Adaptive and Collaborative Inference: Towards a No-compromise Framework for Distributed Intelligent Systems

Alireza Furutanpey[a] and Schahram Dustdar[b]
Distributed Systems Group, TU Wien, Argentinierstrasse 8, Vienna, Austria

Keywords: Edge Computing, Edge Intelligence, Dynamic Neural Networks, Split Computing.

Abstract: Deep Neural Networks (DNNs) are the backbone of virtually all complex, intelligent systems. However, networks which achieve state-of-the-art accuracy cannot execute inference tasks within a reasonable time on commodity hardware. Consequently, latency-sensitive mobile and Internet of Things (IoT) applications must compromise by executing a heavily compressed model locally or offloading their inference task to a remote server. Sacrificing accuracy is unacceptable for critical applications, such as anomaly detection. Offloading inference tasks requires ideal network conditions and harbours privacy risks. In this position paper, we introduce a series of planned research work with the overarching aim to provide a (close to) no compromise framework for accurate and fast inference. Specifically, we envision a composition of solutions that leverage the upsides of different computing paradigms while overcoming their downsides through collaboration and adaptive methods that maximise resource efficiency.

1 INTRODUCTION

From computer vision (CV) to natural language processing (NLP), Deep Learning (DL) methods achieved unprecedented improvement in a wide range of areas. The consistent improvements gave rise to intelligent systems capable of performing complex tasks, such as understanding voice commands and detecting anomalies from a visual feed. However, Deep Neural Network (DNN) models which can reliably execute inference tasks incur high resource consumption on most available hardware. Accordingly, both academia and industry dedicate much attention to conceiving solutions to accommodate local inference tasks in constrained environments. They range from devising novel compact architectures (Sandler et al., 2018) to altering existing architectures via quantisation, pruning, or Knowledge Distillation (Deng et al., 2020). An inherent shared trait of all such approaches is their performance and model capacity trade-offs. Alternatively, mobile applications can offload the inference tasks to a cloud server where we assume unlimited resources capable of fast inference regardless of load. Here, the bottleneck is transferring the input data to a remote server. Especially for CV tasks, there remain three caveats. First, sensitive private data is exposed to application operators and third-party cloud providers. Second, continuous visual data streams must compete for limited bandwidth. Third, transferring the input data results in unacceptable end-to-end latency for time-critical DL-enabled applications. Another alternative is to bring resources to an edge server near the clients. Ideally, such edge servers are equipped with specialised hardware capable of accommodating large models. Nevertheless, simply moving models from the cloud to the edge introduces other limitations. Unlike with cloud servers, we cannot effortlessly horizontally scale edge resources, imposing a hard cap on throughput. Moreover, most approaches disregard fluctuating network conditions in dense urban areas, i.e. even with servers in close proximity, it is difficult to ensure smooth operations regardless of the current load. Lastly, privacy-related concerns are not entirely resolved. Table 1 sum-

Table 1: Strength and Limitations of each compute paradigm. (✓: Acceptable for critical applications, ∼: Limited, ×: Unsuitable for critical applications).

	LC	EC	CC
Latency	✓	∼	×
Accuracy	×	✓	✓
Privacy	✓	∼	×
Throughput	×	∼	✓

[a] https://orcid.org/0000-0001-5621-7899
[b] https://orcid.org/0000-0001-6872-8821

Furutanpey, A. and Dustdar, S.
Adaptive and Collaborative Inference: Towards a No-compromise Framework for Distributed Intelligent Systems.
DOI: 10.5220/0011547800003318
In *Proceedings of the 18th International Conference on Web Information Systems and Technologies (WEBIST 2022)*, pages 144-151
ISBN: 978-989-758-613-2; ISSN: 2184-3252

marises the strengths and limitations of local computing (LC), edge computing (EC) and cloud computing (CC).

Conclusively, neither standalone local, edge, nor cloud computing is a viable solution to satisfy common performance requirements simultaneously. Instead, we advocate for methods that facilitate collaboration between the computing paradigms. More broadly, edge intelligence (EI) has emerged as a research area that situates EC as the central component of a hierarchical network spanning the cloud, the edge and end devices (Zhou et al., 2019). We believe that the edge-centric view of EI is the correct way forward for solving the challenges of intelligent distributed systems.

