COLIBRI: Optimizing Multi-Party Secure Neural Network Inference Time for Transformers*

Daphnee Chabal[1], Tim Muller[1], Eloise Zhang[1], Dolly Sapra[1], Cees de Laat[1], and Zoltán Ádám Mann[2]

[1] University of Amsterdam, Amsterdam, the Netherlands
[2] University of Halle-Wittenberg, Halle, Germany

Abstract. Secure Neural Network Inference (SNNI) protocols enable privacy-preserving inference by ensuring the confidentiality of inputs, model weights, and outputs. However, large neural networks, particularly Transformers, face significant challenges in SNNI due to high computational costs and slow execution, as these networks are typically optimized for accuracy rather than secure inference speed. We present COLIBRI, a novel approach that optimizes neural networks for efficient SNNI using Neural Architecture Search (NAS). Unlike prior methods, COLIBRI directly incorporates SNNI execution time as an optimization objective, leveraging a prediction model to estimate execution time without repeatedly running costly SNNI protocols during NAS. Our results on Cityscapes, a complex image segmentation task, show that COLIBRI reduces SNNI execution time by 26–33% while maintaining accuracy, marking a significant advancement in secure AI deployment.

Keywords: Secure Neural Network Inference · Privacy-Preserving AI · Neural Architecture Search · Transformers · Visual Segmentation · Secure Inference.

1 Introduction

In many real-world Artificial Intelligence (AI) applications, multiple stakeholders are involved, leading to specific security and privacy requirements [1]. In this paper, we focus on the situation where a company – the *model holder* – trains a neural network and then offers inference with this neural network as a service. A *client* can provide an input to the neural network and request inference on this input. The client's input and inference output may contain sensitive information that must be protected from unauthorized access. Additionally, the weights of the neural network constitute the model owner's intellectual property and must also be protected. As illustrated in Fig. 1, Secure Neural Network Inference (SNNI) protocols enable inference, fulfilling all the aforementioned secrecy requirements.

* This paper was published in the *Proceedings of the 40th IFIP International Conference on ICT Systems Security and Privacy Protection (IFIP SEC)*, pp. 17-31, 2025

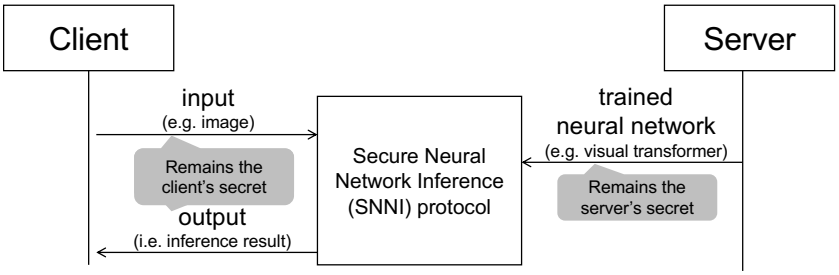

Fig. 1. Overview of the Secure Neural Network Inference (SNNI) process.

A significant challenge with SNNI is its high computation and communication demand [17]. SNNI requires complex cryptographic protocols that need substantial resources for the secure execution of even simple operations. Despite recent significant advancements making SNNI protocols more efficient [12], secure inference remains significantly slower than standard inference [10]. Another challenge is the increasing size and complexity of neural networks developed by the AI community to handle larger inputs and achieve higher accuracy [7].

A more subtle challenge is the mismatch between AI and SNNI objectives. For the AI community, achieving high accuracy is paramount, while keeping costs low is secondary. In SNNI, however, reducing secure inference time is critical, as excessive delays could render services impractical. Functions like SoftMax and ReLU, which minimally affect model size or standard inference time, are computationally expensive in secure execution [19].

This paper introduces COLIBRI, an innovative approach to creating transformer architectures with state-of-the-art accuracy and reduced SNNI execution time. COLIBRI uses neural architecture search (NAS), which automatically identifies the best neural architecture for a task by exploring candidate architectures. Previous NAS-based approaches to accelerate SNNI [13,19,21] optimized proxies such as the number of ReLU operations rather than explicitly targeting SNNI execution time. COLIBRI is the first approach to *optimize explicitly for SNNI execution time in a given hardware/software configuration*. This is important because bottlenecks in SNNI depend heavily on the hardware and software used (e.g. network bandwidth [15] and GPU availability [8]).

