This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

FrankenSplit: Efficient Neural Feature Compression with Shallow Variational Bottleneck Injection for Mobile Edge Computing

Alireza Furtuanpey, Philipp Raith, Schahram Dustdar, *Fellow, IEEE*

Abstract—The rise of mobile AI accelerators allows latency-sensitive applications to execute lightweight Deep Neural Networks (DNNs) on the client side. However, critical applications require powerful models that edge devices cannot host and must therefore offload requests, where the high-dimensional data will compete for limited bandwidth. Split Computing (SC) alleviates resource inefficiency by partitioning DNN layers across devices, but current methods are overly specific and only marginally reduce bandwidth consumption. This work proposes shifting away from focusing on executing shallow layers of partitioned DNNs. Instead, it advocates concentrating the local resources on variational compression optimized for machine interpretability. We introduce a novel framework for resource-conscious compression models and extensively evaluate our method in an environment reflecting the asymmetric resource distribution between edge devices and servers. Our method achieves 60% lower bitrate than a state-of-the-art SC method without decreasing accuracy and is up to 16x faster than offloading with existing codec standards.

Index Terms—Split Computing, Distributed Inference, Edge Computing, Edge Intelligence, Learned Image Compression, Data Compression, Neural Data Compression, Feature Compression, Knowledge Distillation

1 Introduction

Deep Learning (DL) has demonstrated that it can solve real-world problems in challenging areas ranging from Computer Vision (CV) [1] to Natural language Processing (NLP) [2]. Complementary with the advancements in mobile edge computing (MEC) [3] and energy-efficient AI accelerators, visions of intelligent city-scale platforms for critical applications, such as mobile augmented reality (MAR) [4], disaster warning [5], or facilities management [6], seem progressively feasible. Nevertheless, the accelerating pervasiveness of mobile clients gave unprecedented growth in Machine-to-Machine (M2M) communication [7], leading to an insurmountable amount of network traffic. A root cause is the intrinsic limitation of mobile devices that allows them to realistically host a single lightweight Deep Neural Network (DNN) in memory at a time. Local resources cannot meet the demanding requirements of applications that rely on multiple highly accurate DNNs [8], [9]. Hence, clients must frequently offload inference requests [10].

The downside to offloading is that by constantly streaming high-dimensional visual data, the limited bandwidth will inevitably lead to network congestion, resulting in erratic response delays, and it leaves valuable client-side resources idle.

Split Computing (SC) emerged as an alternative to alleviate inefficient resource utilization and to facilitate low-latency and performance-critical mobile inference. The basic idea is to partition a DNN to process the shallow layers with the client and send a processed representation to the remaining deeper layers deployed on a server. The SC paradigm can potentially draw resources from the entire edge-cloud compute continuum. However, current SC methods are only conditionally applicable (e.g., in highly bandwidth-constrained networks) or tailored toward specific neural network architectures. Methods that claim to generalize towards a broader range of architectures do not consider that mobile clients can typically only load a single model into memory. Consequently, SC methods are impractical for applications with complex requirements relying on inference from multiple models concurrently (e.g., MAR). Mobile clients reloading weights from its storage into memory and sending multiple intermediate representations for each pruned model would incur more overhead than directly transmitting image data with fast lossless codecs. Moreover, due to the conditional applicability of SC, practical methods rely on a decision mechanism that periodically probes external conditions (e.g., available bandwidth), resulting in further deployment and runtime complexity [11].

This work shows that we can address the increasing need to reduce bandwidth consumption while simultaneously generalizing the objective of SC methods to provide mobile clients access to low-latency inference from remote off-the-shelf discriminative models even in constrained networks.

We draw from recent advancements in lossy learned image compression (LIC) and the Information Bottleneck (IB) principle [12]. Despite outperforming handcrafted codecs [13], such as PNG, or WebP [14], LIC is unsuitable for real-time inference in MEC since they consist of large models and other complex mechanisms that are demanding even for server-grade hardware. Further, research in compression primarily focuses on reconstruction for human perception containing information superfluous for M2M communication. In comparison, the deep variational information bottleneck (DVIB) provides an objective for learned

feature compression with DNNs, prioritizing information valuable for machine interpretability.

With DVIB, we can conceive generalizable methods that are applicable to off-the-shelf architectures. However, current DVIB approaches typically place the bottleneck at the penultimate layer. Thus, they are unsuitable for most common settings that assume an asymmetric resource allocation between the client and the server. In other words, the objectives of DVIB and MEC contradict each other, i.e., for the latter, we require shifting the bottleneck's location to the shallow layers.

We accommodate the restrictions of mobile clients by introducing a method that moves the bottleneck to the shallow layers and retains generalizability to arbitrary architectures. While shifting the bottleneck does not formally change the objective, we will demonstrate that existing methods for mutual information estimation lead to unsatisfactory results.

To this end, we introduce *FrankenSplit*: A novel training and design heuristic for variational feature compression models embeddable in arbitrary DNN architectures with pre-trained weights for high-level vision tasks. FrankenSplit is refreshingly simple to implement and deploy without additional decision mechanisms that rely on runtime components for probing external conditions. Additionally, by deploying a single lightweight encoder, the client can access state-of-the-art accuracy from multiple large server-grade models without reloading weights from memory for each task. Lastly, the approach does not require modifying discriminative models (e.g., by finetuning weights). Therefore, we can directly utilize foundational off-the-shelf models and seamlessly integrate FrankenSplit into existing systems.

We open-source our repository [1] as an addition to the community for researchers to reproduce and extend our experiments. In summary, our contributions are:

- Thoroughly exploring how shallow and deep bottleneck injection differ for feature compression.
- Introducing a novel saliency-guided training method to overcome the challenges of training a lightweight encoder with limited capacity to compress features usable for several downstream tasks.
- Introducing a generalizable design heuristic for embedding a variational feature compression model into arbitrary DNN architectures.

Section 2 discusses relevant work on SC and LIC. Section 3 discusses the limitations of SC methods and motivates neural feature compression. Section 4 describes the problem domain. Section 5 progressively introduces the solution approach. Section 6 extensively justifies relevant performance indicators and evaluates several implementations of FrankenSplit against various baselines to assess our method's efficacy. Lastly, Section 7 summarizes this work and highlights limitations to motivate follow-up work.

1. https://github.com/rezafuru/FrankenSplit

2 Related Work

2.1 Neural Data Compression

2.1.1 Learned Image Compression

The goal of (lossy) image compression is minimizing bitrates while preserving information critical for human perception. Transform coding is a basic framework of lossy compression, which divides the compression task into decorrelation and quantization [15]. Decorrelation reduces the statistical dependencies of the pixels, allowing for more effective entropy coding, while quantization represents the values as a finite set of integers. The core difference between handcrafted and learned methods is that the former relies on linear transformations based on expert knowledge. Contrarily, the latter is data-driven with non-linear transformations learned by neural networks [16].

Ballé et al. introduced the Factorized Prior (FP) entropy model and formulated the neural compression problem by finding a representation with minimal entropy [17]. An encoder network transforms the original input to a latent variable, capturing the input's statistical dependencies. In follow-up work, Ballé et al. [18] and Minnen et al. [19] extend the FP entropy model by including a hyperprior as side information for the prior. Minnen et al. [19] introduce the joint hierarchical priors and autoregressive entropy model (JHAP), which adds a context model to the existing scale hyperprior latent variable models. Typically, context models are lightweight, i.e., they add a negligible number of parameters, but their sequential processing increases the end-to-end latency by orders of magnitude.

2.1.2 Feature Compression

Singh et al. demonstrate a practical method for the Information Bottleneck principle in a compression framework by introducing the bottleneck in the penultimate layer and replacing the distortion loss with the cross-entropy for image classification [20]. Dubois et al. generalized the VIB for multiple downstream tasks and were the first to describe the feature compression task formally [21]. However, their encoder-only CLIP compressor has over 87 million parameters. Both Dubois and Singh et al. consider feature compression for mass storage, i.e., they assume the data is already present at the target server. In contrast, we consider how resource-constrained clients must first compress the high-dimensional visual data before sending it over a network.

Closest to our work is the Entropic Student (ES) proposed by Matsubara et al. [22], [23], as we follow the same objective of real-time inference with feature compression. Nevertheless, they simply apply the learning objective and a scaled-down version of autoencoder from [17], [18]. Contrastingly, we carefully examine the problem domain of resource-conscious feature compression to identify underlying issues with current methods, allowing us to conceive novel solutions with significantly better rate-distortion performance.

2.2 Split Computing

We distinguish between two orthogonal approaches to SC.

2.2.1 Split Runtimes

Split runtime systems are characterized by performing no or minimal modifications on off-the-shelf DNNs. The objective is to dynamically determine split points according to the available resources, network conditions, and intrinsic model properties. Hence, split runtimes primarily focus on profilers and adaptive schedulers. Kang et al. performed extensive compute cost and feature size analysis on the layer-level characterizations of DNNs and introduced the first split runtime system [24]. Their study has shown that split runtimes are only sensible for DNNs with an early natural bottleneck, i.e., models performing aggressive dimensionality reduction within the shallow layers. However, most modern DNNs increase feature dimensions until the last layers for better representation. Consequently, follow-up work focuses on feature tensor manipulation [25]–[27]. We argue against split runtimes since they introduce considerable complexity. Worse, the system must be tuned toward external conditions, with extensive profiling and careful calibration. Additionally, runtimes raise overhead and another point of failure by hosting a network-spanning system. Notably, even the most sophisticated methods still rely on a natural bottleneck, evidenced by how state-of-the-art split runtimes still report results on superseded DNNs with an early bottleneck [28], [29].

2.2.2 Artificial Bottleneck Injection

By shifting the effort towards modifying and re-training an existing base model (backbone) to replace the shallow layers with an artificial bottleneck, bottleneck injection retains the simplicity of offloading. Eshratifar et al. replace the shallow layers of ResNet-50 with a deterministic autoencoder network [30]. A follow-up work by Jiawei Shao and Jun Zhang further considers noisy communication channels [31]. Matsubara et al. [32], and Sbai et al. [33] propose a more general network agnostic knowledge distillation (KD) method for embedding autoencoders, where the output of the split point from the unmodified backbone serves as a teacher. Lastly, we consider the work in [22] as the state-of-the-art for bottleneck injection.

Although bottleneck injection is promising, there are two problems with current methods. They rely on deterministic autoencoders for a crude approximation to compression or are intended for a specific class of neural network architecture.

This work addresses both limitations of such bottleneck injection methods.

3 The Case for Neural Data Compression

We assume an asymmetric resource allocation between the client and the server, i.e., the latter has considerably higher computational capacity. Additionally, we consider large models for state-of-the-art performance of non-trivial discriminative tasks unsuitable for mobile clients. Progress in energy-efficient ASICs and embedded AI with model compression with quantization, channel pruning, etc., permit constrained clients to execute lightweight DNNs. Nevertheless, they are bound to reduced predictive strength relative to their contemporary unconstrained counterparts [34]. This assumption is sensible considering the trend for DNNs towards pre-trained foundational models with rising computational requirements due to increasing model sizes [35] and costly operations [36].

Lastly, mobile devices cannot realistically load weights for multiple models simultaneously [9], and it is unreasonable to expect that a single compressed model is sufficient for applications with complex requirements that rely on various models concurrently or in quick succession.

Conclusively, despite the wide availability of onboard accelerators, the demand for large models to solve intelligent tasks will further increase, transmitting large volumes of high-dimensional data. The claim is consistent with CISCO's report that emphasizes the accelerating bandwidth consumption by M2M communication [7].