This position paper presents a series of planned research work based on Split Computing (SC) and dynamic neural networks for collaborative and adaptive inference. SC refers to distributing a DNN over a hierarchical network. Dynamic neural networks have dynamic computational graphs that can adapt to different inputs and conditions during inference. The role of SC is to break the narrow view of solutions which view each paradigm in isolation. Dynamic neural networks complement SC approaches by maximising the usage of the available resources to ensure consistent performance in fluctuating conditions. The paradigms intersect in enabling powerful networks to execute their tasks efficiently without over-applying crippling compression techniques.

Section 2 introduces the relevant concepts and summarises existing literature to provide context. Then, Section 3 discusses open problems, which we aim to address with a series of planned research work. Lastly, Section 4 concludes our work by summarising our main insights.

2 LITERATURE REVIEW

2.1 Split Computing

Consider a DNN $\mathcal{D}(.)$ as a stack of functions $y = f^{(n)} \dots f^{(2)}(f^{(1)}(x)) \dots)$. Each function is either a fine-grained layer or a more coarse-grained block which consists of multiple layers. The input x is some data, such as a 3-dimensional image tensor $C \times H \times W$, and y is the result of some task, such as the one-hot-encoded vector for image classification. SC divides $\mathcal{D}(.)$ into a head $\mathcal{D}_H$ and tail $\mathcal{D}_T$ portion s.t. $y = \mathcal{D}_T(\mathcal{D}_H(x))$ where the head is deployed on an end device and the tail on a remote server. Then, instead of either computing y locally or sending x over some network, the end device computes the intermediate features $z = \mathcal{D}_H(x))$ locally before sending it to a remote server to compute $y = \mathcal{D}_T(z)$. The idea stems from the observation that the input data tends to reduce as it progresses through the layers. Applying this observation to SC was first thoroughly explored by Kang et al. Their method *Neurosurgeon* is a scheduler that profiles each layer by computational cost and output data size and ideally splits the network to optimise latency or energy efficiency (Kang et al., 2017).

For an overview on SC, see the recent survey by Matsubara et al (Matsubara et al., 2021). In the following, we focus on work relevant to our discussion in Section 3.

2.1.1 Head Distillation and Neural Compression of Intermediate Features

Early work on SC focused on seeking ideal splitting points of existing DNN models without any considerable modifications to their architecture (Jeong et al., 2018). However, as pointed out by Kang et al., naive SC is only sensible for networks with a *natural bottleneck* at earlier layers, excluding many recent DNN architectures. For example, ResNet-50 does not decrease the input until its third last residual block (He et al., 2016). Since such models shift most work to the end device, it is pointless to split them. Consequently, more recent work focuses on modifying existing architectures with *artificial bottlenecks*. The objective is to introduce and train learnable autoencoder-based neural compression models (Mentzer et al., 2018) at the desired split point of the DNN.

Autoencoders consist of an *encoder* and a *decoder*. The encoder compresses the input to a low-dimensional representation while the decoder reconstructs it. The advantage of autoencoders is twofold. First, autoencoders can learn to extract useful features for downstream tasks without labels (i.e. unsupervised). Second, they introduce a bottleneck providing an ideal partition point. Typically, the architecture of autoencoders is symmetric, i.e. the layers in the decoder are inverse to the layers in the encoder. To mirror the unbalanced computational resources between the client and the server, it is possible to achieve comparable results with an asymmetric architecture where more of the computational burden is shifted towards the decoder (Yao et al., 2020). Additionally, we can fine-tune autoencoder modules embedded in a larger DNN to achieve the best inference result rather than reconstructing an input for human perception. Clearly, compression techniques with learnable parameters will outperform conventional compression techniques.

Eshratifar et al. were some of the first to introduce artificial bottlenecks by injecting an autoencoder into an existing model (Eshratifar et al., 2019). Shao and Zang continued their work and conducted experiments with over-compressed features in channels with varying conditions (Shao and Zhang, 2020). More recently, Yao et al. introduced a general framework for such approaches (Yao et al., 2020).

Matsubara et al. were the first to propose Head network Distillation (HDN), which only modifies the head portion of the model by replacing it with a distilled version (Matsubara et al., 2019). They further build on their initial work, such as introducing a generalised HDN method for object detection tasks and novel training methods for compressing intermediate features (Matsubara et al., 2022b).