Relying on proxies for optimization can yield suboptimal results, while COLIBRI directly targets SNNI execution time for better outcomes. However, explicitly optimizing SNNI execution time during NAS is challenging due to the need to evaluate numerous neural architectures. Running the costly SNNI process for each candidate would result in prohibitive computational costs for NAS, requiring thousands of GPU days. To address this, COLIBRI employs a prediction model to estimate SNNI execution time for candidate architectures in a given setup, avoiding the need for actual SNNI execution during NAS.

We implemented COLIBRI using the HR-NAS search algorithm [5] for visual transformers and the CrypTen SNNI protocol [9]. Because the COLIBRI

approach is orthogonal to how NAS algorithms and SNNI protocols work, COLIBRI is adaptable to different choices for implementation, making it versatile for various AI tasks. Our implementation achieves efficient SNNI for large transformers in visual segmentation tasks, which are more complex than simpler benchmarks (e.g., image classification [4, 21]). Empirical results show that COLIBRI-generated visual transformers reduce SNNI execution time by 26-33% compared to baseline NAS approaches, while maintaining accuracy.

2 Preliminaries

Secure Neural Network Inference (SNNI) protocols enable collaborative AI inference without sharing confidential data, as shown in Fig. 1. Clients encrypt input data, while model holders protect the secrecy of trained weights, ensuring secure execution of the inference process. Only the client receives the output, with cryptographic guarantees provided by secure multi-party computation (MPC) or homomorphic encryption (HE) [12].

SNNI protocols simulate neural network operations using cryptographic functions, tailored to each layer type. Linear layers (e.g., fully connected or convolutional) are relatively efficient, while non-linear operations like SoftMax and ReLU are computationally intensive under MPC/HE [12]. Our implementation uses CrypTen [9], an MPC-based SNNI protocol, though COLIBRI is compatible with any SNNI protocol.

Neural Architecture Search (NAS) [16] automates the discovery of optimal neural network architectures by exploring a space of possible architectures. NAS evaluates performance metrics like accuracy and computational efficiency, to retain or prune architectures from the search space. NAS uses methods like evolutionary algorithms or gradient-based optimization to perform the search. Modern NAS approaches often use a *supernet* – a large, over-parameterized neural network that encompasses numerous potential architectures [2]. For example, in HR-NAS [5], the supernet consists of 51 search blocks, and each search block is composed of Search Units (SUs). A Search Unit (SU) represents possible architecture paths, such as mini-Transformers or convolution-based sequences. Pruning reduces each search block to at most one SU, while iteratively shrinking dimensions (e.g., matrix sizes, channels, tokens, strides) within SUs. Supernet training combines pruning and training, lowering computational costs. In HR-NAS, the size of search units is penalized, while their contribution to task accuracy is rewarded [5]. The process outputs a trained neural network.

Predicting SNNI time can be done with a prediction model, accounting for the SNNI protocol and hardware/software configuration [20]. The model uses formulas to predict execution time for various layer types. In an offline phase, parameters in the formulas are fine-tuned using measured SNNI execution times across layer types and sizes. This calibration captures how layer types and sizes influence execution time for the given configuration.

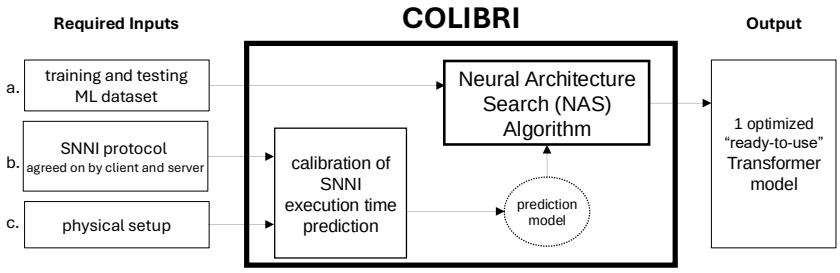

Fig. 2. Overview of the COLIBRI approach.

3 Our Approach

By combining NAS with SNNI and a prediction model for predicting SNNI execution time of candidate neural architectures, COLIBRI offers a first-of-its-kind approach that directly minimizes predicted SNNI execution time in the given client/server setup during NAS, going beyond existing methods that merely aim at reducing the use of allegedly expensive operations like ReLU or SoftMax. As shown in Fig. 2, COLIBRI takes three inputs: (a) the training and testing datasets for an AI task to be performed by the neural network, (b) the chosen SNNI protocol that client and server will use for secure inference, and (c) the used client/server setup, including hardware and software. The output of COLIBRI is a trained Neural Network, optimized for high accuracy and low SNNI execution time in the given client-server setup, ready to be used for secure inference. COLIBRI is adaptable to various SNNI protocols, NAS algorithms, AI tasks, and Neural Network types. In our implementation, we used the CrypTen SNNI protocol [9] and the HR-NAS [5] search algorithm for visual transformers. A central part of COLIBRI is the prediction model that is used to predict SNNI execution time for a neural network in a given hardware/software setup.