3.1 Limitations of Split Computing

Still, it would be valuable to leverage advancements in energy-efficient mobile chips beyond applications where local inference is sufficient. In particular, SC can potentially draw resources from an entire edge-cloud compute continuum while binary on- or offloading decision mechanisms will leave valuable client or server-side resources idle. Figure 1 illustrates generic on/offloading and split runtimes. The caveat is that both SC approaches discussed

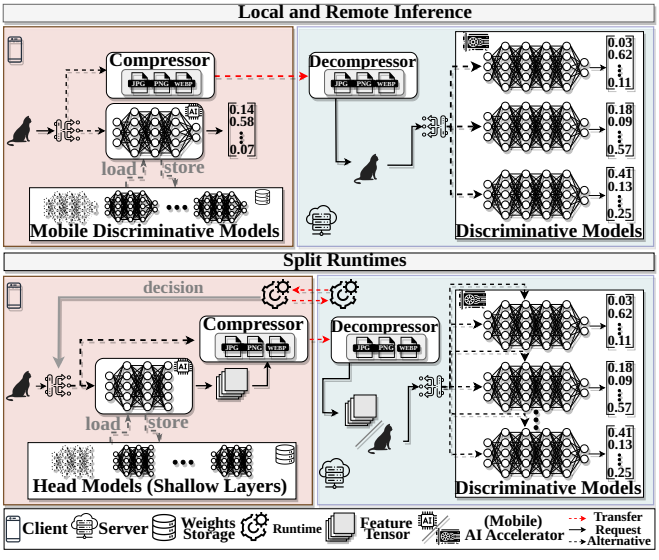

Fig. 1: Prediction with on/offloading and split runtimes

in Section 2.2 are only conditionally applicable. In particular, split runtimes reduce server-side computation for inference tasks with off-the-shelf models by onloading and executing shallow layers at the client. This approach introduces two major limitations.

First, when the latency is crucial, this is only sensible if the time for client-slide execution, transferring the features, and remotely executing the remaining layers is less than the time of directly offloading the task. More recent work [27]–[29] relies on carefully calibrated dynamic decision mechanisms. A runtime component periodically measures (e.g., network bandwidth) and internal conditions (e.g., client load) to measure ideal split points or whether direct offloading is preferable.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

Second, since the shallow layers must match the deeper layers, split runtimes cannot accommodate applications with complex requirements, which is a common justification for MEC (e.g., MAR). Constrained clients would need to swap weights from the storage in memory each time the prediction model changes. Worse, the layers must match even for models predicting the same classes with closely related architectures.

Hence, it is particularly challenging to integrate split runtimes into systems that can increase the resource efficiency of servers by adapting to shifting and fluctuating environments [37], [38]. For example, when a client specifies a target accuracy and a tolerable lower bound, the system could select a ResNet-101 that can hit the target accuracy but may temporarily fall back to a ResNet-50 to ease the load when necessary.

3.2 Execution Times with Resource Asymmetry

Table 1 summarizes the results of a simple experiment to demonstrate limitations incurred by resource asymmetry. The client is an Nvidia Jetson NX2 equipped with an AI accelerator, and the server hosts an RTX 3090 (see Section 6 for details on hardware configurations). We measure the execution times of ResNet variants, classifying a single $3 \times 224 \times 224$ tensor at two split points.

TABLE 1: Execution Times of Split Models

Model	Split Index	Head [NX2] (ms)	Head [3090] (ms)	Tail [3090] (ms)	Rel. Exec. [NX2] (%)	Contribution [NX2] (%)
ResNet-50	Stem	1.5055	0.1024	4.9687	23.25	0.037
	Stage 1	8.2628	0.9074	4.0224	67.26	0.882
ResNet-101	Stem	1.5055	0.1024	9.8735	13.23	0.021
	Stage 1	8.2628	0.9074	8.9846	47.91	0.506
ResNet-152	Stem	1.5055	0.1024	14.8862	9.18	0.015
	Stage 1	8.2628	0.9074	13.8687	37.34	0.374

Similar to other widespread architectural families, ResNets organize their layers into four top-level layers, and the top-grained ones recursively consist of finer-grained ones. While the terminology differs for architectures, we will uniformly refer to top-level layers as *stages* and the coarse-grained layers as *blocks*.

Split point *stem* assigns the first preliminary block as the head model. It consists of a convolutional layer with batch normalization [39] and ReLU activation, followed by max pooling. Split point *Stage 1* additionally assigns the first stage to the head. Notice how the shallow layers barely constitute the overall computation, even when the client takes more time to execute the head than the server for the entire model. Further, compare the percentage of total computation time and relate them to the number of parameters. At best, the client contributes to 0.02% of the model execution when taking 9% of the total computation time and may only contribute 0.9% when taking 67% off the computation time.

Despite a powerful AI accelerator, it is evident that utilizing client-side resources to aid a server is inefficient. Consequently, SC methods commonly include some form of quantization and data size reduction to offset resource asymmetry. In the following, we conceive a hypothetical SC method to provide intuition behind the importance of reducing transfer costs.

3.3 Feature Tensor Dimensionality and Quantization

Typically, most work starts with some statistical analysis of the output layer dimensions, as illustrated in Figure 2. Excluding repeating blocks, the feature dimensionality is identical for ResNet-50, -101, and -152. The red line marks the cutoff where the size of the intermediate feature tensor is less than the original input. ResNets (including more modern variants [40]), among numerous recent architectures [35], [36], do not have an early natural bottleneck and will only drop below the cutoff from the first block of the second stage (S3RB1-2). Since executing until S3RB1-2 is only about 0.06% Modern methods reduce the number of layers a client must execute with feature tensor quantization and other clever (typically statistical) methods that statically or dynamically prune channels [11]. For our hypothetical

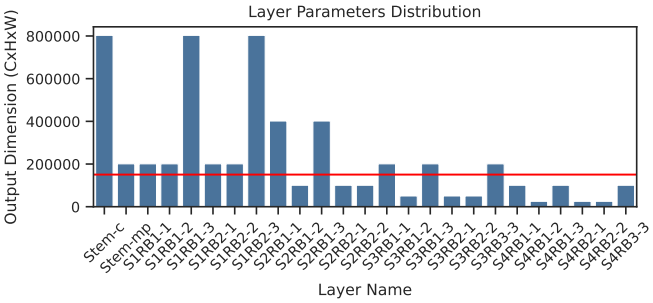

Fig. 2: Output dimensionality distribution for ResNet

method, we use the execution times from Table 1. We generously assume that the method applies feature tensor quantization and channel pruning to reduce the expected data size without a loss in accuracy for the *ImageNet* classification task [41] and with no computational costs. Further, we reward the client for executing deeper layers to reflect deterministic bottleneck injection methods, such that the output size of the stem and stage one are 802816 and 428168 bits, respectively. Note that, for stage one, this is roughly a 92% reduction relative to its original FP32 output size. Yet, the plots in Figure 3 show that offloading with PNG, let alone more modern lossless codecs (e.g., WebP), will beat SC in total request time, except when the data rate is severely constrained. Evidently, using reasonably powerful

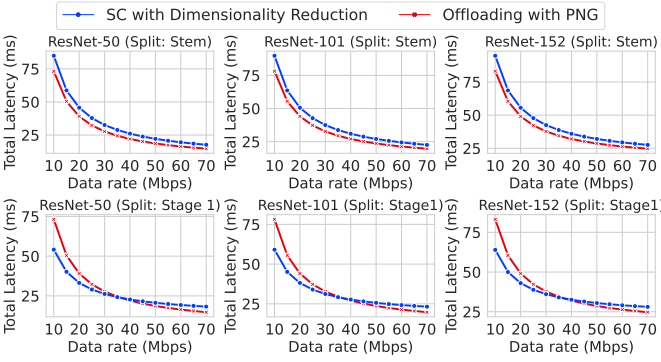

Fig. 3: Inference latency for SC and offloading

AI accelerators to execute the shallow layers of a target model is not an efficient use of client-side resources.

3.4 The advantage of learned methods

In a narrow sense, more modern work on SC considers minimizing transmitting data with feature tensor quantization and other clever (typically statistical) methods that statically or dynamically prune channels.

While dimensionality reduction can be seen as a crude approximation to compression, it is not equivalent to it [19]. Compression aims to reduce the entropy of the latent under a prior shared between the sender and the receiver [16]. Dimensionality reduction (especially channel pruning) may seem effective for simple tasks (e.g., CIFAR-10 [42]). However, this is more due to the overparameterization of large DNNs. Precisely, for a simple task, we can prune most channels or inject a small autoencoder at the shallow layers that may appear to achieve unprecedented compression rates relative to the unmodified head's feature tensor size. In Section 6.3.9, we will show that methods working reasonably well on a simple dataset can ultimately falter on more challenging datasets.

From an information-theoretic point of view [43], tensor dimensionality is not an adequate measure (i.e., $C \times H \times W \times$ Precision) to determine transfer costs. Instead, we should consider the entropy of the feature tensor [43]. Then, we can optimize a model to reduce uncertainty and compress an input according to its information content.

To summarize, the potential of SC is inhibited by primarily focusing on shifting parts of the model execution from the server to the client. SC's viability is not determined by how well they can partially compute a split network but by how well they can reduce the input size. Therefore, we pose the following question: *Is it more efficient to focus the local resources* ***exclusively*** *on compressing the data rather than executing shallow layers of a network that would constitute a negligible amount of the total computation cost on the server?*

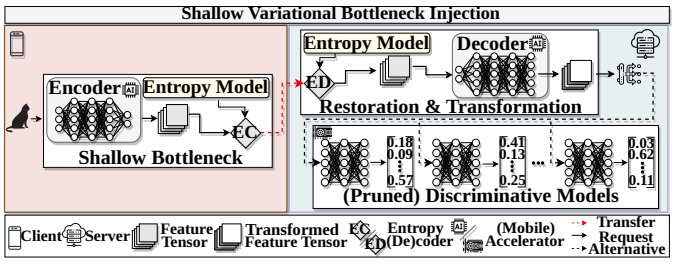

Fig. 4: Prediction with Variational Bottleneck Injection

In Figure 4, we sketch predictions with our proposed approach. There are two underlying distinctions to common SC methods.

First, the model is not split between the client and the server. Instead, it deploys a lightweight encoder, and a decoder replaces the shallow layers of a backbone, i.e., the backbone is split within the server. A single decoder architecture corresponds to backbones with related architectures. Notably, a decoder restores and transforms the compressed signals to a backbone that may accommodate multiple tasks. The encoder is decoupled from a particular task and the decoder-backbone pair. Section 5.3 elaborates how separating the concerns permits one encoder instance to accommodate multiple decoder-backbone pairs.

Second, compared to split runtimes, the decision to apply the compression model may only depend on *internal conditions*. It can decouple the client from any external component (e.g., server, router). Ideally, applying the encoder should always be preferable if a mobile device has the minimal required resources. Since our method does not alter the backbones, we do not need to maintain additional models to accommodate clients who cannot apply the encoder. Instead, we can simply route the image tensor to the input layer of the (unmodified) model.

The following describes the limitations of existing work for constrained devices to conceive a method with the abovementioned description.

