx-$N = 5$ Reduce Cell

Figure 10: Smaller *Normal* and *reduce* cells found by Sphynx with $N = 5$ from CIFAR-100.

F.5 Recycling DARTS/PC-DARTS cells to Sphynx Search Space

What will happen if we skip the search phase and evaluate the final network applying existing NAS cells such as DARTS/PC-DARTS cells? We modify the cells following the rule in Section 4.1:

Method	ReLU Budgets			
	25K	30K	40K	50K
SPHYNX $N = 5$ Cells	64.24%	66.51%	67.92%	68.80%
SPHYNX $N = 7$ Cells	$\mathbf{66.13 \pm 0.09\%}$	$\mathbf{67.37 \pm 0.05\%}$	$\mathbf{68.23 \pm 0.21\%}$	$\mathbf{65.57 \pm 0.06\%}$

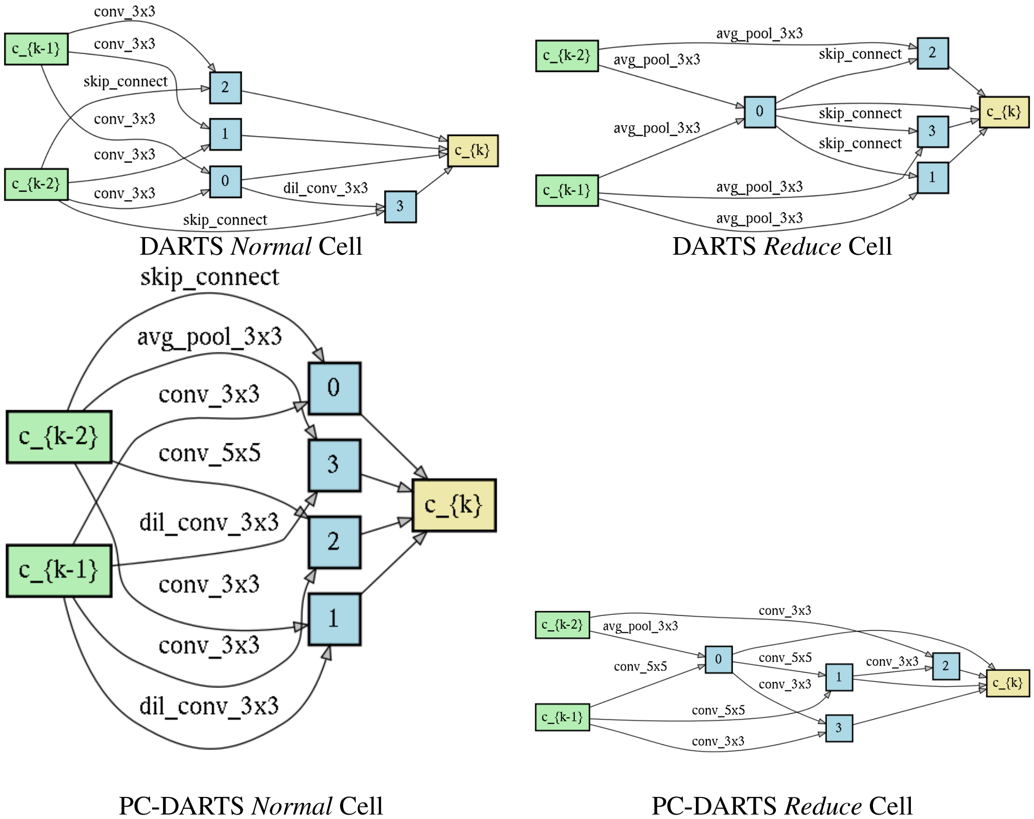
Table 13: *Test accuracy comparison between $N = 5$ cells (smaller cells) and $N = 7$ cells (original) found from* SPHYNX.

1. We replace separable convolution with regular convolution.
2. We change the convolution module from ReLU-Conv-BN to Conv-BN.
3. We replace max-pooling layers with average pooling layers.
4. We add the non-linear layers at the end of each cell.

These modification, compare to Appendix E.1, allows designing ReLU-efficient networks with existing NAS cells. Our ablation study shows that the cells found directly from the SPHYNX perform superior to modified DARTS and PC-DARTS cells searched from the original micro-search space as shown in Table 14.

Method	ReLU Budgets			
	25K	30K	40K	50K
DARTS Cell	59.86%	62.03%	66.92%	68.60%
PC-DARTS Cell	64.32%	66.24%	67.78%	69.07%
SPHYNX	$\mathbf{66.13 \pm 0.09\%}$	$\mathbf{67.37 \pm 0.05\%}$	$\mathbf{68.23 \pm 0.21\%}$	$\mathbf{65.57 \pm 0.06\%}$

Table 14: *The final architecture performance (equivalent reduce locations) comparison between the cells* SPHYNX *found to existing cells from DARTS and PC-DARTS. Separable convolutions, separable dilated convolution, and max-pooling replaced by vanilla convolution, vanilla dilated convolution, and average pooling, respectively.*



DARTS *Normal* Cell

DARTS *Reduce* Cell

PC-DARTS *Normal* Cell

PC-DARTS *Reduce* Cell

Figure 11: Converted DARTS and PC-DARTS *normal*/*reduce* cells for SPHYNX design space.

G Boosting Performance with Knowledge Distillation

This section includes a direct comparison to the DeepReDuce paper on the CIFAR-100 dataset with knowledge distillation. DeepReDuce empirically showed that leveraging KD with the pretrained teacher model with original network architecture and ReLU pruned student network boosts the test accuracy over regular training. We apply the KD methods from [36], which reformulates the objective function by adding the regression mean square error loss between teacher's and student's logit. Finally, we formally state the objective functions for the knowledge distillation.