Note the distinction between methods that embed autoencoders within a model and methods that introduce artificial bottlenecks by modifying existing model parts. Both objectives are to minimise the head's computational load and compress transferred data with an encoder-decoder structure, but there is a crucial distinction. The former injects an asymmetric autoencoder into the model to compress and reconstruct an input. The advantage of this approach is its generic application, but it trades a non-negligible amount of accuracy for compression strength. Contrastingly, the latter *remodels the existing architecture of the head and the first blocks of the tail* to function akin to an encoder-decoder structure. This approach can virtually retain the accuracy of the original model even for a substantial amount of compression but requires careful consideration for each model (Matsubara et al., 2022a).

2.1.2 Privacy Preserving Properties of Split Computing

The importance of protecting sensitive data is increasingly seeping into public consciousness, and various regions incrementally put it into law. Therefore, applications which require processing sensitive data are often constrained to local computing. Since SC processes the original input before sending it over a network, it possesses some privacy-preserving properties. Hence, privacy is a common justification for SC, but almost all work neglects to evaluate privacy-related claims. Presumably, this is due to the difficulty of quantifying the privacy-preserving properties of intermediate features.

Privacy-preserving claims were not entirely unjustified. Most literature on privacy exploits of DNNs focuses on the training data. In contrast, the reconstruction of inference data is more challenging since they do not contribute to the trained model parameters. Additionally, inference samples do not follow the same distribution, i.e. it is difficult to retrieve statistical information from them (He et al., 2019).

Oh & Lee performed an image reconstruction attack, targeting the reconstruction of the intermediate features back to the original input (Oh and Lee, 2019). Their attack requires access to the original model, and their results show that their attack only fails on models split at the deeper layers or complex models. Neither solution is satisfactory in practical scenarios. Splitting at a deeper layer shifts more computational burden on the end device, defeating the purpose of SC altogether. Furthermore, the current state-of-the-art focuses on simplifying the head model as aggressively as possible without sacrificing accuracy.

He et al. have shown that with no access to the model structure, it is still possible to accurately and reliably perform an image inversion attack. They suggest protective measures such as differential privacy by adding noise to the original input or intermediate features (He et al., 2019). However, they conducted their experiments on a small model with simple datasets and did not evaluate the suggested defence mechanisms. Wu et al. conceived a similar attack and have shown promising results on simple models by introducing noise to model parameters instead of to the input or intermediate representation (Wu et al., 2021).

2.2 Dynamic Neural Networks

The main advantage of Dynamic Neural Networks is their capability to allocate computational resources on demand by activating a subset of the model components depending on the input complexity and other constraints. The core assumption of this paradigm is that some tasks are easier to process than others. For example, a cluttered image where the object is barely visible is intuitively more difficult to classify than one where the object is well-centred and easily distinguishable from the background. Dynamic Neural Networks encompass a broad class of different methods and architectures. For a complete overview, we recommend the work by Han et al. (Han et al., 2021) Here, we focus on specific classes of Dynamic Neural Networks we believe are most compatible with EC and SC.

2.2.1 Early Exiting

Most work on early exiting (EE) is based on the work by Teerapittayanon et al. They extend existing architectures with intermediate classifiers to process easy samples without passing them through the entire network (Teerapittayanon et al., 2016). Superficially, this method appears to tie well with SC since we can simply place an early exit at an end device. However, in practice, it is a challenging problem with diverging objectives. For instance, the ideal splitting and branching locations do not necessarily coincide (Chiang et al., 2021). Teerapittayanon et al. extend their work on BranchyNet to conceive a framework for a distributed DNN deployed over a hierarchical network where a device can perform early inference with an intermediate classifier at each network layer (Teerapittayanon et al., 2017). Li et al. introduced *Edgent* and proposed combining SC with EE but only in a limited capacity, which does not distinguish between the complexity of the tasks (Li et al., 2019). Moreover, their experiments only considered a simple model with a small data set, which does not necessarily generalise to real-life scenarios. Laskaridis et al. introduced *SPINN*, which improves on Edgent by classifying simple tasks early and incorporates learned compression methods and have shown that this approach works on more recent models trained on larger datasets (Laskaridis et al., 2020).

2.2.2 Dynamic Depth and Width

While EE approaches are based on the assumption that some tasks are easier to classify than others, static pruning approaches assume that neural networks are overparameterised and that not all parts of the model attribute equally to the final classification task. EE increases inference speed by only executing parts of the DNN sequence but requires tuning the threshold value, the placement of the branches and training of the intermediate classifiers. The correct implementation of intermediate classifiers is non-trivial and will lead to a drop in accuracy without careful calibration (Pacheco et al., 2021). Pruning increases inference speed by permanently removing parts of the model but permanently reducing its capacity.