3.1 SNNI execution time prediction model

The SNNI execution time prediction model helps the NAS algorithm optimize Neural Network architectures by minimizing SNNI execution time, avoiding the need to execute the costly SNNI protocol for all candidate architectures. This explicit optimization is a key advantage of COLIBRI over traditional NAS methods, which often rely on metrics like parameter count or FLOPs, poorly correlating with SNNI execution time. COLIBRI's prediction model builds on the analytical prediction approach from [20] (see Table 1). Typically, SNNI protocols execute the secure inference process layer by layer. Similarly, our prediction model predicts the SNNI execution time per layer, summing these to estimate the total SNNI execution time. The secure execution time T_i of layer i is calculated as:

$$T_i = c_1 \cdot S_{\text{in}} + c_2 \cdot S_{\text{par}} + c_3 \cdot N_{\text{op}} + c_4 \cdot S_{\text{out}} + c_5. \quad (1)$$

Table 1. Factors contributing to SNNI execution time for different layer types [20]. N_I/N_O: number of inputs/outputs; C_I/C_O: number of input/output channels; H_I/W_I: input height/width; H_O/W_O: output height/width; H_F/W_F filter height/width; N: number of neurons; C: number of channels; H/W: height/width of input and output

Layer type	$S_{\mathbf{in}}$	$S_{\mathbf{par}}$	$N_{\mathbf{op}}$	$S_{\mathbf{out}}$
Fully connected	N_I	$(N_I+1)N_O$	$2N_IN_O+N_O$	N_O
Convolutional	$H_IW_IC_I$	$C_IH_FW_FC_O$	$2C_IH_FW_FH_OW_OC_O$	$H_OW_OC_O$
MaxPool / AveragePool	H_IW_IC	0	$H_FW_FH_OW_OC$	H_OW_OC
Batch normalization	HWC	$2C$	$2HWC$	HWC
ReLU	N	0	N	N

Here, S_{in}, S_{par}, and S_{out} are the input, parameter, and output sizes, respectively, of layer i, and N_{op} is the number of operations in non-secure inference with layer i. These parameters are estimated for different types of layers as presented in Table 1. The coefficients $c_1, \ldots, c_5$ capture how factors like input size and parameter size affect a layer's secure execution time. These values depend on the SNNI protocol, layer type, and client/server hardware and network setup.

COLIBRI's prediction model consists of sub-models for each supported neural network layer type, following Equation (1). The coefficients $c_1, \ldots, c_5$ and the calculations of S_{in}, S_{par}, S_{out}, and N_{op} vary by layer type, as summarized in Table 1. This model enables fast and accurate SNNI execution time predictions.

The prediction model used in COLIBRI can be calibrated for various SNNI protocols and different hardware and software properties of the client, the server, and the network connection between them. Here, the calibration is done by adjusting the coefficients $c_1, \ldots, c_5$ in each layer-type-specific sub-model. To achieve this, $c_1, \ldots, c_5$ are derived from a regression analysis of actual SNNI execution times, measured when running the specific SNNI protocol in the given client/server setup. The calibration phase involves creating neural networks with layers of different types and sizes and measuring the per-layer SNNI execution time by running the protocol in the specific client/server setup. The linear regression model is trained with non-negative coefficients (from Equation (1)) for each layer type to derive $c_1, \ldots, c_5$.

The simple structure of the prediction model ensures fast and accurate predictions, as validated in [20]. Trained on actual SNNI execution times, the model reliably captures protocol and setup impacts, enabling the NAS algorithm to make informed decisions about candidate architectures.

3.2 NAS algorithm

Using the calibrated prediction model, NAS aims to produce a Neural Network optimized for high accuracy and low SNNI execution time. The NAS algorithm trains and prunes a supernet iteratively. Pruning is done incrementally, removing the least promising parts of the supernet. Typically, in NAS algorithms, the metrics that form the basis for pruning decisions are combined into a loss function. Thus, in COLIBRI, the output of the SNNI execution time prediction model

is integrated into the loss function. In the following discussion, we describe the change required for a representative NAS algorithm, HR-NAS [5].