4 Problem Formulation

The goal is for constrained clients to request real-time predictions from a large DNN while maximizing resource efficiency and minimizing bandwidth consumption with compression methods. Figure 5 illustrates the possible approaches when dedicating client resources exclusively for compression. Strategy a) corresponds to offloading strategies with CPU-bound handcrafted codecs. Strategy b) represents recent LIC models. Learned methods can achieve considerably lower bitrates with comparable distortion than commonly used handcrafted codecs [16]. Nevertheless, we must consider that the overhead of executing large DNNs may dominate the reduced transfer time. Strategy c) is our advocated method with an embeddable variational feature compression that draws from the same underlying Nonlinear Transform Coding (NTC) framework as b). The challenge is to reduce overhead to make variational compression models suitable for real-time prediction with limited client resources.

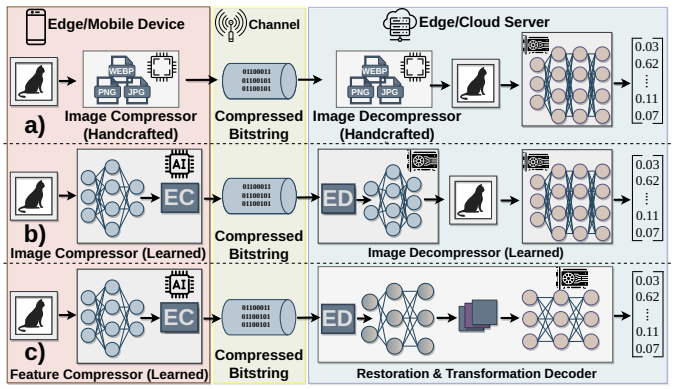

Fig. 5: Utilizing client resources with (learned) codecs

To overcome the limitations of existing methods, we require (i) a resource-conscious encoder design. The encoder should minimize the transfer cost without increasing the predictive loss. Additionally, (ii) the decoder should exploit the available server-side resources without incurring significant overhead. Lastly, (iii) a compression model should fit for different downstream tasks and architectural families (e.g., CNNs or Vision Transformers).

Before we can conceive an adequate method, we must formalize the properties of a suitable objective and elaborate on the limitations of existing methods when applied to shallow bottlenecks.

4.1 Rate-Distortion Theory for Model Prediction

By Shannon's rate-distortion (r-d) theory [44], we seek a mapping bound by a distortion constraint from a random variable (r.v.) X to an r.v. U, minimizing the bitrate of the outcomes of X. More formally, given a distortion measure $\mathcal{D}$ and a distortion constraint D_c, the minimal bitrate is:

$$\min_{P_{U|X}} I(X;U) \text{ s.t. } \mathcal{D}(X,U) \leq D_c \tag{1}$$

where $I(X;U)$ is the mutual information and is defined as

$$\mathrm{I}(X;U) = \int\!\!\int p(x,u) \log\left(\frac{p(x,u)}{p(x)p(u)}\right) dxdu \tag{2}$$

In lossy image compression, U is typically the reconstruction of $\tilde{X}$ of the original input, and the distortion measure is some sum of squared errors $d(x,\tilde{x})$. Since the r-d theory does not restrict us to image reconstruction [45], we can apply distortion measures relevant to M2M communication. Notably, when our objective is to minimize predictive loss rather than reconstructing the input, we keep information that may be excessive for model predictions.

To elaborate on the potential of discarding information for discriminative tasks, consider the Data Processing Inequality (DPI). For any 3 r.v.s X, Y, Z that form a Markov chain $X \leftrightarrow Y \leftrightarrow Z$ where the following holds:

$$I(X;Y) \geq I(X;Z) \tag{3}$$

Then, describe the information flow in an n-layered sequential DNN, layer with the information path by viewing layered neural networks as a Markov chain of successive representations [46]:

$$I(X;Y) \geq I(R_1;Y) \geq I(R_2;Y) \geq \ldots I(R_n;Y) \geq I(\tilde{Y};Y) \tag{4}$$

In other words, the final representation before a prediction R_n cannot have more mutual information with the target than the input X and typically has less. In particular, for high-level vision tasks that map a high dimensional input vector with strong pixel dependencies to a small set of labels, we can expect $I(X;Y) \gg I(\tilde{R_n},Y)$.

4.2 From Deep to Shallow Bottlenecks

When the task is to predict the ground-truth labels Y from a joint distribution $P_{X,Y}$, the r-d objective is essentially given by the information bottleneck principle [12]. By relaxing the (1) with a lagrangian multiplier, the objective is to maximize:

$$I(Z;Y) - \beta I(Z;X) \tag{5}$$

Specifically, an encoding Z should be a minimal sufficient statistic of X respective Y, i.e., we want Z to contain relevant information regarding Y while discarding irrelevant information from X. Practical implementations differ by the target task and how they approximate (5). For example, an approximation of $I(Z;Y)$ for an arbitrary classification task the conditional cross entropy (CE) [13]:

$$\mathcal{D} = H(P_Y, P_{\tilde{Y}|Z}) \tag{6}$$

Using (6) for estimating $I(Z;Y)$ to end-to-end optimize a neural compression model is not a novel idea (Section 2.1.2). However, a common assumption in such work is that the latent variable is the final representation R_n of a large backbone, which we refer to as *Deep Variational Information Bottleneck Injection (DVBI)*. Conversely, we work with resource-constrained clients, i.e., to conceive lightweight encoders, we must shift the bottleneck to the shallow layers, which we refer to as *Shallow Variational Bottleneck Injection (SVBI)*. Intuitively, the existing methods for DVBI should generalize to SVBI, e.g., estimate the distortion term with (6) as in [20].

While shifting the bottleneck to the shallow layers results in an encoder with less capacity, the objective still approximates to (1). Yet, as we will show in Section 6.3.9, applying the objective from [20] will result in incomparably worse results when moving the bottleneck to the shallow bottlenecks.

A more promising method to estimate $I(Z;Y)$ is Head Distillation (HD) [32], [33] since it naturally aligns with shallow bottlenecks. As we will show in Section 6.3, HD yields significantly better results than applying (6). Surprisingly, despite showing promising results, HD is a suboptimal estimation for $I(Z;X)$ to approximate eq. (1).

The following elaborates on SVBI and formulates the VIB objective for HD.

4.3 Head Distilled Deep Variational IB

Ideally, the bottleneck is embeddable in an existing predictor $P_{\mathcal{T}}$ without decreasing the performance. Therefore, it is not the hard labels Y that define the task but the soft labels $Y_{\mathcal{T}}$. For simplicity, we handle the case for one task and defer discussion on multiple downstream tasks and DNNs to Section 5.3.

To perform SVBI, take a copy of $P_{\mathcal{T}}$. Then, mark the location of the bottleneck by separating the copy into a head $\mathcal{P}_h$ and a tail $\mathcal{P}_t$. Importantly, both parts are deterministic, i.e., for every realization of r.v. X there is a representation $\mathcal{P}_h(x) = h$ such that $\mathcal{P}_{\mathcal{T}}(x) = \mathcal{P}_h(\mathcal{P}_t(x))$. Lastly, replace the head with an autoencoder and a parametric entropy model.

The encoder is deployed at the sender, the decoder at the receiver, and the entropy model is shared. We distinguish between two optimization strategies to train the bottleneck's compression model. First, is *direct optimization* corresponding to the DVIB objective in (5), except we replace the CE with the standard KD loss [47] to estimate $I(Z;Y)$. The second is *indirect optimization* and describes HD with the objective:

$$I(Z;H) - \beta\, I(Z;X) \tag{7}$$

Unlike the former, the latter does not directly correspond to (1) for a representation Z that is a minimal sufficient statistic of X respective $Y_{\mathcal{T}}$. Instead, it replaces Y with a proxy task for the compression model to replicate the output of the replaced head, i.e., training methods approximating (7) optimize for a Z that is a minimal sufficient statistic of X respective H. Figure 6 illustrates the difference between estimating the objectives (5) and (7).

With faithful replication of H, the partially modified DNN has an information path equivalent to its unmodified version. A sufficient statistic retains the information necessary to replicate the input for a deterministic tail, i.e., the final prediction does not change. The problem of (7) is that it is a suboptimal approximation of (1). Although sufficiency

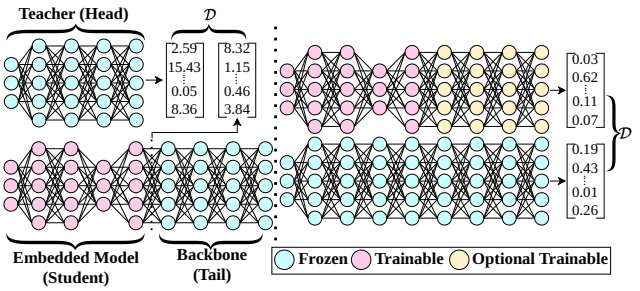

Fig. 6: Left: Head Distillation. Right: Direct Optimization

still holds, it optimizes Z respective H and not $Y_{\mathcal{T}}$ to be minimal.

Based on the above formulations, the following proposes a practical method to train stochastic feature compression models. Additionally, it addresses the limitations of HD and includes architectural considerations.

5 Solution Approach

Our solution focuses on two distinct but intertwined aspects. First is an appropriate training objective. The second concerns a practical implementation by introducing an architectural design heuristic to accommodate backbones with various architectures with a single encoder architecture.

5.1 Loss Function for End-to-end Optimization

We follow NTC [16] to implement a neural compression algorithm. Specifically, we embed a stochastic compression model that we jointly optimize with an entropy model.

Our objective resembles variational image compression optimization, as introduced in [17], [18]. For an image vector x, we have a parametric analysis transform $g_a(x;\phi_g)$ that maps x to a latent vector z. Then, a quantizer Q discretizes z to $\bar{z}$, such that an entropy coder can use the entropy model to losslessly compress $\bar{z}$ to a sequence of bits. In learned image compression, a parametric synthesis transforms $g_s(\bar{z};\theta_g)$ maps $\bar{z}$ to a reconstruction of the input $\tilde{x}$

However, we favor HD over direct optimization as a distortion measure since the former yields considerably better results even with a suboptimal loss function (Section 6.3.9). Therefore, we require a $g_s(\bar{z};\theta_g)$ that maps $\bar{z}$ to an approximation of a representation $\tilde{h}$ (i.e., the output of shallow layers of an arbitrary backbone).

Analogous to variational inference, we approximate the intractable posterior $p(\tilde{z}|x)$ with a parametric variational density $q(\tilde{z}|x)$ as follows (excluding constants):

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} D_{\mathrm{KL}}\left[q \| p_{\tilde{\boldsymbol{z}}|\boldsymbol{x}}\right] = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\mathbb{E}_{\tilde{\boldsymbol{z}}\sim q}[\underbrace{-\log p(x|\tilde{z})}_{\text{distortion}} - \overbrace{\log p(\tilde{z})}^{\text{weighted rate}}] \tag{8}$$

By assuming a Gaussian distribution such that the likelihood of the distortion term is given by

$$P_{x|\tilde{z}}(x \mid \tilde{z}, \theta_g) = \mathcal{N}(x \mid g_s(\tilde{z};\theta_g),\ \mathbf{1}) \tag{9}$$

we can use the square sum of differences between h and $\tilde{h}$ as our distortion loss.

The rate term describes the cost of compressing $\tilde{z}$. Analogous to the LIC methods discussed in Section 2.1, we apply uniform quantization $\bar{z} = \lfloor \tilde{z} \rceil$. Since discretization leads to problems with the gradient flow, we apply a continuous relaxation by adding uniform noise $\eta \sim \mathcal{U}(-\frac{1}{2}, \frac{1}{2})$. Combining the rate and distortion term, we derive the loss function for estimating objective (7) as:

$$\mathcal{L} = \|\mathcal{P}_h(x) \text{ - } (g_s(g_a(x;\phi_g)+\eta;\theta_g)\|_2^2 + \beta \log(g_a(x;\theta_g)+\eta) \tag{10}$$

As described in Section 4.3, by using HD for the distortion term, we rely on H as a proxy target, i.e., the loss in Equation (10) is a suboptimal approximation of (1).