In a sense, DNNs with dynamic depth and width combine the assumptions of approaches to selectively skip redundant layers and channels based on the properties of the current input. In principle, such networks should achieve comparable speed-ups as pruning and EE without their downsides. For instance, a pruning algorithm removes parts of a model which were only necessary for edge cases and will subsequently permanently fail to classify such cases. Conversely, a dynamic neural network could skip the same parts and only activate them for the edge case it would otherwise misclassify.

Layer skipping approaches are especially promising on residual networks since they behave similarly to an ensemble of shallow networks (Veit et al., 2016). That is, individual blocks are removable without affecting other layers and learn features which are possibly relevant only for a subset of samples. However, layer skipping requires a discrete decision function which is not differentiable, i.e. we cannot compute the gradients needed for backpropagation during the training phase. Veit and Belongie (Veit and Belongie, 2018) propose a solution where they apply the Gumbel softmax trick (Jang et al., 2016) to a gating mechanism. Hermann et al. show that the same idea is transferable to *dynamic width*, where they use the same trick to learn a function skipping filters instead of blocks (Herrmann et al., 2020). Note that network depth and width are not orthogonal parameters and correlate. Hence, we additionally consider methods which combine multiple approaches, such as the work by Xia et al (Xia et al., 2021).

Lastly, we observe that combining SC with dynamic parameters is less obvious, which is presumably why there is no substantial work on it yet. Later, we discuss how we intend to bridge the concepts.

3 DISCUSSION

3.1 Deployment Abstractions for Split Computing

Although end-to-end systems to partition, deploy and monitor DNNs over a hierarchical network already exist, they still lack a layer of abstraction to decouple the runtime system and the intricacies of SC to application developers. Ideally, we could provide a framework that allows developers to implement their DNNs in a manner that is agnostic towards its deployment.

To this end, we propose combining SC with the Function-as-a-Service (FaaS) paradigm by introducing serverless abstractions (Aslanpour et al., 2021). The idea is for application developers to organise DNNs into blocks such that a scheduler can deploy the functions with all or only a subset of the blocks. Then, the input is processed by a series of recursive function calls, i.e. each call executes one block taking the result of the previous block as its input. This approach has two advantages: First, the serverless abstraction seamlessly combines LC, EC and CC, i.e. developers can implement their DNNs agnostic towards where each block ends up being deployed or executed. Second, splitting DNNs becomes *completely dynamic*. For each input, at each step, the runtime can decide whether it executes the next step of the recursion at the current machine or passes it to another. Figure 1 shows a concept sketch of our planned work. A DNN with *n* blocks is deployed as a function, and each function call executes the next unprocessed block, starting at block 1 up on the client-side. Then,

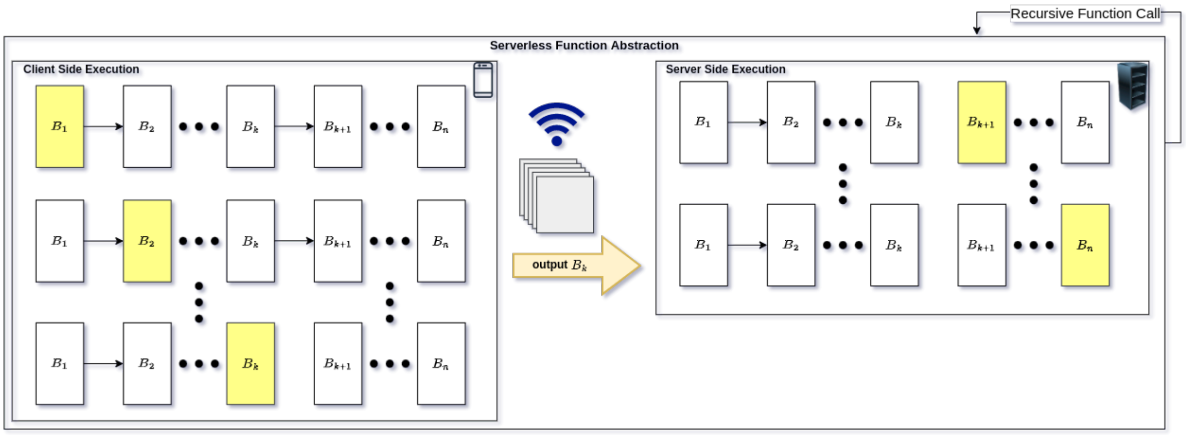


Figure 1: Concept sketch of SC with recursive function calls, where the runtime decides to offload at block B_{n-k}.

the runtime determines that block k is the right place to offload the features, and the execution continues at block $k+1$ on the server-side. In the following, we describe two planned research directions based on the serverless abstraction.