The original loss function of HR-NAS is:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{cost}} \tag{2}$$

Where $\mathcal{L}_{\text{task}}$ represents the standard loss used in training for the specific machine learning task. The second term, $\mathcal{L}_{\text{cost}}$, is an L1 penalty term that captures the complexity of the neural network architecture. This penalty term aims to drive the overall complexity of the supernet to a minimum, and is defined as:

$$\mathcal{L}_{\text{cost}} = \lambda_{\text{FLOPs}} \cdot \sum_{i \in SU} \Delta_i \cdot |\alpha_i| \tag{3}$$

Where SU is the set of search units in the supernet, and for every search unit i, Δ_i is its computational cost, while α_i is a trainable parameter, denoting the importance factor of search unit i in maintaining or enhancing the model's accuracy. λ_{FLOPs} is an empirically determined coefficient for the L1 penalty.

In HR-NAS, the computational cost for every search unit, denoted as Δ_i, is measured in FLOPs alongside a resource-aware term that quantifies the search unit's overall cost. Pruning search unit i during the NAS would reduce the computational cost of the model by Δ_i. This value can be determined based on the shape of the search unit (i.e., the shape of its designated input matrix, its number of channels and kernel sizes for convolutional units, and number of tokens for Transformer units) in addition to the resource-aware factor reflecting the search unit's computational demands. In COLIBRI, the $\mathcal{L}_{\text{cost}}$ penalty term is modified to integrate the predicted SNNI execution time, as follows:

$$\mathcal{L}_{\text{cost}}^{\text{COLIBRI}} = \lambda_{\text{sec}} \cdot \sum_{i \in SU} \Gamma_i \cdot |\alpha_i| \tag{4}$$

Here, Γ_i is the predicted SNNI execution time of search unit i, generated by the prediction model. While the α_i coefficient remains semantically the same as in the original penalty term, the λ_{FLOPs} coefficient is renamed to λ_{sec} to reflect the switch in the computational cost metric from non-secure inference cost in FLOPs to SNNI execution time in seconds. Compared to the loss of HR-NAS in (3), the loss of COLIBRI in (4) leads to different relative penalties for various search units. Operations such as ReLU that are expensive in secure inference have relatively higher values of Γ_i and thus a higher penalty in the loss.

The NAS algorithm in COLIBRI, utilizing the loss $\mathcal{L}_{\text{cost}}^{\text{COLIBRI}}$, is summarized as pseudocode in Algorithm 1. The process starts by initializing the supernet M_Θ, where Θ represents the trainable weights. The model is trained over E epochs. For each epoch, the loss function ($\mathcal{L}_{\text{task}} + \mathcal{L}_{\text{cost}}$) is computed (line 3), and the supernet is trained with this loss function (line 4). After each epoch, and for every Search Unit, the importance factor α_i is computed (line 7), while the prediction model is used to compute Γ_i (line 6). At every pre-defined pruning interval (pri) epochs, the least important search units with the smallest α_i are removed (line 10), while the remaining SU_i are pruned by an amount r_j (line 12). r_j is relative to the height and width of the input of a given search unit. The process repeats until the model is fully trained and optimized for fast SNNI execution. The final optimized model is returned by the algorithm (line 17).

After NAS finishes, the resulting Neural Network is deployed. Client and server can collaboratively execute the chosen SNNI protocol, achieving secure

Algorithm 1: COLIBRI Neural Architecture Search

Input :
- supernet M_Θ (where Θ are the trainable weights),
- set of search units SU,
- prediction model Pred_L,
- number E of epochs for training,
- pruning interval pri,
- relative pruning amount r (%).

Output: A trained model optimized for fast SNNI execution time.

```
epoch = 0;
while epoch < E do
    M_Θ.train() ;
    foreach i in SU do
        Compute Γ_i = Pred_L(i);
        Calculate α_i, the importance factor for i;
    end
    Calculate L_task and L_cost for one epoch;
    if epoch%pri == 0 then
        Remove SU_i with min(α);
        foreach remaining j in SU do
            Prune j by amount r_j;
        end
    end
    epoch+ = 1;
end
return optimized M_Θ
```

inference with high accuracy and low SNNI execution time, as optimized by COLIBRI. The neural network can be reused for secure inference multiple times.

4 Experiments

We evaluated COLIBRI using visual transformers on the Cityscapes dataset [3], which reflects the complexity of real-world AI tasks like high-resolution urban scene segmentation. By optimizing for accuracy and SNNI execution time, COLIBRI proves effective for real-world applications.