The suboptimality stems from treating every pixel in H equally important. The implication here is that the MSE in (10) overly strictly penalizes pixels at spatial locations that contain redundant information that later layers can safely discard. Contrarily, the loss may not penalize the salient pixels enough when $\tilde{h}$ is numerically close to h.

Hence, we can improve the loss in (10) by introducing additional signals that regularize the suboptimal distortion term. The challenge is finding a tractable method that emphasizes the salient pixels necessary for multiple instances of a high-level vision task (e.g., classification of various datasets and labels). Moreover, the method should exclusively concern the loss function, i.e., it should not introduce any additional model components or operations during inference.

5.2 Saliency Guided Distortion

We consider HD an extreme form of Hint Training (HT) [48], [49] where the hint becomes the primary objective rather than an auxiliary regularization term. Sbai et al. perform deterministic bottleneck injection with HD using the suboptimal distortion term [33]. Nevertheless, their method only considers dimensionality reduction without a parametric entropy model as an approximation to compression, i.e., it is generalized by the loss in (1)($\beta = 0$). Matsubara et al. add further hints from the deeper layers by extending the distortion term with the sum of squared between the deeper layers [23], [32]. This approach has several downsides besides prolonged train time. The distortion term may now dominate the rate term, i.e., without exhaustively tuning the hyperparameters for each distortion term, the optimization algorithm should favor converging towards local optima. Moreover, we show Section 6.3.2 that pure HD can significantly outperform this method using the loss in Equation (1) without the hints from the deeper layers.

In principle, we could improve the performance by extracting signals from deeper layers and directly transferring them to the bottleneck. The caveat is that the effectiveness of knowledge distillation decreases for teachers when the student has considerably less capacity than the teacher [48]. Hence, instead of directly introducing hints at the encoder, we propose regularizing the distortion term with saliency maps.

For each sample, we require a vector $\mathcal{S}$, where each $s_i \in \mathcal{S}$ is a weight term for a spatial location salient about the conditional probability distributions of the remaining

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

tail layers. Then, we should be able to improve the r-d performance by regularizing the distortion term in (10) with

$$\mathcal{L}_{\text{distortion}} = \gamma_1 \cdot \mathcal{L}_1 + \gamma_2 \cdot s_i \cdot \frac{1}{N} \sum_i (h_i - \tilde{h}_i)^2 \qquad (11)$$

Where $\mathcal{L}_1$ is the distortiont term from Equation (10), and γ_1, γ_2 are nonnegative real numbers summing to 1. We default to $\gamma_1 = \gamma_2 = \frac{1}{2}$ in our experiments. Figure 7

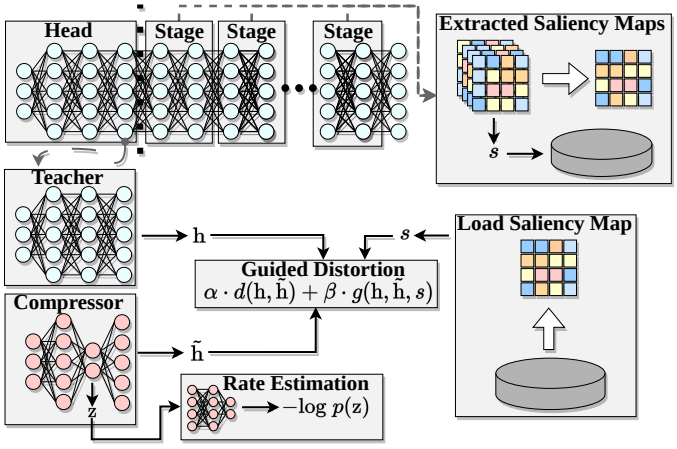

Fig. 7: Training setup

describes our final training setup. Note that we only require computing the saliency maps once, and they are architecturally agnostic towards the encoder.

We derive the saliency maps using *class activation mapping (CAM)* [50]. Although CAMs are typically used to improve the explainability of DNNs, they suit our purposes by allowing us to summarize salient pixel locations. Specifically, we use a variant of Grad-CAM [51] to measure a spatial location's importance at any stage. Figure 8 illustrates some examples of saliency maps when averaged over the deeper backbone stages. In this work, we favor Grad-

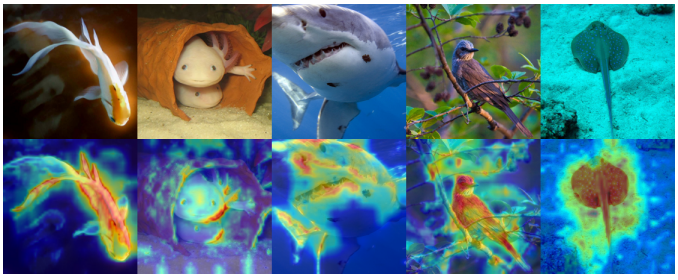

Fig. 8: Extracted saliency maps using Grad-CAM

CAM over (more intricate) methods due to its architecture-agnostic nature and computational efficiency. For example, mixing with guided backpropagation [52] could refine the resulting saliency maps with finer-grained feature importance scaling. However, guided backpropagation relies on specific properties of the activation function and requires adjustments for each architectural family.

5.3 Network Architecture

The beginning of this section broke down our aim into three problems. We addressed the first with SVBI and proposed a novel training method for low-capacity compression models. A generalizable resource-asymmetry-aware autoencoder design remains. Additionally, the encoder should be reusable for several backbones. To not inflate the significance of our contribution, we refrain from including components based on existing work in efficient neural network design.

5.3.1 Model Taxonomy

We introduce a minimal taxonomy described in Figure 9 for our approach. The top-level, *Archtype*, reflects the primary inductive bias of the model. *Architectural families* describe variants (e.g., ResNets such as ResNet [53], Wide ResNet [54], ResNeXt [40], etc.). *Directly related* refers to the same architecture of different sizes (e.g., Swin-T, Swin-S, Swin-B, etc.). The challenge is to conceive a design heuristic

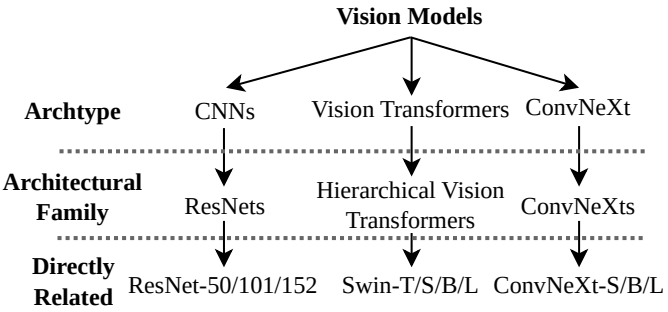

Fig. 9: Simple taxonomy with minimal example

that can exploit the available server resources to aid the lightweight encoder with minimal overhead on the prediction task. First, we concretize shallow features by describing how to locate the layers for bottleneck placement. Then, we derive the heuristic to conceive decoder models for arbitrary architectural families and how to account for client-server resource asymmetry.

Lastly, we describe how to share trained compressor components among directly related architectures.

5.3.2 Bottleneck Location by Stage Depth

Consider how most modern DNNs consist of an initial embedding followed by a few stages (Described in Section 3.1). Within directly related architectures, the individual components are identical. The difference between variants is primarily the embed dimensions or the block ratio of the deepest stage. For example, the block ratio of ResNet-50 is 3:4:**6**:3, while the block ratio of ResNet-101 is 3:4:**23**:3. Consequently, the stage-wise organization of models defines a natural interface for SVBI. For the remainder of this work, we refer to the *shallow layers* as the layers before the deepest stage (i.e., the initial embedding and the first two stages).

5.3.3 Decoder Blueprints

A key characteristic distinguishing archetypes is the inductive bias introduced by basic building blocks (e.g., convolutions versus attention layers). To consider the varying representations among non-related architectures, we should not disregard architecture-induced bias by directly repurposing neural compression models for SC. For example,

a scaled-down version of Ballé et al.'s [17] convolutional neural compression model can yield strong r-d performance for bottlenecks reconstructing a convolutional layer [22]. However, we will show that this does not generalize to other architectural families, such as hierarchical vision transformers [55].

One potential solution is to use identical components for the compression model from a target network. While this may be inconsequential for server-side decoders, it is inadequate for encoders due to the heterogeneity of edge devices. Vendors have varying support for the basic building blocks of a DNN, and particular operations may be prohibitively expensive for the client. Hence, in FrankenSplit, the encoder is fixed, but the decoder is adaptable. Regardless of the decoder architecture, we account for the heterogeneity with a uniform encoder architecture composed of three downsampling residual blocks of two stacked 3×3 convolutions with ReLU non-linearity, totaling around $140,000$ parameters. We handle the varying representations by introducing *decoder blueprints* tailored towards an architectural family, i.e., one blueprint corresponds to all directly related architectures.

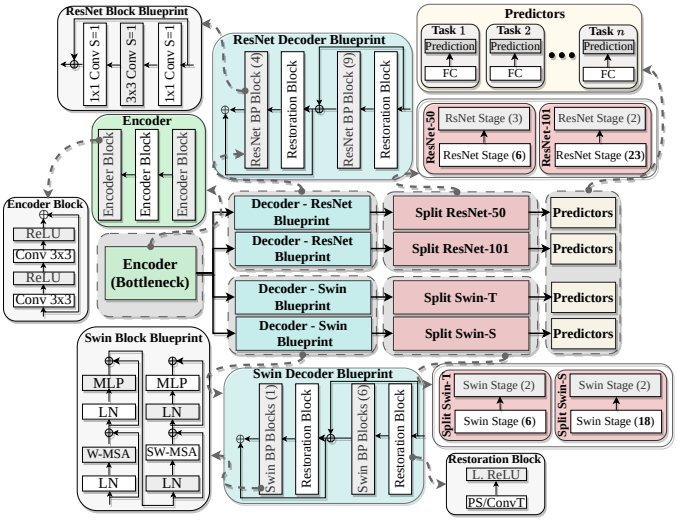

Fig. 10: Reference implementation of FrankenSplit

Figure 10 illustrates a reference implementation of FrankenSplit post-training with two blueprints applied to two variants. Creating blueprints is required only once for an architectural family. Boxes within the gray areas are separate instances (i.e., only one encoder), and boxes with the same name share an architecture. The rounded boxes outside organize layer views from coarse to fine-grained. We elaborate on how a single encoder can accommodate multiple decoder-backbone pairs in Section 5.3.4. The numbers in the parentheses refer to stage depth. Since the backbones are foundational models extensively trained on large datasets, we can naturally accommodate several downstream tasks by attaching separately trained predictors.

Blueprint instances replace a backbone's first two stages (i.e., the shallow layers) with two blueprint stages, taking a compressed representation as input instead of the original sample. The work by Liang et al. [56] inspires our approach to treat decoding as a restoration problem. Each stage comprises a restoration block and several blueprint (transformation) blocks, followed by a residual connection. The idea is to separate restoration (i.e., upsampling, ''smoothing'' quantized features) from transformation (i.e., matching the target representation regardless of encoder architecture). The restoration block is agnostic regarding the target architecture and optionally upsamples. The blueprint blocks induce the same bias as the target architectural family.