3.1.1 Dynamic Width and Depth for Split Neural Networks

As stated in Section 2.2.2, no substantial work combines SC approaches with dynamic neural networks other than intermediate classifiers for early exiting. Since skipping particular layers and channels can shift the ideal moment to offload, there is a polarisation between most methods which decide the correct splitting point during an offline profiling step. However, with our proposed abstraction, the system can choose to offload intermediate features depending on the current task and not just based on static parameters, naturally bridging SC to DNNs with dynamic width and depth.

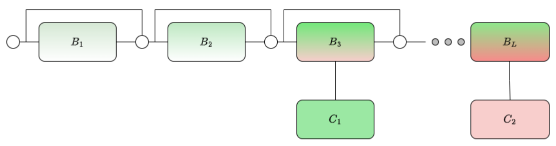
To emphasise this advantage, consider two DNNs with identical architectures, except where one is static and the other is dynamic. For the former, the decision to transfer intermediate features is statically parameterised by the network conditions, i.e. the task must be processed by a fixed number of blocks. The latter is further parameterised by the intrinsic properties of the input, where the gating module variably reduces its size, such that the intermediate features are transferable considerably earlier.

3.1.2 Split Dynamic Neural Networks with Intermediate Classifiers

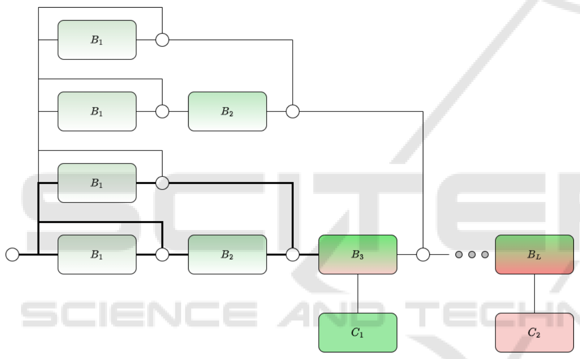
The proposed abstraction is trivially compatible with EE approaches described in Section 2.2.1. At each step of the recursion, the results of an intermediate classifier can trigger a decision for the function to terminate or continue processing. However, we are also interested in investigating their compatibility with layer and channel skipping approaches. Our interest stems from the findings of Veit et al., who have shown that residual networks exhibit ensemble-like behaviour, i.e. we can view networks with residual connections as a collection of paths rather than a single deep network. The implication is that the paths do not depend on each other despite being jointly trained (Veit et al., 2016).

Each block is responsible for extracting different features with varying relevance to the inputs, which *does not necessarily coincide with block depth*. Such findings suggest that the underpinning assumption of EE is flawed. Most work considers strictly sequential dependency of blocks rather than how each input depends on different activation of non-sequential paths. Therefore, intermediate classifiers will needlessly execute some earlier blocks but miss out on potentially necessary features they could have gotten from a later block. Specifically, we hypothesise that a network capable of executing only the necessary blocks for inputs should be able to integrate intermediate classifiers more efficiently than conventional sequential networks.

Most EE methods discard the computed features from a branch if the intermediate classifier's confidence is below a statically configured threshold value, incurring needless overheads for samples that branches cannot classify. If our hypothesis is correct, then a network which can dynamically select the most relevant paths of each input should achieve higher accuracy at the intermediate classifier. Additionally, it should minimise redundant execution by only considering branches for paths which involve a specific number or combination of blocks. For instance, consider a branchy network with L blocks and N (intermediate) classifiers. Figure 2a shows a regular view of a network with residual connections, which only considers sequential processing of the blocks. Current methods will always execute C_1 and toss away the results if the confidence is below a set threshold. Contrarily, Figure 2b shows an unravelled view of the same network, with 2^n implicit paths, i.e. each block doubles the number of possible paths. For brevity, the figure only shows the 2^3 paths of the first three blocks, with one intermediate classifier placed at block 3. With this view, it is possible to configure the network only to execute C_1 if the bolded path processes the input.



(a) Regular view on a branchy network.



(b) Unraveled view on the same network.