Furthermore, we used two Virtual Machines to simulate the two parties performing SNNI with COLIBRI's output model. The same setup was used to calibrate the prediction model and measure SNNI execution time. To ensure robustness, experiments were conducted in two setups, requiring separately calibrated prediction models, to compare COLIBRI and HR-NAS outputs. Setup 1 consisted of two identical virtual machines, each having 32GB of RAM with 8 double-thread CPU cores; the machines communicated with 7Gbps bandwidth (10ms ping time). Setup 2 consisted of two physical servers, each with 64GB of

RAM and 8 double-thread CPU cores, and 10Gbps bandwidth (5ms ping time). Both Setup 1 and 2 machines ran the Debian 5.10 Linux OS and operated on 8 double-thread CPU cores. We implemented COLIBRI with CrypTen [9] 0.4.1, HR-NAS [5], and Python 3.9.2.

NAS training was conducted in a high-performance environment to handle the computational demands of training and hyperparameter tuning. Nodes with 1-2 GPUs (Tesla A40 or A10), 60-68 single-thread CPU cores, and 50GB RAM were used. NAS training for every Transformer was set at epochs $E = 500$, with iterative supernet pruning set at $pri = 20$. For the λ coefficient in the loss function, we experimented with two different values. The first value, $\lambda_{\text{sec}} = 0.00014$, was empirically determined to correspond to the original λ_{FLOPS} used in HR-NAS [5]. The second value, $\lambda_{\text{sec}} = 0.00028$, was chosen because COLIBRI uses λ in units of seconds instead of FLOPs, which influences the effect of λ on the final loss term.

Prediction model calibration. To perform the calibration of the prediction model, three types of neural networks were created, each with different combinations of layers, resulting in a total of 30 neural networks, each consisting of 15 to 40 layers. Every layer has a random selection (from a predefined pool of viable options) of hyper-parameters, including kernel size, stride, padding, and number of input and output channels. All of these layers were converted to their secure counterpart for the CrypTen protocol [9]. SNNI execution time of each layer of these neural networks was measured when executing the protocol in Setup 1 and Setup 2 separately. To ensure reliable data collection, each neural network was assigned one of ten random seeds to initiate the protocol, while inference time on each neural network was measured 5 times. The accuracy of these neural networks was irrelevant for the prediction model, so training was omitted. The linear regression for predictions was created using the scikit-learn 1.5.2 module in Python and saved into usable files with joblib 1.4.2. To evaluate the regression model, Mean Absolute Scaled Error (MASE) and R^2 scores were used.

Comparison methodology and metrics. To evaluate the performance of COLIBRI, we compare the Transformers generated by COLIBRI with those produced by the original NAS algorithm, HR-NAS [5], under similar conditions. A key advantage of HR-NAS is its stable block structure; pruning affects only search units within blocks, not the blocks themselves. As a result, models generated by both HR-NAS and COLIBRI have the same number of search blocks and a similar number of layers, differing only in layer dimensions and function types. This allows a direct comparison of their effectiveness without the confounding factor of architecture depth. Transformers produced by COLIBRI and HR-NAS are compared based on three main characteristics: size (# of parameters), accuracy on the AI task, and SNNI execution time. For the task of image segmentation, accuracy is measured per input image with two metrics: mean Intersection over Union (mIoU) and mean Accuracy (mAcc).

Table 2. Evaluation of Prediction model in Setup 2.

Layer type	Average Real SNNI time (s)	Average Predicted SNNI time (s)	R^2	MASE
Fully Connected	0.26	0.26	0.99	0.02
Convolutional	0.79	0.79	0.99	0.05
MaxPool	15.47	15.54	0.99	0.03
AveragePool	0.002	0.002	0.81	0.41
Batch normalization	0.2	0.12	0.98	0.37
ReLU	7.03	6.86	0.97	0.08

Table 3. Results for the image segmentation dataset Cityscapes. Average over 20 inferences. Best SNNI execution times per setup are marked bold.

QUOKKA option	Confidence Threshold	SNNI time (s) CLIENT	SNNI time (s) SERVER	Accuracy (%)	Accuracy drop from plaintext (%)

5 Results

Prediction model validation. The regression models developed for each layer type demonstrated high predictive accuracy for SNNI execution times. Each regression model was trained using the measured execution times specific to the corresponding layer type (see Table 2).