Two distinctions exist between the original blocks and their corresponding blueprint (transformation). First, the latter modifies operations not to reduce the latent spatial dimensions. Second, the embedding layer dimensions and stage depths may differ to reflect the resource asymmetry commonly found in MEC.

Although we should consider the resource asymmetry between the client and the server (i.e., by allocating more parameters to the decoder), there are limitations. Learning a function that can accurately retain necessary information is limited by the encoder's capacity (Section 4.1). Still, when end-to-end optimizing the compression model, it can benefit from increasing the decoder's capacity for restoration with diminishing returns.

Intuitively, we implement blueprints that result in decoder instances with, at most, the same execution time as the head of a target backbone. As a reminder, unlike most work in SC, we advocate keeping the execution time roughly equal on the server rather than reducing it. The encoder's responsibility is not to minimize the server load by executing shallow backbone layers. FrankenSplit treats the encoder entirely separate from the backbone. Besides dedicating the encoder exclusively to reducing transfer size, this separation of concern is necessary to accommodate several backbones with a single encoder instance.

5.3.4 Encoder Re-Usability

We argue that the representation of shallow layers generalizes well enough that it is possible to reuse compressor components. Consider the experiment illustrated in Figure 11,

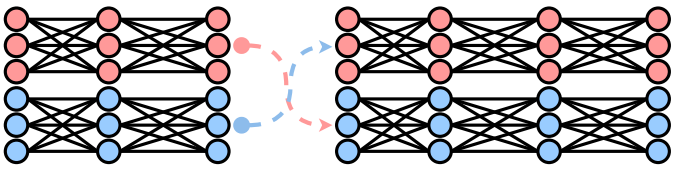

Fig. 11: Routing head outputs to different tails

where we split several backbones into head and tail models. The backbones are off-the-shelf models from *torch image models* (timm) [57] and pre-trained on the ImageNet [41] dataset. The head models consist of the initial embedding and shallow layers, i.e., the first two stages. The remaining layers comprise the substantially larger tails (roughly $2-5\%$ of total model parameters).

Then, we freeze the tail parameters and route the head output to all non-corresponding tails (e.g., ConvNeXt-T to Swin-T/S/B) and measure the accuracy every few iterations with a batch size of 128 as we finetune the head parameters using cross entropy loss. Each head-tail pair is a separate model built by attaching a copy of the head from one architecture to the tail of another. Where dimensions between

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

head and tail pairs do not match, we add a single 1×1 convolutional layer.

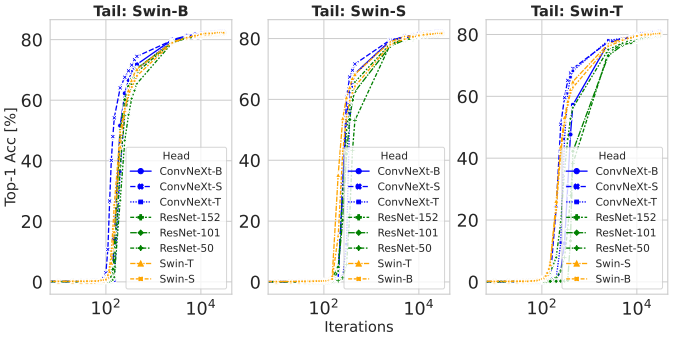

Fig. 12: Recovering Top-1 accuracy of rerouted heads

Figure 12 shows how rerouting the input between head models first (0 iterations) results in near 0% accuracy across all head-tail pairs. However, the concatenated models quickly converge near their original accuracy (roughly $80 - 83\%$) within just a few iterations (10100 iterations with 128 samples corresponding to one epoch on the ImagNet dataset). Notice that this holds regardless of whether the head-tail pairs are directly related to the modified network.

Therefore, if a compressor can sufficiently approximate the representation of just one head (i.e., the shallow layers of a network), it should be possible to accommodate arbitrary tails (i.e., the deeper layers of a network).

Crucially, applying the distortion measure in (10) or (11) does not result in an inherently different encoder behavior. Like training the compression model with a distortion measure from LIC, the purpose of the encoder is reducing uncertainty by decorrelating the data and discarding information. The distortion measure only controls what information an encoder should prioritize. Regardless of the target backbone's architecture, the encoder should decorrelate the input to reduce uncertainty. Conversely, the decoder seeks a mapping to the backbone's representation.

In other words, if we can map the latent to one representation, we can map it to any other with comparable information content. We can freeze the encoder and train various decoders to support arbitrary architectures once we train one compression model with a particular teacher as described in Figure 7. The blueprints facilitate an efficient transformation from the encoder's compressed representation to an input suitable for a particular backbone.

Notice that this method keeps the encoder parameters frozen, permitting us to deploy a single set of weights across all clients. Moreover, it does not modify the backbones at any step. After deployment, splitting is replaced with rerouting the input to a layer index (Section 5.3.2). Then, we can serve clients with the same models regardless of whether they applied the compressor.

6 Evaluation

6.1 Training & Implementation Details

We optimize our compression models initially on the 1.28 million ImageNet [41] training samples for 15 epochs, as described in section 5.1 and section 5.2, with some slight practical modifications for stable training. We aim to minimize bitrate without sacrificing predictive strength. Hence, we first seek the lowest β resulting in lossless prediction.

We use Adam optimization [58] with a batch size of 16 and start with an initial learning rate of $1 \cdot 10^{-3}$, then gradually lower it to $1 \cdot 10^{-6}$ with an exponential scheduler.

To implement our method, we use PyTorch [59], CompressAI [60] for entropy estimation and entropy coding, and pre-trained backbones from timm [57]. All baseline implementations and weights were either taken from CompressAI or the official repository of a baseline. To compute the saliency maps, we use a modified *XGradCAM* method from the library in [61] and include necessary patches in our repository. Lastly, to ensure reproducibility, we use torchdistill [62].

6.2 Experiment Setting

The experiments reflect the deployment strategies illustrated in Figure 5 and Figure 4. Ultimately, we must evaluate whether FrankenSplit enables latency-sensitive and performance-critical applications. Regardless of the particular task, a mobile edge client requires access to a DNN with high predictive strength on a server. Therefore, we must show whether FrankenSplit adequately solves two problems associated with offloading high-dimensional image data for real-time discriminative tasks. First, whether it considerably reduces the bandwidth consumption compared to existing methods without sacrificing predictive strength. Second, whether it improves inference times over various communication channels, i.e., it must remain competitive even when stronger connections are available.

Lastly, the evaluation should assess whether our method generalizes to arbitrary backbones. However, since it is infeasible to perform exhaustive experiments on all existing visual models, we focus on three well-known representatives and a subset of their variants instead. Namely, (i) ResNet [53] for classic residual CNNs. (ii) Swin Transformer [55] for hierarchical vision transformers, which are receiving increasing adaptation for a wide variety of vision tasks. (iii) ConvNeXt [63] for modernized state-of-the-art CNNs. Table 2 summarizes the relevant characteristics of the unmodified backbones subject to our experiments.

TABLE 2: Overview of Backbone Performance on Server

Backbone	Ratios	Params	Inference (ms)	Top-1 Acc. (%)
Swin-T	2:2:6:2	28.33M	4.77	81.93
Swin-S	2:2:18:2	49.74M	8.95	83.46
Swin-B	2:2:30:2	71.13M	13.14	83.88
ConvNeXt-T	3:3:9:3	28.59M	5.12	82.70
ConvNeXt-S	3:3:27:3	50.22M	5.65	83.71
ConvNeXt-B	3:3:27:3	88.59M	6.09	84.43
ResNet-50	3:4:6:3	25.56M	5.17	80.10
ResNet-101	3:4:23:3	44.55M	10.17	81.91
ResNet-152	3:8:36:3	60.19M	15.18	82.54

6.2.1 *Baselines*

Since our work aligns closest to learned image compression, we extensively compare FrankenSplit with learned and

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

handcrafted codecs applied to the input images, i.e., the input to the backbone is the distorted output. Comparing task-specific methods to general-purpose image compression methods may seem unfair. However, FrankenSplit's universal encoder has up to 260x less trainable parameters and further reduces overhead by not including side information or a sequential context model.

The naming convention for the learned baselines is the first author's name, followed by the entropy model. Specifically, we choose the work by Ballé et al. [17], [18] and Minnen et al. [19] for LIC methods since they represent foundational milestones. Complementary, we include the work by Cheng et al. [64] to demonstrate improvements with architectural enhancement.

As the representative for disregarding autoencoder size to achieve state-of-the-art r-d performance in LIC, we chose the work by Chen et al. [65] Their method differs from other LIC baselines by using a partially parallelizable context model, which trades off compression rate with execution time according to the configurable block size. We refer to such context models as Blocked Joint Hierarchical Priors and Autoregressive (BJHAP). Due to the large autoencoder, we found evaluating the inference time on constrained devices impractical when the context model is purely sequential and set the block size to 64x64. Additionally, we include the work by Lu et al. [66] as a milestone of the recent effort on efficient LIC with reduced autoencoders but only for latency-related experiments since we do not have access to the trained weights.

As a baseline for the state-of-the-art SC, we include the Entropic Student (ES) [22], [23]. The ES demonstrates the performance of directly applying a minimally adjusted LIC method for feature compression. One caveat is that we intend to show how FrankenSplit generalizes beyond CNN backbones, despite the encoder's simplistic CNN architecture. Although Matsubara et al. evaluate the ES on a wide range of backbones, most have no lossless configurations. Nevertheless, comparing bottleneck injection methods using different backbones is fair, as we found that the choice does not significantly impact the r-d performance (Section 6.3.5). Therefore, for an intuitive comparison, we choose ES with ResNet-50 using the same factorized prior entropy model as FrankenSplit.

We separate the experiments into two categories to assess whether our proposed method addresses the above-mentioned problems.

6.2.2 Criteria rate-distortion performance

We measure the bitrate in bits per pixel (bpp) because it permits directly comparing models with different input sizes. Choosing a distortion measure to draw meaningful and honest comparisons is challenging for feature compression.

Unlike evaluating reconstruction fidelity for image compression, PSNR or MS-SSIM does not provide intuitive results regarding predictive strength. Similarly, reporting absolute values (e.g., top-1 accuracy) gives an unfair advantage to experiments conducted on higher capacity backbones and veils the efficacy of a proposed method.

Hence, for a transparent evaluation, we determine the adversarial effects of codecs with image classification since it provides an unambiguous performance metric with established benchmark datasets. Specifically, we evaluate the distortion with the relative measure *predictive loss*, i.e., the drop in top-1 accuracy incurred by codecs. In particular, for SVBI methods, (near) lossless prediction implies that the reconstruction is a sufficient approximation for shallow features of an arbitrary feature extractor.

To ensure a fair comparison, we give the LIC and handcrafted baselines a grace threshold of 1.0% top-1 accuracy, to account for mitigating predictive loss incurred by codec artifacts [67]. For FrankenSplit, we set the threshold at 0.4%, reflecting the configuration with the lowest predictive loss of the ES. Note that, unlike the ES, FrankenSplit does not rely on finetuning the tail parameters of a backbone to improve r-d performance.

6.2.3 Measuring latency and overhead

To account for the resource asymmetry in MEC, we use NVIDIA Jetson boards[2] for representing capable but resource-constrained mobile clients. Contrastingly, the server hosts a powerful GPU. Table 3 summarizes the hardware we use in our experiments.