Let f_s and f_t be the student and teacher network, respectively. Let f_s is parameterized by $\mathbf{w}$. Let g_s and g_t be the network logit corresponding from f_s and f_t, respectively. Given a train (T) and validation (V) dataset of labeled pairs (x, y) drawn from a joint distribution (X, Y). The KD objective functions is defined as following:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{(x,y) \in T} \mathcal{L}(f_s(\mathbf{w}, x), y) + \|g_s(\mathbf{w}, x) - g_t(x)\|_2^2 \tag{G.1}$$

This section includes a direct comparison to the DeepReDuce paper on the CIFAR-100 dataset with knowledge distillation. DeepReDuce empirically showed that leveraging KD with the pretrained teacher model with original network architecture and ReLU pruned student network boosts the test accuracy over regular training. We apply the KD methods from [36], which reformulates the objective function by adding the regression mean square error loss between teacher's and student's logit. Finally, we formally state the objective functions for the knowledge distillation.

Our teacher model achieves 76.20% test accuracy, and the network consists of equivalent cells we found from SPHYNX but replacing the convolution modules from Conv-BN to ReLU-Conv-BN. Table 15 shows the direct comparison of SPHYNX and DeepReDuce. Our results show that SPHYNX overall achieves the better test accuracy for the ReLU budget except for the DeepReDuce network with a 24.6K ReLU budget. In the PI latency perspective, DeepReDuce with 24.6K ReLU network outperforms SPHYNX 25.6K ReLU network by 117ms faster. While the online latency consists of computing the linear layer and non-linear layer from GC, the linear layer computation time may not be neglectable in an extremely low-ReLU budget network. We conjecture that the linear layer computation in SPHYNX is taking more than DeepReDuce due to the network size resulting in significant PI latency although small ReLU budget difference (24.6K vs. 25.6K).

Table 15: Knowledge Distillation Results on CIFAR-100. The lower the ReLUs and PI latency the better.

DeepReDuce			**SPHYNX**			**Improvement**		
ReLUs	Test Acc.	PI Lat.	ReLUs	Test Acc.	PI Lat.	ReLUs	Test Acc.	PI Lat.
24.6K	68.41%	579ms	25.6K	68.40%	727ms	1.04×	+0.16	+117ms
28.7K	68.70%	738ms	31.2K	70.21%	861ms	1.08×	+1.51	+123ms
49.2K	71.10%	1190ms	41.0K	71.57%	1114ms	0.83×	+0.47	-76ms
57.3K	72.68%	1350ms	51.2K	72.85%	1335ms	0.89×	+0.17	-15ms

H Architecture Details for ImageNet

We transfer the cells we learned from CIFAR-100 to a complex image classification dataset. Primarily, we construct a slightly modified network following from DARTS. We start the network with three convolution stem layers with a stride of 2, reducing the input image resolution from 224×224 to 28 ×28. We note that the first stem convolution is Conv-BN's sequence and increases the channel dimension from 3 to $C/2$. The second and third stem convolution module has ReLU-Conv-BN sequence. The second stem convolution increase the channel dimensions from $C/2$ and C, and the third stem convolution maintains the channel dimensions but only reduces the feature map size by half. Then the cells are stacked, taking second and third stem convolutions as inputs.

I Details for Final Networks from SPHYNX for Different ReLU Budgets

As described in Section 4.2, the number of initial channels and network depth determine the total ReLU count. We provide these two hyperparameters we used for various ReLU budgets on CIFAR-100, Tiny-ImageNet, and ImageNet in Table 16. Also, we offer the VGG11 and ResNet18 network specifications for various ReLU budgets we used as the comparison. For both VGG11 and ResNet18, we only control the initial number of channels to satisfy the desired ReLU budget.

Table 16: Detailed network configurations at various ReLU budgets for SPHYNX, VGG11, and ResNet18.

Dataset	**Init. Ch.**	**Network Depth**	**ReLUs**	**Reduce Cells Loc.**
CIFAR-100	5	5	25.6K	$(0, 1)$
CIFAR-100	5	6	30.2K	$(0, 1)$
CIFAR-100	5	8	41.0K	$(1, 3)$
CIFAR-100	5	10	51.2K	$(0, 5)$
CIFAR-100	7	10	71.7K	$(0, 5)$
CIFAR-100	10	10	102.4K	$(0, 5)$
CIFAR-100	15	15	230.0K	$(2, 6)$
Tiny-ImageNet	5	5	102.4K	$(0, 1)$
Tiny-ImageNet	5	10	204.8K	$(0, 5)$
Tiny-ImageNet	7	10	286.7K	$(0, 5)$
Tiny-ImageNet	20	10	819.2K	$(0, 5)$
ImageNet	10	10	172K	$(1, 5)$
ImageNet	20	10	345K	$(1, 5)$
ImageNet	30	10	517K	$(1, 5)$
ImageNet	40	10	690K	$(1, 5)$
ImageNet	50	10	862K	$(1, 5)$
ImageNet	60	10	1034K	$(1, 5)$
ImageNet (VGG11)	4	11	472K	N/A
ImageNet (VGG11)	8	11	936K	N/A
ImageNet (VGG11)	64	11	7488K	N/A
ImageNet (ResNet18)	8	18	288K	N/A
ImageNet (ResNet18)	16	18	577K	N/A
ImageNet (ResNet18)	32	18	1154K	N/A
ImageNet (ResNet18)	64	18	2308K	N/A