Figure 2: Modified figure from (Veit et al., 2016) to include branches.

3.2 Computational Reuse with Intermediate Features

The idea of computational reuse is to exploit the spatio-temporal locality of inputs, i.e. nearby during the same time interval are highly likely to send similar requests. For instance, consider a cognitive assistance application that provides descriptions of observed objects where multiple clients perform similar observations, such as viewing the same animal from a different angle. Then, we can eliminate redundant computation by executing the tasks once by storing them in a cache for reuse. It is out of the scope of this work to give a more detailed explanation of edge caching for computational reuse. We recommend the recent work by Al Azad & Mastorakis, where they provide a brief overview of existing methods (Al Azad and Mastorakis, 2022). Here, we focus on reusing the computation of inference-related tasks and their challenges. Traditional computation elimination methods rely on trivial exact matching methods, such as URIs for cached web content or hash keys for content-addressable storage. Conversely, input images are virtually never identical. Instead, we leverage that semantically correlating images map to the same output (Guo et al., 2018). Therefore, we require approximate methods which are non-trivial for several reasons. For one, the methods must find the similarity among high-dimensional input that matches them according to their correct classification result. Additionally, the method must be fast enough for a cache miss to not noticeably increase the perceived latency and scale for sufficiently large cache size. Moreover, due to the approximate nature of the method, we need to consider that cache misses can happen even if a suitable object is present. Lastly, for similar reasons, we must deal with false cache hits, i.e. when the input has a different label than the objects found in the cache. In other words, when we conceive a solution, we need to leverage trade-offs between cache size, latency, sensitivity and precision.

Current methods only consider non-learned feature extractors combined with simple classifiers or decouple computing of the matching key and the actual task.

We identify two problems with such approaches. First, tossing away intermediate features is computational waste, leading to performance degradation under conditions that lead to increased cache misses. Second, hand-crafted feature extractors cannot be fine-tuned towards the distribution of the expected input.

To solve the abovementioned problems, we propose a method that utilises the intermediate features of a split DNN as the cache key to eliminating computational waste. The work by Venugopal et al. shows promising results with a semantic cache based on neural feature extractors but decouples it from the actual classification model (Venugopal et al., 2018). Moreover, we hypothesise that by utilising current state-of-the-art HDN approaches (Section 2.1.1), it is possible to create significantly more space-efficient and reliable query keys. Specifically, we can exploit the properties of the head model to reduce the dimensionality of input data while maximising the retention of relevant information for the current task. In short, by leveraging the properties of DNNs with bottleneck injection, we can produce keys with minimal noise and negligible overhead. Figure 3 describes the architecture of the planned system.

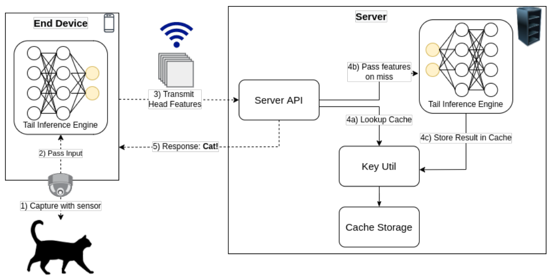


Figure 3: Conceptual system architecture of an inference execution engine with a cache for computational reuse. Step 4 b) and 4 c) are only performed on a cache miss.

3.3 Defensive Mechanisms for Split Computing

There is an inherent risk with exposing sensitive data to third parties. Even trustworthy providers are susceptible to malicious attacks or accidental leakage through negligence. Section 2.3 presented work demonstrating how reducing exposure of sensitive data by allowing the server only to store and process intermediate input representations is not a satisfactory solution. However, all of the introduced attacks were targeted toward unmodified models, which do not represent the current state-of-the-art methods in SC. Since most SC approaches focus on simplifying the head model and compressing the intermediate features by minimising redundant information, we suspect the success of most inversion attacks. Nevertheless, to evaluate the efficacy of countermeasurements, it is essential to create a proper threat assessment by first carrying out input inversion attacks on SC models introduced in recent work. Once we have assessed the threat level of inversion attacks, we can start conceiving solutions tailored toward SC models. The most promising defensive mechanisms suggest that stochastic methods introduce noise on either the model parameters or intermediate features. Therefore, we propose introducing a stochastic layer at the artificial bottleneck. Specifically, we aim to conceive a method which injects the artificial bottleneck based on *variational autoencoders* (Kingma and Welling, 2013) instead of conventional ones.