TABLE 3: Clients and Server Hardware Configuration

Device	Arch	CPU	GPU
Server	x86	16x Ryzen @ 3.4 GHz	RTX 3090
Client (TX2)	arm64x8	4x Cortex @ 2 GHz	Vol. 48 TC
Client (NX)	arm64x8	4x Cortex @ 2 GHz	Pas. 256 CC

6.3 Rate-Distortion Performance

We measure the predictive loss by the drop in top-1 accuracy from Table 2 using the ImageNet validation set for the standard classification task with 1000 categories. Analogously, we measure filesizes of the entropy-coded binaries to calculate the average bpp. To demonstrate that we can accommodate a non-CNN backbone with a CNN encoder, we start with a Swin-B implementation of FrankenSplit. Figure 13 shows r-d curves with the Swin-B backbone. The architecture of FrankenSplit-FP (FS-FP) and FrankenSplit-SGFP (FS-SGFP) are identical. We train both models with the loss functions derived in Section 5.1. The difference is that FS-SGFP is saliency-guided, i.e., FS-FP represents the pure HD training method and is an ablation to the saliency-guided distortion.

6.3.1 Effect of Saliency Guidance

Although FS-FP performs better than almost all other models, it is trained with the suboptimal objective discussed in Section 4.3. We identified the issue as overly skewing the objective needlessly towards the distortion term. Consequently, we proposed regularizing the distortion term by applying extracted saliency maps in Section 5.2 to improve the r-d performance. We favor Grad-CAM to compute the saliency maps over comparable methods for two reasons. First, it is generically applicable to arbitrary vision models. Second, it does not introduce additional tunable hyperparameters. The suboptimality of the unregularized objective is demonstrated by FS-SGFP outperforming FS-FP. By

2. nvidia.com/en-gb/autonomous-machines/embedded-systems/

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

simply guiding the distortion loss with saliency maps, we achieve a 25% lower bitrate without impacting predictive strength or additional runtime overhead.

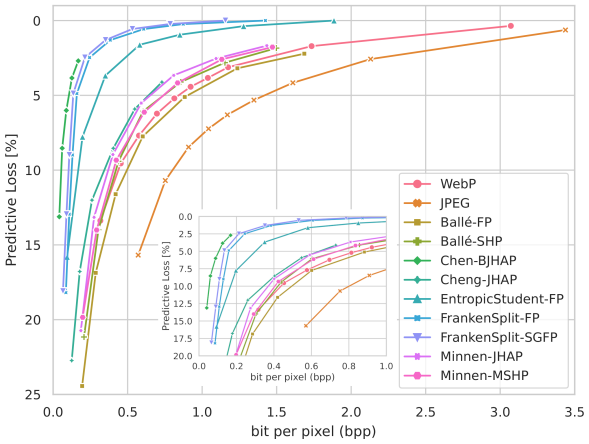

Fig. 13: Rate-distortion curve for ImageNet

6.3.2 Comparison to the ES

Even without saliency guidance, FS-FP consistently outperforms ES by a large margin. Specifically, FS-FP and FS-SGFP achieve 32% and 63% lower bitrates for the lossless configuration.

We ensured that our bottleneck injection incurs comparable overhead for a direct comparison to the ES. Moreover, the ES has an advantage due to finetuning tail parameters in an auxiliary training stage. Therefore, we attribute the performance gain to the more sophisticated architectural design decisions described in Section 5.3.

6.3.3 Comparison to Image Codecs

For almost all lossy codec baselines, Figure 13 illustrates that FS-(SG)FP has a significantly better r-d performance. Comparing FS-FP to Ballé-FP demonstrates the r-d gain of task-specific compression over general-purpose image compression. Although the encoder of FrankenSplit has 25x fewer parameters, both codecs use an FP entropy model with encoders consisting of convolutional layers. Yet, the average file size of FS-FP with a predictive loss of around 5% is 7x less than the average file size of Ballé-FP with comparable predictive loss.

FrankenSplit also beats modern general-purpose LIC without including any of their heavy-weight components. The only baseline FrankenSplit does not convincingly outperform is Chen-BJHAP. Nevertheless, in Section 3.4, we demonstrate that the incurred overhead offsets the compression gain disproportionately.

6.3.4 Image Codec Incurred Predictive Loss

For clarity, we separately evaluate r-d performance on the other backbones listed in Table 2 for FrankenSplit and baseline codecs.

Earlier, we argued that measuring PSNR is unsuitable to assess effects on downstream prediction. Since the image codecs are entirely decoupled from the predictive task, the bitrate is identical regardless of the backbone. We use this opportunity to plot PSNR instead of bpp against predictive loss in Figure 14.

Considering that compression models aggressively discard information, it is intuitive that the predictive loss is comparable across backbones. While some models handle distorted samples better, the difference in predictive loss is at most 3-5%. Still, the discrepancy demonstrates that PSNR is not a suitable measure for downstream tasks even within the same codec. More importantly, the discrepancy across baselines is considerably wider. For example, it is around 10% between Minnen-MSHP and Chen-BJHAP for lower PSNR levels.

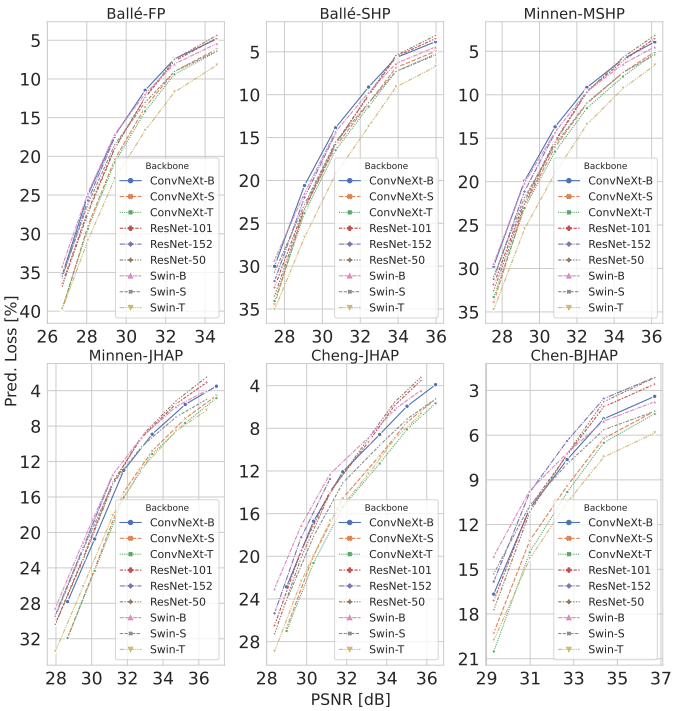

Fig. 14: Predictive Loss of baselines on multiple Backbones

6.3.5 Blueprints Generalization to Arbitrary Backbones

We now evaluate the r-d performance of other implementations of FrankenSplit to determine whether the blueprint heuristics generalize to arbitrary architectures. We create a decoder blueprint (Section 5.3.3) for each of the three architectural families (Swin, ResNet, and ConvNeXt). Then, we perform bottleneck injection at the layers before the deepest stage (Section 5.3.2), Figure 15 plots r-d performance of directly related architectures sharing the corresponding blueprint but with separately trained compressors. All models are trained as described in Figure 7. Across all architectural families, we observe similar r-d performance. The (near) lossless configurations of the largest backbones (Swin-B, ConvNeXt-B, ResNet-152) require around the same bpp, whereas smaller models tend to require 3-4% more bpp for comparable predictive loss.

Next, we conduct experiments to determine the importance of finding an adequate blueprint but assigning mismatching instances to a backbone. Table 4 summarizes the

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

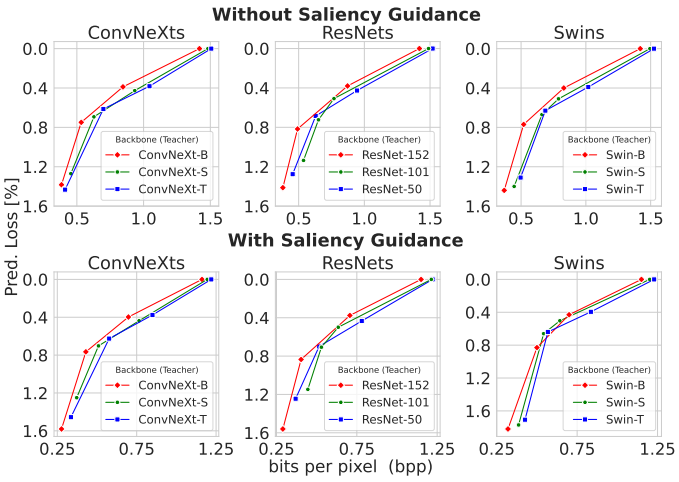

Fig. 15: Rate-distortion curve for various backbones

results for the largest backbones with varying decoder sizes. The Swin blueprint for the Swin-B decoder results in the FrankenSplit implementation from FS-FP from Figure 13. With 1% overhead in parameters, the compressor achieves 5.08 kB for 0.40% predictive loss. However, once we train compressors with ResNet or ConvNeXt restoration blocks, the r-d performance for the Swin-B is significantly worse when overhead is roughly equal. A blueprint that performs

TABLE 4: Effect of Mismatching Blueprints

Blueprint-Backbone	Params Overhead (%)	File Size (kB)	Pred. Loss (%)
ConvNext-Swin	0.96	19.05	2.49
		15.07	3.00
	2.86	14.46	2.53
		11.37	2.74
	5.89	12.53	1.36
		10.08	2.09
ResNet-Swim	1.03	22.54	0.82
		18.19	0.99
	2.73	16.32	0.81
		12.68	0.98
	5.25	13.89	0.79
		10.01	0.98

well for its intended target architecture results in substantially worse r-d performance for other architecture. Only increasing the decoder size brings the r-d performance closer to configurations that apply the appropriate blueprint.

From our findings, we can draw several conclusions. The r-d performance regarding the backbone network is near-agnostic. The implication is that the information content of the teachers (i.e., shallow layers) of varying architectures is comparable. We explain this by considering that we select the shallow layers as all layers preceding the deepest stage, which have comparable parameters across varying architectures.

Additionally, choosing a decoder architecture with the correct inductive bias (i.e., a blueprint) can transform compressed features significantly more efficiently.

6.3.6 Single Encoder with Multiple Backbones

We conduct a similar experiment as head rerouting from Section 5.3.4. However, we finetune the decoders instead of the head models.

We first select the compressors with (a near) lossless prediction from Figure 15 for each architectural family. Then, we choose the encoder from one of the compressors corresponding to the largest variants. Finally, we attach the decoders from the other compressors and finetune their parameters. We use unweighted head distillation and cross entropy (between the backbone classifier outputs and the hard labels) as the loss function. Analogous to the experiment in Section 5.3.4, we set the batch size as 128 and use PyTorch's Adam optimizer with a learning rate of $7{\cdot}10^{-5}$. To

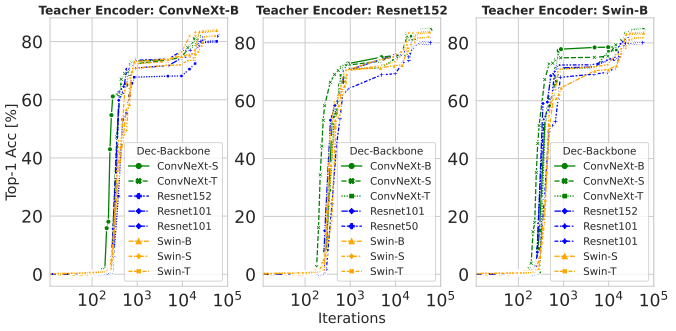

Fig. 16: Iterations to recover accuracy with decoder

demonstrate the limited importance of the initial teacher, we repeat this process for each of the three encoders separately and summarize the results in Figure 16. Note that the bitrate does not change due to freezing the encoder parameters. Hence, we report iterations until accuracy is restored to exemplify the similarity to the rerouting experiment in Section 5.3.4. We consider an accuracy restored if it is within $0.25 \pm 0.25\%$ of its original accuracy.