The low MASE values across all layer types indicate minimal prediction errors relative to the actual execution times. Additionally, R^2 scores are consistently close to 1.0, highlighting the models' effectiveness in capturing the variance in the measured SNNI execution times. These results confirm that the predicted execution times closely align with the real SNNI execution times, validating the accuracy of our regression approach for modeling SNNI execution times across different layer types.

Table 2 shows that among all layer types, ReLU and MaxPool are more time-consuming to compute using SNNI. For ReLU, this is consistent with findings in [19]. MaxPool has been less tested in the context of SNNI work, which often excludes it in favor of convolutional, ReLU, and fully connected layers [12]. Our MaxPool measurements (and predictions) are, however, in line with [20]. Indeed, like ReLU, MaxPool is not trivially computed in a secure protocol as no information can be revealed on which value of a matrix is the maximum, thus requiring complex cryptographic primitives [18].

SNNI execution time. Given the particular complexity of the segmentation task and the size of the images in Cityscapes, SNNI with CrypTen on relevant visual transformers takes several minutes per image in our inference setup. To ensure that the evaluation does not take a prohibitively long time, we limited the evaluation to a randomly selected subset of the Cityscapes testing dataset. We opted for randomly selecting 5 images and running inference four times on each of them. This led to 20 results for each combination of technical

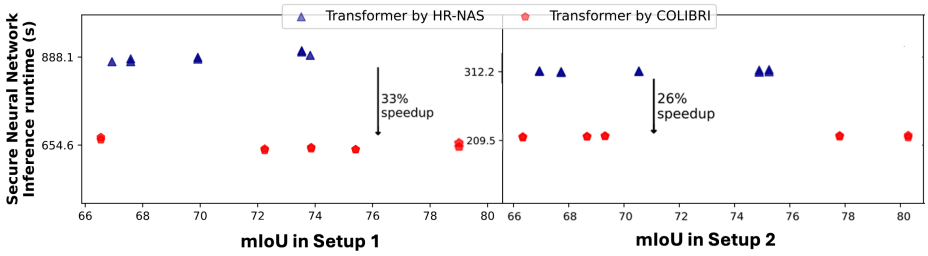

Fig. 3. Measured SNNI execution time and accuracy metric mIOU.

setup and tested transformer, allowing us to investigate variance across different images as well as variance between different runs on the same image.

Our results showed that accuracy and SNNI execution time differed little between images and between different runs on the same image. The results are shown in Fig. 3. Overall, our experiments show that COLIBRI-generated Transformers have consistently lower SNNI execution time than HR-NAS-generated Transformers. The average speedup of COLIBRI-generated Transformers over HR-NAS-generated Transformers is 33% in Setup 1 and 26% in Setup 2. A Mann-Whitney U test with $\alpha = 0.001$ confirmed that the difference in SNNI execution time between transformers generated by COLIBRI versus HR-NAS is *statistically significant*. Moreover, the COLIBRI-generated Transformer was faster than the HR-NAS-generated Transformer *on each image*.

The validity of COLIBRI was further demonstrated when looking at task accuracy. Here, we aimed at accuracy retention rather than accuracy drop. Fig. 3 shows some variation between images, but a Mann-Whitney U test with $\alpha = 0.001$ confirmed that there is *no statistically significant difference* between COLIBRI-generated Transformers and HR-NAS-generated Transformers in terms of accuracy. This holds true for both of the considered accuracy metrics and for both of the considered technical setups. Thus, overall, we can state that COLIBRI leads to significantly faster transformers without affecting accuracy. Further details of the experimental results are given in Table 3. An interesting phenomenon that can be observed from this table is that for Setup 1, transformers created by COLIBRI are larger (i.e., have more parameters) than the transformer created by HR-NAS. Still, secure inference with the COLIBRI-generated transformers is faster than with the HR-NAS-generated transformer. This once again demonstrates that size is a poor predictor of SNNI execution time, and that COLIBRI's way of explicitly optimizing for SNNI execution time achieves the desired effect.

Overall, the experimental results demonstrate the efficacy of using COLIBRI over another NAS algorithm. This not only proves the feasibility of SNNI execution time prediction in NAS but also highlights the potential of this approach to produce Neural Networks optimized for secure inference.

6 Discussion

High SNNI execution time. Despite the speedup achieved by COLIBRI, SNNI execution time is about 10 minutes in Setup 1 and about 3 minutes in Setup 2. The appropriateness of this duration may vary depending on the specific use case. Notably, these times are achieved on a complex segmentation task with large images, exceeding the complexity of typical benchmarks used in SNNI evaluations. If desired, SNNI execution time can be mitigated in several ways: upgrading the technical setup (e.g., higher CPU capacity, increased bandwidth, reduced network latency) or using a faster SNNI protocol than CrypTen, such as [8], which offers improved efficiency.