We base our method on the observation that most inversion attacks use a loss function that minimises the distance between the features produced by the target model and the inverse model. With a stochastic layer, the decoder reconstructs the input by sampling from a distribution, i.e. adversaries cannot rely on creating models that aim to learn weights for inversing deterministic functions. Moreover, we predict that introducing a stochastic layer in existing SC models should only incur a negligible accuracy penalty. However, we cannot apply the same training methods since training variational bottlenecks is less straightforward.

4 CONCLUSION

Traditional methods of compressing models or task offloading are bound to unacceptable trade-offs. To satisfy the requirements of critical AI applications, we have summarised existing work on promising methods to bridge the gap between isolated computing paradigms, and maximise network efficiency. Additionally, we have introduced a series of planned research that leverage the presented literature. We do not claim that completing our proposed methods is sufficient to overcome all the challenges we have identified in this work. However, we believe that successfully implementing the planned system, is a considerable step toward conceiving an end-to-end framework for demanding applications which cannot afford compromises.

REFERENCES

Al Azad, M. W. and Mastorakis, S. (2022). Reservoir: Named data for pervasive computation reuse at the network edge. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 141–151. IEEE.

Aslanpour, M. S., Toosi, A. N., Cicconetti, C., Javadi, B., Sbarski, P., Taibi, D., Assuncao, M., Gill, S. S., Gaire, R., and Dustdar, S. (2021). Serverless edge computing: Vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, ACSW '21, New York, NY, USA. Association for Computing Machinery.

Chiang, C.-H., Liu, P., Wang, D.-W., Hong, D.-Y., and Wu, J.-J. (2021). Optimal branch location for cost-effective inference on branchynet. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5071–5080. IEEE.

Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532.

Eshratifar, A. E., Esmaili, A., and Pedram, M. (2019). Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE.

Guo, P., Hu, B., Li, R., and Hu, W. (2018). Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th annual international conference on mobile computing and networking*, pages 19–34.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. (2021). Dynamic neural networks: A survey.

IEEE Transactions on Pattern Analysis and Machine Intelligence.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, Z., Zhang, T., and Lee, R. B. (2019). Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162.

Herrmann, C., Bowen, R. S., and Zabih, R. (2020). Channel selection using gumbel softmax. In *European Conference on Computer Vision*, pages 241–257. Springer.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jeong, H.-J., Lee, H.-J., Shin, C. H., and Moon, S.-M. (2018). Ionn: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 401–411.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Laskaridis, S., Venieris, S. I., Almeida, M., Leontiadis, I., and Lane, N. D. (2020). Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–15.

Li, E., Zeng, L., Zhou, Z., and Chen, X. (2019). Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457.

Matsubara, Y., Baidya, S., Callegaro, D., Levorato, M., and Singh, S. (2019). Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26.

Matsubara, Y., Callegaro, D., Singh, S., Levorato, M., and Restuccia, F. (2022a). Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. *arXiv preprint arXiv:2201.02693*.

Matsubara, Y., Levorato, M., and Restuccia, F. (2021). Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys (CSUR)*.

Matsubara, Y., Yang, R., Levorato, M., and Mandt, S. (2022b). Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2685–2695.

Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Van Gool, L. (2018). Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402.

Oh, H. and Lee, Y. (2019). Exploring image reconstruction attack in deep learning computation offloading. *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications - EMDL '19*.

Pacheco, R. G., Couto, R. S., and Simeone, O. (2021). Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

Shao, J. and Zhang, J. (2020). Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE.

Teerapittayanon, S., McDanel, B., and Kung, H.-T. (2017). Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 328–339. IEEE.

Veit, A. and Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18.

Veit, A., Wilber, M. J., and Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29.

Venugopal, S., Gazzetti, M., Gkoufas, Y., and Katrinis, K. (2018). Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

Wu, M., Ye, D., Zhang, C., and Yu, R. (2021). Spears and shields: attacking and defending deep model co-inference in vehicular crowdsensing networks. *EURASIP Journal on Advances in Signal Processing*, 2021(1):1–21.

Xia, W., Yin, H., Dai, X., and Jha, N. K. (2021). Fully dynamic inference with deep neural networks. *IEEE Transactions on Emerging Topics in Computing*.

Yao, S., Li, J., Liu, D., Wang, T., Liu, S., Shao, H., and Abdelzaher, T. (2020). Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 476–488.

Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., and Zhang, J. (2019). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762.