Besides requiring more iterations for convergence, the results are unsurprisingly similar to the head routing experiment outlined in Figure 12. Since we can infer from the earlier results that decoders can sufficiently approximate the head output, finetuning the decoder is near-equivalent to finetuning a head.

6.3.7 Generalization to multiple Downstream Tasks

Arguably, SVBI naturally generalizes to multiple downstream tasks due to approximating shallow features. We provide empirical evidence by evaluating the r-d performance of the compressors from Figure 13 without retraining the weights on different datasets.

Specifically, attach separate classifiers to the Swin-B backbone (as illustrated in Figure 10). Using PyTorch's Adam optimizer, we train each classifier for five epochs with no augmentation, a learning rate of $5 \cdot 10^{-5}$. A classifier refers to the last layers of a network.

For FrankenSplit-(SG)FP, we applied none or only rudimentary augmentation to evaluate how our method handles a type of noise it did not encounter during training. Hence, we include the Food-101 [68] dataset since it contains noise in high pixel intensities. Additionally, we include CIFAR-100 [42]. Lastly, we include Flower-102 [69] datasets to contrast more challenging tasks. The classifiers achieve an 87.73%, 88.01%, and 89.00% top-1 accuracy, respectively. Figure 17 summarizes the r-d curves for each task. Our method still demonstrates clear r-d performance gains over the baselines. More importantly, notice how FS-SGFP outperforms FS-FP on the r-d curve for the Food-101 dataset, with a comparable margin to the ImageNet

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

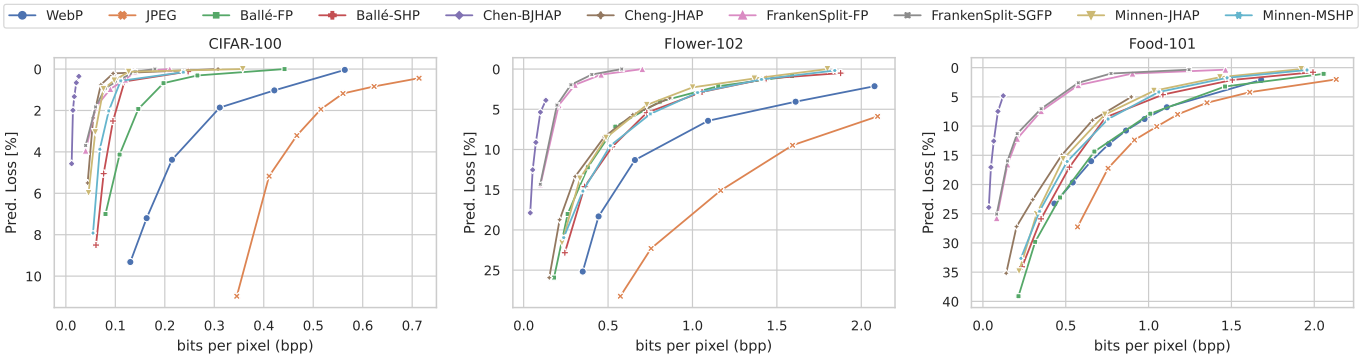

Fig. 17: Rate-distortion curve for multiple downstream tasks

dataset. Contrarily, on the Flower-102 datasets, there is less performance difference. Presumably, on simple datasets, the suboptimality of HD is less significant. Considering how easier tasks require less model capacity, the diminishing efficacy saliency guidance is consistent with our claims from Section 4.

6.3.8 *Effect of Tensor Dimensionality on R-D Performance*

Section 3.3 argues that measuring tensor dimensionality is inadequate to assess whether partial execution on the client is worthwhile.

To verify, we implement and train additional instances of FrankenSplit with the Swin-B backbone and show results in Figure 18 FS-SGFP(S) is the model with a small

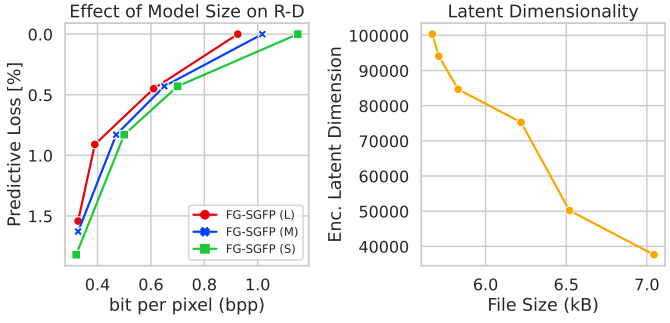

Fig. 18: Comparing effects on sizes

encoder ($\sim 140'000$ parameters) we have used for our previous results. FS-SGFP(M) and FS-SGFP(L) are medium and large models where we increased the (output) channels $C = 48$ to 96 and 128, respectively. Besides the number of channels, we've trained the medium and large models using the same configurations. On the left, we plot the r-d curves showing that increasing encode capacity naturally results in lower bitrates without additional predictive loss. For the plot on the right, we train further models with $C = \{48, 64, 96, 108, 120, 128\}$ using the configuration resulting in lossless prediction. Notice how increasing output channels will result in higher dimensional latent tensors $C \times 28 \times 28$ but inversely correlates to compressed file size. Arguably, increasing the encoder capacity will yield more powerful transforms to decorrelate the input.

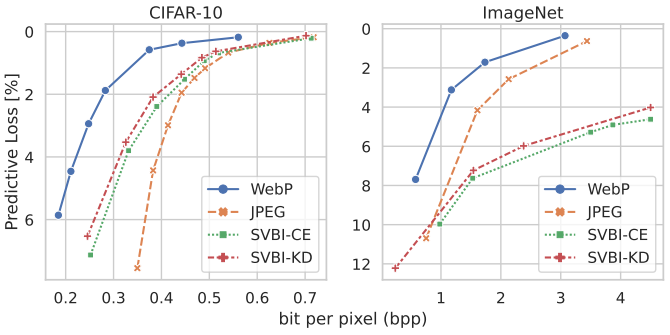

Fig. 19: Contrasting the r-d performance

6.3.9 *The Limitations of Direct Optimization for SVBI*

Section 5.1 mentioned that direct optimization does not work for SVBI as it does for DVBI, where the bottleneck is at the penultimate layer. Specifically, it performs incomparably worse than HD despite the latter's inherent suboptimality. We demonstrate this by applying the SVBI-CE and SVBI-KD objective on the CIFAR-10 [42] and ImageNet dataset. All models are identical and trained with the setup in Section 6.1, except we train for more epochs to account for slower convergence.

Figure 19 summarizes the results the results. On CIFAR-10, SVBI-CE and SVBI-KD yield moderate performance gain over JPEG. Yet, they perform substantially worse on ImageNet.

Sufficiency as a necessary precondition may explain why the objective in (5) does not yield good results when the bottleneck is at a shallow layer, as the mutual information $I(Y;\tilde{Y})$ is not adequately high. Since the representation of the last hidden and shallow layer are so far apart in the information path, there is insufficient information to minimize $\mathcal{D}(H;\tilde{H})$. The compression model approximates the intermediate representation for a simple classification task to minimize predictive loss by incurring higher bitrates. Consequently, for the challenging ImageNet classification task, the same method incurs significant predictive loss even when skewing the r-d objective heavily towards high bitrates.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3381952

6.4 Prediction Latency and Overhead

We exclude entropy coding from our measurement, since not all baselines use the same entropy coder. For brevity, the results implicitly assume the Swin-B backbone for the remainder of this section. Inference times with other backbones for FrankenSplit can be derived from Table 5. Analogously, the inference times of applying LIC models for different unmodified backbones can be derived using Table 2. Notably, the relative overhead decreases the larger the tail is, which is favorable since we target inference from more accurate predictors.

TABLE 5: Execution Times of FS (S) with Various Backbones

Backbone	Overhead Prams (%)	Inf. Server+NX (m/s)	Inf. Server+TX2 (m/s)
Swin-T	2.51	7.83	9.75
Swin-S	1.41	11.99	13.91
Swin-B	1.00	16.12	18.04
ConvNeXt-T	3.46	6.83	8.75
ConvNeXt-S	1.97	8.50	10.41
ConvNeXt-B	0.90	9.70	11.62
ResNet-50	3.50	13.16	10.05
ResNet-101	2.01	8.13	15.08
ResNet-152	1.48	18.86	20.78

6.4.1 Computational Overhead

We first disregard network conditions to get an overview of the computational overhead of applying compression models. Table 6 summarizes the execution times of the prediction pipeline's components. Enc. NX/TX2 refers to the encoding time on the respective client device. Analogously, dec. refers to the decoding time at the server. Lastly, Full NX/TX2 is the total execution time of encoding at the respective client plus decoding and the prediction task at the server. Lu-JHAP demonstrates how LIC models without a sequential context component are noticeably faster but are still 9.3x-9.6x slower than FrankenSplit despite a considerably worse r-d performance. Notice that the computational load of FrankenSplit is near evenly distributed between the client and the server. The significance of considering resource asymmetry is emphasized by how the partially parallelized context model of Chen-BJHAP leads to faster decoding on the server. Nevertheless, it is slower than other JHAP baselines due to the overhead of the increased encoder size outweighing the performance gain of the blocked context model on constrained hardware.

TABLE 6: Inference Pipeline Components Execution Times

Model	Prams Enc./Dec.	Enc. [NX/TX2] (ms)	Dec. (ms)	Full [NX/TX2] (ms)
FrankenSplit	**0.14M/ 2.06M**	**2.92/ 4.87**	2.00	**16.34/ 18.29**
Ballé-FP	3.51M/ 3.51M	27.27/ 48.93	**1.30**	41.71/ 63.37
Ballé-SHP	8.30M/ 5.90M	28.16/ 50.89	1.51	42.81/ 65.54
Minnen-MSHP	14.04M/ 11.65M	29.51/ 52.39	1.52	44.17/ 67.05
Minnen-JHAP	21.99M/ 19.59M	4128.17/ 4789.89	275.18	4416.7/ 5078.2
Cheng-JHAP	16.35M/ 22.27M	2167.34/ 4153.95	277.26	2457.7/ 4444.3
Lu-JHAP	5.28M/ 4.37M	2090.88/ 5011.56	352.85	2456.8/ 5377.8
Chen-BJHAP	36.73M/ 28.08M	3111.01/ 5837.38	43.16	3167.3/ 5893.6

6.4.2 Competing against Offloading

The average compressed filesize gives the transfer size from the ImageNet validation set. Using the transfer size, we evaluate transfer time on a broad range of standards. Since we did not include the execution time of entropy coding for learned methods, the encoding and decoding time for the handcrafted codecs is set to 0. The setting favors the baselines because both rely on sequential CPU-bound transforms. Table 7 summarizes how our method performs in various standards. Due to space constraints, we only include LIC models with the lowest request latency (Minnen-MSHP) or the lowest compression rate (Chen-BJHAP). Still, with Table 6 and the previous results, we can infer that the LIC baselines have considerably higher latency than FrankenSplit.