Applying COLIBRI to different types of SNNI protocols. COLIBRI is compatible with other and newer SNNI protocols, which may differ significantly from CrypTen. For instance, some involve more than two parties [6], protect against malicious adversaries [18], or separate input-independent operations into an offline preprocessing phase [8]. Investigating the speedups COLIBRI can achieve with such protocols would be an interesting avenue for future research.

Varying technical setups. COLIBRI fine-tunes neural network architectures for fast SNNI in a specific client/server setup, making it ideal for scenarios where the setup is consistent and used repeatedly. If the inference setup differs from the one used for calibrating COLIBRI's prediction model, the neural network will still function accurately and securely, but SNNI execution time may be suboptimal. One way to address this is to run COLIBRI multiple times, creating networks optimized for various setups. During inference, the network optimized for the setup closest to the current conditions can be used. Future research could focus on automating this process to enhance efficiency and effectiveness.

7 Related work

Recent advances in Secure Neural Network Inference (SNNI) focused on improving computational efficiency by simplifying non-linear layers like SoftMax and ReLU, which are computational bottlenecks in secure protocols. Existing approaches primarily target two strategies: (1) reducing or removing these layers entirely to simplify inference [18], or (2) applying polynomial approximations to linearize them, thereby streamlining the underlying MPC protocol [10]. While effective, these methods focus narrowly on layer-specific optimizations and overlook broader architectural inefficiencies.

Some efforts have employed neural compression techniques, such as knowledge distillation and NAS, to reduce neural network complexity without compromising accuracy. These approaches have largely focused on Convolutional Neural Networks (CNNs) [14], with limited exploration of Transformers. For Transformers, methods like MPCFormer [10] and SecFormer [11] leverage knowledge distillation and approximate SoftMax layers, while MPCViT [19] and PriViT [4] reduce reliance on costly non-linear functions. SAL-ViT [21] employs NAS to

optimize attention heads, but all these approaches lack a comprehensive strategy for minimizing overall SNNI execution time. Additionally, they often focus on simplified tasks or benchmarks, limiting real-world applicability.

COLIBRI introduces a novel direction by targeting a previously overlooked variable: predicted SNNI execution time during neural network design. Unlike existing methods, COLIBRI integrates a calibrated prediction model directly into the NAS process, enabling optimization of architectures specifically for SNNI execution time. This approach is compatible with existing techniques for accelerating non-linear layers but shifts the focus to holistic architectural optimization. To maintain consistency, we employed HR-NAS [5], ensuring the number of SoftMax and ReLU layers remains constant by keeping the search unit structure fixed. Instead, COLIBRI prunes layer dimensions based on predicted SNNI execution time, providing targeted and flexible optimization.

COLIBRI's Contribution While prior work has evaluated SNNI on relatively small tasks, COLIBRI addresses the challenges of complex, real-world applications. By optimizing speed and accuracy for tasks like image segmentation while accounting for hardware and network factors, COLIBRI sets a new benchmark for SNNI acceleration research. A comparison of COLIBRI with related methods is provided in Table 4.

We are the first to apply these techniques to visual transformers in complex image segmentation tasks, such as Cityscapes, surpassing the simpler benchmarks typically used for SNNI evaluation. COLIBRI's adaptability to various protocols and hardware setups represents a significant advancement in the field.

8 Conclusions

This paper has introduced COLIBRI, a novel approach for automatically creating neural networks – particularly Transformers – with high accuracy and low SNNI execution time. By integrating an SNNI execution time prediction model into NAS, COLIBRI enables the design of architectures explicitly optimized for fast secure inference without running the full SNNI protocol during NAS. This makes COLIBRI scalable and efficient for complex tasks.

Table 4. Summary of approaches related to COLIBRI. KD: Knowledge Distillation

Approach	**Compression technique**	**Optimization target**	**Supports complex segmentation**	**Low compute setup**
MPCFormer [10]	KD	-	No	No
SecFormer [11]	KD	-	No	No
MPCVit [19]	NAS/KD	# of attention heads	No	Yes
PriViT [4]	KD	-	No	-
SAL-ViT [21]	NAS	Hybrid attention costs	No	Yes
COLIBRI	NAS	Predicted SNNI time	Yes	Yes

Our work opens a new avenue of research into SNNI execution time prediction during NAS, emphasizing the importance of explicitly considering SNNI execution time when optimizing AI models. Departing from previous works that only targeted the number of allegedly expensive functions, such as ReLU, in their NAS strategy, COLIBRI enables the creation of Transformers optimized for a specific SNNI setup.

In the future, we aim to investigate COLIBRI performance with different underlying SNNI protocols and NAS algorithms, as well as with different physical setups. It can be beneficial to assess memory utilization and communication overhead within SNNI protocols to inform future modifications.

References

1. Chan, K.Y., Abu-Salih, B., Qaddoura, R., Ala'M, A.Z., Palade, V., Pham, D.S., Del Ser, J., Muhammad, K.: Deep neural networks in the cloud: Review, applications, challenges and research directions. Neurocomputing (2023)
2. Chitty-Venkata, K.T., Emani, M., Vishwanath, V., Somani, A.K.: Neural architecture search for transformers: A survey. IEEE Access **10** (2022)
3. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The Cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE Conference on CVPR (2016)
4. Dhyani, N., Mo, J., Cho, M., Joshi, A., Garg, S., Reagen, B., Hegde, C.: PriViT: Vision transformers for fast private inference. arXiv preprint arXiv:2310.04604 (2023)
5. Ding, M., Lian, X., Yang, L., Wang, P., Jin, X., Lu, Z., Luo, P.: HR-NAS: Searching efficient high-resolution neural architectures with lightweight transformers. In: Proceedings of the IEEE/CVF Conference on CVPR (2021)
6. Dong, Y., Xiaojun, C., Jing, W., Kaiyun, L., Wang, W.: Meteor: improved secure 3-party neural network inference with reducing online communication costs. In: Proceedings of the ACM Web Conference (2023)
7. Gholami, A., Yao, Z., Kim, S., Mahoney, M.W., Keutzer, K.: AI and memory wall. RiseLab Medium Post (2021)
8. Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: FSS-based secure training and inference with GPUs. In: IEEE Symposium on Security and Privacy (SP). IEEE (2024)
9. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: CrypTen: Secure multi-party computation meets machine learning. Advances in Neural Information Processing Systems (NeurIPS) **34** (2021)
10. Li, D., Shao, R., Wang, H., Guo, H., Xing, E.P., Zhang, H.: MPCFormer: fast, performant and private transformer inference with MPC. arXiv preprint arXiv:2211.01452 (2022)
11. Luo, J., Zhang, Y., Zhang, J., Mu, X., Wang, H., Yu, Y., Xu, Z.: SecFormer: Fast and accurate privacy-preserving inference for transformer models via SMPC. arXiv preprint arXiv:2401.00793 (2024)
12. Mann, Z.Á., Weinert, C., Chabal, D., Bos, J.W.: Towards practical secure neural network inference: The journey so far and the road ahead. ACM Computing Surveys **56**(5) (2023)
13. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference system for neural networks. In: 29th USENIX Security Symposium. USENIX Association (2020)

14. Peng, H., Huang, S., Zhou, T., Luo, Y., Wang, C., Wang, Z., Zhao, J., Xie, X., Li, A., Geng, T., et al.: AutoReP: Automatic ReLU replacement for fast private network inference. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2023)
15. Prins, J., Mann, Z.Á.: Secure neural network inference for edge intelligence: Implications of bandwidth and energy constraints. In: IoT Edge Intelligence. Springer (2024)
16. Ren, P., Xiao, Y., Chang, X., Huang, P.Y., Li, Z., Chen, X., Wang, X.: A comprehensive survey of neural architecture search: Challenges and solutions. ACM Computing Surveys (CSUR) **54**(4) (2021)
17. de Vries, R., Mann, Z.Á.: Secure neural network inference as a service with resource-constrained clients. In: Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC) (2023)
18. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: FALCON: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies (2021)
19. Zeng, W., Li, M., Xiong, W., Tong, T., Lu, W.j., Tan, J., Wang, R., Huang, R.: MPCViT: Searching for accurate and efficient MPC-friendly vision transformer with heterogeneous attention. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2023)
20. Zhang, E., Mann, Z.Á.: Predicting the execution time of secure neural network inference. In: IFIP International Conference on ICT Systems Security and Privacy Protection. Springer (2024)
21. Zhang, Y., Chen, D., Kundu, S., Li, C., Beerel, P.A.: SAL-ViT: Towards latency efficient private inference on ViT using selective attention search with a learnable softmax approximation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2023)