TABLE 7: Total Latency with Various Wireless Standards

Standard/ Data Rate (Mbps)	codec	Transfer (ms)	Total [TX2] (ms)	Total [NX] (ms)
BLE/ 0.27	FS-SGFP (0.23)	142.59	160.48	158.53
	FS-SGFP (LL)	209.89	227.78	225.83
	Minnen-MSHP	348.85	415.89	393.01
	Chen-BJHAP	40.0	6167.79	3441.41
	WebP	865.92	879.06	879.06
	PNG	2532.58	2545.72	2545.72
4G/ 12.0	FS-SGFP (0.23)	3.21	21.09	19.15
	FS-SGFP (LL)	4.72	22.61	20.66
	Minnen-MSHP	7.85	74.89	52.01
	Chen-BJHAP	0.9	6128.69	3402.31
	WebP	19.48	32.63	32.63
	PNG	56.98	70.13	70.13
Wi-Fi/ 54.0	FS-SGFP (0.23)	0.71	18.6	16.65
	FS-SGFP (LL)	1.05	18.93	16.99
	Minnen-MSHP	1.74	68.78	45.9
	Chen-BJHAP	0.2	6127.99	3401.61
	WebP	4.33	17.47	17.47
	PNG	12.66	25.81	25.81
5G/ 66.9	FS-SGFP (0.23)	0.58	18.46	16.51
	FS-SGFP (LL)	0.85	18.73	16.78
	Minnen-MSHP	1.41	68.44	45.56
	Chen-BJHAP	0.16	6127.95	3401.57
	WebP	3.49	16.64	16.64
	PNG	10.22	23.36	23.36

Generally, the more constrained the network is the more we can benefit from reducing the transfer size. In particular, FrankenSplit is up to 16x faster in highly constrained networks, such as BLE. Conversely, offloading with fast handcrafted codecs may be preferable in high-bandwidth environments. Yet, FrankenSplit is significantly better than offloading with PNG, even for 5G. Figure 20 plots the inference latencies against handcrafted codecs using the NX client. For stronger connections, such as 4G LTE, it is still 3.3x faster than using PNG. Nevertheless, compared to WebP, offloading seems more favorable when bandwidth is high. Still, this assumes that the rates do not fluctuate and that the network can seamlessly scale for an arbitrary number of client connections. Moreover, we did not apply any optimizations to the encoder.

7 Conclusion

This work introduced a novel lightweight compression framework to facilitate critical MEC applications relying on large DNNs. We showed that a minimalistic implementation

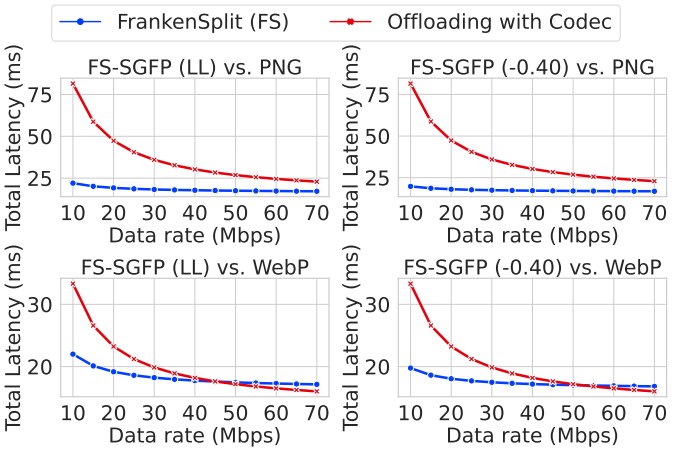

Fig. 20: Comparing effects on sizes

of our design heuristic is sufficient to outperform numerous baselines. However, there are several limitations. We emphasize that the primary insight of the reported results is the potential of adequate distortion measures and regularization methods for neural feature compression. Despite significantly improving rate-distortion performance, better methods may exist to extract saliency maps. Moreover, the Factorized Prior entropy model does not discriminate between inputs. Although side information with hypernetworks taken from LIC trivially improves rate-distortion performance, our results show that it may not be a productive approach to repurpose existing image compression methods directly. Hence, conceiving an efficient way to include task-dependent side information is a promising direction.

REFERENCES

[1] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, *et al.*, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[2] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[3] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, and L. Guo, "Computation offloading in mobile edge computing networks: A survey," *Journal of Network and Computer Applications*, p. 103366, 2022.

[4] T. Rausch, W. Hummer, C. Stippel, S. Vasiljevic, C. Elvezio, S. Dustdar, and K. Krösl, "Towards a platform for smart city-scale cognitive assistance applications," in *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 330–335, 2021.

[5] R. R. Arinta and E. Andi W.R., "Natural disaster application on big data and machine learning: A review," in *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pp. 249–254, 2019.

[6] Q. Xin, M. Alazab, V. G. Díaz, C. E. Montenegro-Marin, and R. G. Crespo, "A deep learning architecture for power management in smart cities," *Energy Reports*, vol. 8, pp. 1568–1577, 2022.

[7] U. Cisco, "Cisco annual internet report (2018–2023) white paper," *Cisco: San Jose, CA, USA*, vol. 10, no. 1, pp. 1–35, 2020.

[8] Q. Zhang, X. Li, X. Che, X. Ma, A. Zhou, M. Xu, S. Wang, Y. Ma, and X. Liu, "A comprehensive benchmark of deep learning libraries on mobile devices," in *Proceedings of the ACM Web Conference 2022*, pp. 3298–3307, 2022.

[9] Q. Zhang, X. Che, Y. Chen, X. Ma, M. Xu, S. Dustdar, X. Liu, and S. Wang, "A comprehensive deep learning library benchmark and optimal library selection," *IEEE Transactions on Mobile Computing*, 2023.

[10] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 397–411, USENIX Association, July 2021.

[11] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[12] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," 2000.

[13] Y. Yang, S. Mandt, and L. Theis, "An introduction to neural data compression," 2022.

[14] G. LLC, "An image format for the web."

[15] V. Goyal, "Theoretical foundations of transform coding," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 9–21, 2001.

[16] J. Ballé, P. A. Chou, D. Minnen, S. Singh, N. Johnston, E. Agustsson, S. J. Hwang, and G. Toderici, "Nonlinear transform coding," *CoRR*, vol. abs/2007.03034, 2020.

[17] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[18] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," 2018.

[19] D. Minnen, J. Ballé, and G. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," 2018.

[20] S. Singh, S. Abu-El-Haija, N. Johnston, J. Ballé, A. Shrivastava, and G. Toderici, "End-to-end learning of compressible features," *CoRR*, vol. abs/2007.11797, 2020.

[21] Y. Dubois, B. Bloem-Reddy, K. Ullrich, and C. J. Maddison, "Lossy compression for lossless prediction," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14014–14028, 2021.

[22] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "SC2 Benchmark: Supervised Compression for Split Computing," *Transactions on Machine Learning Research*, 2023.

[23] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "Supervised compression for resource-constrained edge computing systems," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2685–2695, 2022.

[24] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[25] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*, pp. 671–678, IEEE, 2018.

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

[27] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, pp. 1–24, 2022.

[28] H. Liu, W. Zheng, L. Li, and M. Guo, "Loadpart: Load-aware dynamic partition of deep neural networks for edge offloading," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 481–491, 2022.

[29] A. Bakhtiarnia, N. Milošević, Q. Zhang, D. Bajović, and A. Iosifidis, "Dynamic split computing for efficient deep edge intelligence," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2023.

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

[31] J. Shao and J. Zhang, "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)*, pp. 1–6, 2020.

[32] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "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*, HotEdgeVideo'19, (New York, NY, USA), p. 21–26, Association for Computing Machinery, 2019.

[33] M. Sbai, M. R. U. Saputra, N. Trigoni, and A. Markham, "Cut, distil and encode (cde): Split cloud-edge deep inference," in *2021 18th*

Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 1–9, 2021.

[34] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[35] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[36] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[37] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Infaas: Automated model-less inference serving.," in *USENIX Annual Technical Conference*, pp. 397–411, 2021.

[38] K. Zhao, Z. Zhou, X. Chen, R. Zhou, X. Zhang, S. Yu, and D. Wu, "Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale," *IEEE Transactions on Mobile Computing*, pp. 1–16, 2022.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[40] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[42] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[44] C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," in *IRE National Convention Record, 1959*, vol. 4, pp. 142–163, 1959.

[45] T. Berger, "Rate distortion theory for sources with abstract alphabets and memory," *Information and Control*, vol. 13, no. 3, pp. 254–273, 1968.

[46] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 ieee information theory workshop (itw)*, pp. 1–5, IEEE, 2015.

[47] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[48] L. Wang and K.-J. Yoon, "Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[49] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[50] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

[51] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, pp. 336–359, oct 2019.

[52] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learniserverng for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[54] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[55] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

[56] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, "Swinir: Image restoration using swin transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1833–1844, 2021.

[57] R. Wightman, "Pytorch image models." https://github.com/rwightman/pytorch-image-models, 2019.

[58] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[59] A. D. I. Pytorch, "Pytorch," 2018.

[60] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "Compressai: a pytorch library and evaluation platform for end-to-end compression research," *arXiv preprint arXiv:2011.03029*, 2020.

[61] J. Gildenblat and contributors, "Pytorch library for cam methods." https://github.com/jacobgil/pytorch-grad-cam, 2021.

[62] Y. Matsubara, "torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation," in *International Workshop on Reproducible Research in Pattern Recognition*, pp. 24–44, Springer, 2021.

[63] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

[64] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," 2020.

[65] T. Chen, H. Liu, Z. Ma, Q. Shen, X. Cao, and Y. Wang, "End-to-end learnt image compression via non-local attention optimization and improved context modeling," *IEEE Transactions on Image Processing*, vol. 30, pp. 3179–3191, 2021.

[66] M. Lu, P. Guo, H. Shi, C. Cao, and Z. Ma, "Transformer-based image compression," in *2022 Data Compression Conference (DCC)*, pp. 469–469, 2022.

[67] X. Luo, H. Talebi, F. Yang, M. Elad, and P. Milanfar, "The rate-distortion-accuracy tradeoff: Jpeg case study," *arXiv preprint arXiv:2008.00605*, 2020.

[68] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pp. 446–461, Springer, 2014.

[69] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, 2008.

Alireza Furutanpey received a MSc from the Technical University of Vienna, Austria in 2022 with distinction in the field of Computer Science. He is now a PhD candidate at the Distributed Systems Group in the field of Edge Computing. His research interests include Mobile Edge Computing, Edge Intelligence and Machine Learning.

Philipp Raith received a MSc from the Technical University of Vienna, Austria in 2021 with distinction in the field of Computer Science. He is now a PhD candidate at the Distributed Systems Group in the field of Edge Computing. His research interests include Serverless Edge Computing, Edge Intelligence and Operations for AI.

Schahram Dustdar is a full professor of computer science and heads TU Wien's Distributed Systems Group. His research interests include distributed systems, Edge Intelligence, complex and autonomic software systems. He's the editor in chief of Computing; associate editor of ACM Transactions on the Web, ACM Transactions on Internet Technology, IEEE Transactions on Cloud Computing, and IEEE Transactions on Services Computing. He's also on the editorial boards of IEEE Internet Computing and IEEE Computer. He has received the ACM Distinguished Scientist award and Distinguished Speaker Award and the IBM Faculty Award. He is an elected member of Academia Europaea, where he's was Informatics Section chairman from 2015 to 2022. He is an IEEE Fellow and AAIA Fellow where he is the current President.